Hybrid CSP & Global Optimization

Michel RUEHER

University of Nice Sophia-Antipolis / I3S – CNRS, France

Courtesy to Alexandre Goldsztejn, Yahia Lebbah, Claude Michel

June, 2011

ACP Summer School
"Hybrid Methods for Constraint Programming"
Turunç

Outline

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**

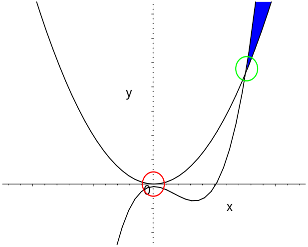
  Mainly rely on interval computation
  ... available systems (e.g., Globsol) are **quite slow**

- **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 0 as the minimum …

Branch and Bound Algorithm (1)

- **BB Algorithm –Scheme**
  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**

Branch and bound algorithm (2)

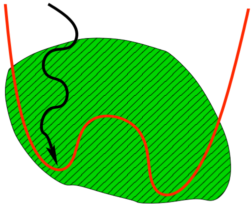
Function $\texttt{BB}(\texttt{IN}\ \mathbf{x}, \epsilon; \texttt{OUT}\ \mathcal{S}, [L, U])$

$\mathcal{S}$: set of proven feasible points
$\mathbf{f_x}$ denotes the set of possible values for f in $\mathbf{x}$
nbStarts: number of starting points in the first upper-bounding

$\mathcal{L} := \{\mathbf{x}\}; \quad (L, U) := (-\infty, +\infty);$
$\mathcal{S} := \mathit{UpperBounding}(\mathbf{x}', \mathit{nbStarts});$
while $w([L, U]) > \epsilon \wedge \mathcal{L} \neq \emptyset$ **do**
  $\mathbf{x}' := \mathbf{x}''$ such that $\underline{\mathbf{f}}_{\mathbf{x}''} = \mathit{min}\{\underline{\mathbf{f}}_{\mathbf{x}''} : \mathbf{x}'' \in \mathcal{L}\}; \quad \mathcal{L} := \mathcal{L} \setminus \{\mathbf{x}\}';$
  $\bar{\mathbf{f}}_{\mathbf{x}'} := \mathit{min}(\bar{\mathbf{f}}_{\mathbf{x}'}, U);$
  $\mathbf{x}' :=$ **Prune**$(\mathbf{x}');$
  $\underline{\mathbf{f}}_{\mathbf{x}'} :=$ **LowerBound**$(\mathbf{x}');$
  $\mathcal{S} := \mathcal{S} \cup$ **UpperBounding**$(\mathbf{x}', 1);$
  if $\mathbf{x}' \neq \emptyset$ **then** $(\mathbf{x}'_1, \mathbf{x}'_2) := \mathit{Split}(\mathbf{x}'); \ \mathcal{L} := \mathcal{L} \cup \{\mathbf{x}'_1, \mathbf{x}'_2\};$
  if $\mathcal{L} \neq \emptyset$ **then** $(L, U) := (\mathit{min}\{\underline{\mathbf{f}}_{\mathbf{x}''} : \mathbf{x}'' \in \mathcal{L}\}, \mathit{min}\{\bar{\mathbf{f}}_{\mathbf{x}''} : \mathbf{x}'' \in \mathcal{S}\})$
endwhile

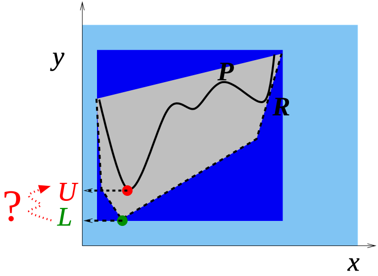
Computing "sharp" upper bounds

- **Upper bounding**
  - **local search**
    $\rightarrow$ approximate feasible point x_{approx}
  - **epsilon inflation process and proof**
    $\rightarrow$ 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)

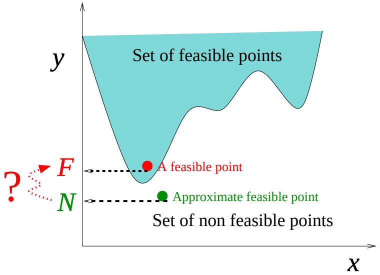
Using the lower bound to get an upper-bound



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

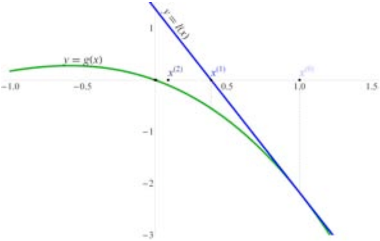
Handling square systems of equations

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

  $\longrightarrow$ 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**



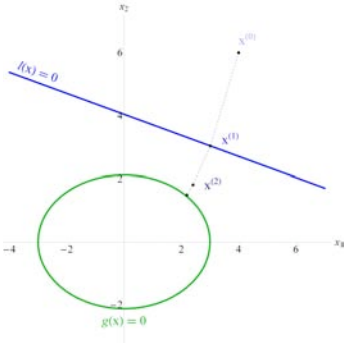
Handling under-constrained systems of equations

Manifold of solutions
$\longrightarrow$ linear system $l(x) = 0$ is under-constrained
$\longrightarrow$ **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 $||x^{(1)} - x^{(0)}||$)

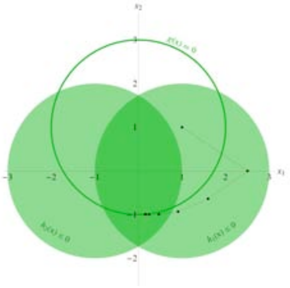


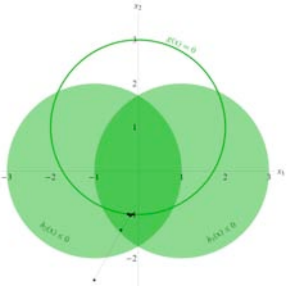
Handling under-constrained systems of equations and inequalities

- Under-constrained systems of equations and **inequalities**
  ⇝ 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**

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**

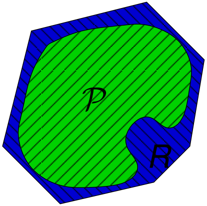
Lower bounding

- **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 $\longrightarrow \underline{\mathbf{f}}^*$

$\longrightarrow$ 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

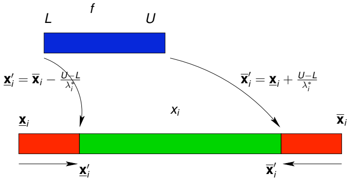
  $$\mathbf{x_i} \geq \underline{\mathbf{x}}'_\mathbf{i} \text{ with } \underline{\mathbf{x}}'_\mathbf{i} = \overline{\mathbf{x}}_\mathbf{i} - \frac{\mathbf{U} - \mathbf{L}}{\lambda_\mathbf{i}^*}$$

  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**

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

  has no solution when the domain of x is set to $[\underline{\mathbf{x}}_i, \underline{\mathbf{x}}'_i]$, **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 !!

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

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**:

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

   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
$\longrightarrow$ 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 $n' = \mathcal{C}ard(\mathcal{C}onflict(p_v))$

- $\longrightarrow$ if p_v is installed, none of the p_v conflicting packages can be installed
- $\longrightarrow$ if p_v is not installed, then the conflicting packages can freely be either installed or not

MILP: handling multi criteria (1)

Assume the following n criteria:

$$min \sum_{i=1}^{m} c_i^1.x_i,\ \ldots,\ min \sum_{i=1}^{m} c_i^n.x_i$$

considered in a lexical order.

To solve them using a mono criteria optimiser, we can:

1. use a **linear combination** of the criteria
2. **sequentialy solving**
   - $o_1 = min \sum_{i=1}^{m} c_i^1.x_i \;\; s.t. Ct,$
   - then $o_2 = \sum_{i=1}^{m} c_i^2.x_i \;\; s.t. \sum_{i=1}^{m} c_i^1.x_i <= o_1, Ct,$

   $\ldots$

MILP model: handling multi criteria (2)

Paranoid

- **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.

Examples of optimization criteria

- **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.

Running different criteria combinations

rand915 from sarge-etch-lenny set with ***mccs***

Combination	Time (s)	Removed	Notuptodate	Unsat
no criteria	**2.24**	**810**	129	40
lexicographic	37.98	**47**	435	**195**
lexsemiagregate	19.52	47	435	47
leximax	**238.35**	133	132	100
agregate	19.38	233	**31**	**18**

with ***Cplex (12)*** on a T7700 @ 2.40GHz laptop running linux

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**
  $\longrightarrow$ Good performance for one or two criteria
  $\longrightarrow$ Available in the *experimental version of* **apt-get**, debian package manager
- **Homework** : find a nice and efficient **CP model** :)