Constraint Programming over Continuous Domains

Michel RUEHER

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

June, 2011

ACP Summer School

"Hybrid Methods for Constraint Programming"

Turunç

Outline

Basics on Intervals

Local consistencies

Relations between 2B, Box and 3B consistencies

Implementation issues

QUAD*: a global constraint*

Search

Interval arithmetic: notations

$c_j(x_1, \ldots, x_n)$	A relation over the real numbers
x, y	Real variables or vectors
X or $\mathbf{x}$ or D_x	The domain of variable x (i.e. intervals)
$X = [\underline{X}, \overline{X}]$	The set of real numbers x verifying
$\mathbf{x} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$	$\underline{X} \leq x \leq \overline{X}$ (resp. $\underline{\mathbf{x}} \leq x \leq \overline{\mathbf{x}}$)
$\mathcal{C}$	The set of constraints
$\mathcal{D}$	The set of domains of all the variables
$\mathcal{R}$	The set of real numbers
$\mathcal{R}^\infty$	$\mathcal{R} \cup \{-\infty, +\infty\}$, set of real numbers extended with infinity symbols
$\mathcal{F}$	The set of floating point numbers
a^+ (resp. a^-)	The smallest (resp. largest) number of $\mathcal{F}$ strictly greater (resp. lower) than a
$\tilde{k}$	smallest interval containing real number k
$\Phi_{cstc}(P)$	closure (filtering) by consistency of CSP P $cstc$ stands for $2B, Box, 3B, Bound$

Natural interval extension (1)

- In general, it is not possible to compute **the exact enclosure** of the range for an arbitrary function over the real numbers

  $\rightarrow$ The interval extension of a function is an interval function that computes an **outer approximation** of the range of the function over a domain

Natural interval extension (2)

F the **natural** interval extension of a real function f is obtained by replacing:

- Each constant k by its natural interval extension $\tilde{k}$
- Each variable by a variable over the intervals
- Each mathematical operator in f by its **optimal** interval extension

Optimal extensions for basic operators:

- $[a, b] \ominus [c, d] = [a - d, b - c]$
- $[a, b] \oplus [c, d] = [a + c, b + d]$
- $[a, b] \otimes [c, d] =$
  $[\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$
- $[a, b] \oslash [c, d] = [\min(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}), \max(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d})]$
  if $0 \notin [c, d]$
  otherwise $\rightarrow [-\infty, +\infty]$

Interval extension: properties

- If $\mathbf{0} \notin \mathbf{F}(\mathbf{X})$, then no value exists in the box **X** such that $f(X) = 0$ $\longrightarrow$
  Equation $f(x)$ does not have any root in the box X
- Interval arithmetic can be implemented taking into account **round-off errors**
- No monotonicity but interval arithmetic preserves **inclusion monotonicity**: $Y \subseteq X \quad \Rightarrow \quad F(Y) \subseteq F(X)$
- No distributivity but interval arithmetic is **sub-distributive**: $X(Y + X) \subseteq XY + XZ$

Properties of Natural interval extension (2)

- Let be $F : \mathcal{I}^n \rightarrow \mathcal{I}$ the natural interval extension of $f : \mathcal{R}^n \rightarrow \mathcal{R}$ and
  $f_{sol} = \Box\{f(v_1, \ldots, v_n) \mid v_1 \in I_1, \ldots, v_n \in I_n\}$
  f_{sol}: **Hull consistency**

  If each variable has **only one occurrence** in f

  then $f_{sol} \equiv F(I_1, \ldots, I_n)$
  else $f_{sol} \subseteq F(I_1, \ldots, I_n)$

- Let be $C : \mathcal{I}^n \rightarrow \mathcal{B}ool$ the natural interval extension of equation $c : \mathcal{R}^n \rightarrow \mathcal{B}ool$

  If each variable has **only one occurrence** in c, then :
  $C(D_1, \ldots, D_n) \Leftrightarrow$
  $(\exists x_1 \in D_1, \ldots, \exists x_n \in D_n \mid c(x_1, \ldots, x_n))$

Local consistencies (1)

- Informally speaking, a constraint system C satisfies a partial consistency property if **a relaxation of C is consistent**
- Consider $X = [\underline{x}, \overline{x}]$ and $C(x, x_1, \ldots, x_n) \in \mathcal{C}$: if $C(x, x_1, \ldots, x_n)$ does not hold for any values $a \in [\underline{x}, x']$, then X may be shrunken to $X = [x', \overline{x}]$

2B–consistency (1)

Variable x is 2B–consistency for constraint $f(x, x_1, \ldots, x_n) = 0$ if the lower (resp. upper) bound of the domain X is the smallest (resp. largest) solution of $f(x, x_1, \ldots, x_n)$

Definition: 2B–consistency

Let $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP and $C \in \mathcal{C}$ a k-ary constraint over $(X_1, \ldots, X_k)$

C is 2B–consistency iff :

$$\forall i, X_i = \Box\{\tilde{x}_i \mid \exists \tilde{x}_1 \in X_1, \ldots, \exists \tilde{x}_{i-1} \in X_{i-1}, \exists \tilde{x}_{i+1} \in X_{i+1}, \ldots, \exists \tilde{x}_k \in X_k \;:\; c(\tilde{x}_1, \ldots, \tilde{x}_{i-1}, \tilde{x}_i, \tilde{x}_{i+1} \ldots, \tilde{x}_k)\}$$

A CSP is 2B–consistent iff all its constraints are consistent

Relation between 2B-Consistency and the natural interval extension of a constraint

Let be $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ CSP such that constraints $\mathcal{C}$ do not contain multiple occurrences of variables in $\mathcal{X}$,

Let be $c \in \mathcal{C}$ a k-ary constraint defined by over $(x_1, \ldots, x_k)$,

c is **2B-Consistent** iff $\forall x_i \in \{x_1, \ldots, x_k\}$, with $D_{x_i} = [a, b]$, the following relations hold :

- $C(D_{x_1}, \ldots, D_{x_{i-1}}, [a, a^+), D_{x_{i+1}}, \ldots, D_{x_k})$
- $C(D_{x_1}, \ldots, D_{x_{i-1}}, (b^-, b], D_{x_{i+1}}, \ldots, D_{x_k})$

where $[a, a^+)$ and $(b^-, b]$ are semi-open intervals

Arc-consistency versus 2B-Consistency

The 2B-consistency is a **weaker** consistency than arc consistency:

- Arc consistency enforces a condition on **all elements** of the domains
- 2B-Consistency only enforces a condition **on the bounds** of the interval (i.e. the domain)

Example

$P: \ \mathcal{C} = \{x = y^2\}$ with $D_x = [1, 4], \ D_y = [-2, +2]$

P is **2B-Consistent but not Arc-consistent** since value $0 \in D_y$ does not have any **support** in D_x

2B-consistency filtering(1)

Algorithms that achieve 2B-consistency filtering are based upon projection functions

Analytic functions always exist when the variable to express in terms of the others appears only once in the constraint
$\longrightarrow$ considers that **each occurrence is a different new variable**
$\longrightarrow$ initial constraints are decomposed into "primitive" constraints

Decomposition of a constraint system (1)

- The decomposition of constraint $\mathcal{C}$ in a set of primitives constraints *decomp*$(\mathcal{C})$ does not change the semantics: $\mathcal{C}$ and *decomp*$(\mathcal{C})$ the same solutions
- The **scope** of the verification done by 2B–filtering algorithms is **reduced** by the decomposition: if variable x has multiple occurrences in constraint $c \in \mathcal{C}$, then the different occurrences of x in *decomp*(c) may take **different values**

$\longrightarrow$ 2B–filtering of *decomp*$(\mathcal{C})$ will be **weaker** than 2B–filtering of $\mathcal{C}$

Decomposition of a constraint system (2)

Example:

Let be $c\ : x_1 + x_2 - x_1 = 0$ with $D_{x_1} = [-1, 1],\ D_{x_2} = [0, 1]$ with $D_{x_3} = [-1, 1]$
$\rightarrow$ *decomp*$(c) = \{x_1 + x_2 - x_3 = 0, x_1 = x_3\}$

Each projection function of *decomp*(c) can be computed with operations of the interval calculation

Constraint c is not 2B-consistent since $x_2 = 1$ does not have any support
...whereas *decomp*(c) **is 2B-consistent**: $x_1 = -1$ and $x_3 = 0$ satisfy $x_1 + x_2 - x_3 = 0$ when $x_2 = 1$

Early stopping of the propagation algorithm

In case of **asymptotic convergence**, it is not realistic to try to reduce the intervals until no more floating point number can be removed !

$\longrightarrow$ **To Stop the propagation** before reaching the fixed point

Example of slow convergence

Let be :

$X = 2 \times Y$
$Y = X$
$D_X = [-10, 10], D_Y = [-10, 10]$

2B-consistency will make the following reductions:

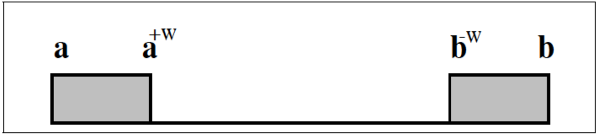
$D_Y = [-5, 5]$	$D_X = [-5, 5]$
$D_Y = [-2.5, 2.5]$	$D_X = [-2.5, 2.5]$
$D_Y = [-1.25, 1.25]$	$D_X = [-1.25, 1.25]$
$D_Y = [-0.625, 0.625]$	$D_X = [-0.625, 0.625]$

......

... better to stop propagation before reaching the fixed point !

"Width" of the bound

a^{+w} stands for $(w+1)^{th}$ float after a
a^{-w} stands for $(w+1)^{th}$ float before a



2B(w)–Consistency

Let be $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ a CSP, $x \in \mathcal{X}$, $D_x = [a, b]$, w a positive integer D_x is **2B(w)–Consistent** if for each constraint $c(x, x_1, \ldots, x_k)$ in $\mathcal{C}$:

$$\exists v \in [a, a^{+w}),\ \exists v_1, \ldots, v_k \in D_{x_1} \times \cdots \times D_{x_k} \mid c(v, v_1, \ldots, v_k)$$

$$\exists v' \in (b^{-w}, b],\ \exists v'_1, \ldots, v'_k \in D_{x_1} \times \cdots \times D_{x_k} \mid c(v', v'_1, \ldots, v'_k)$$

A CSP is 2B(w)–Consistent iff all its domains are 2B(w)–consistent

Problems with 2B(w)-Consistency

- 2B(w)-Consistency filtering **depends on the evaluation order** of projection functions (no fixed point)
- There is no direct relationship between the value of w and the accuracy of filtering
- Decomposition of a constraint system influences the order of evaluation of the projection functions , and can therefore affect the result of filtering by 2B(w)-Consistency
- If handled values are very heterogeneous, it is essential to use a **relative** value for w

Example of 2B(w)-Consistency filtering (1)

Let be:

$c_1 : x = y, \quad c_2 : x = z, \quad c_3 : x^2 = 4$

$D_x = [-10, 2], D_y = [-2, 2], D_z = [-1.5, 2]$ and $w = 1$

- **Order 1 :** $< c_1, c_2, c_3 >$
  $Q = \{< c_1, x >, < c_1, y >, < c_2, x >, < c_2, z >, < c_3, x >$

1. Selection of $< c_1, x >$, $D_x \leftarrow [-\mathbf{2}, \mathbf{2}]$
2. Selection of $< c_1, y >$, D_y is not changed
3. Selection of $< c_2, x >$, D_x is not changed because the reduction is lower than w
4. Selection of $< c_2, z >, < c_3, x >$ cannot achieve any reductions

Example of 2B(w)-Consistency filtering (2)

$c_1 : x = y, \quad c_2 : x = z, \quad c_3 : x^2 = 4$
$D_x = [-10, 2], D_y = [-2, 2], D_z = [-1.5, 2]$ and $w = 1$

- **Order 2:** $< c_2, c_1, c_3 >$
  $Q = \{< c_2, x >, < c_2, z >, < c_1, x >, < c_1, y >, < c_3, x >$

  1. Selection of $< c_2, x >$, $D_x \leftarrow [-1.5, 2]$
     No addition in Q
  2. Selection of $< c_2, z >, < c_1, x >, < c_1, y >$ cannot achieve any reductions
  3. Selection of $< c_3, x >$, $D_x \leftarrow [\mathbf{2}, \mathbf{2}]$
     $< c_1, y >$ and $< c_2, z >$ are pushed in Q
  4. Selection of $< c_1, y >$, $D_y \leftarrow [\mathbf{2}, \mathbf{2}]$
  5. Selection of $< c_2, z >$, $D_z \leftarrow [\mathbf{2}, \mathbf{2}]$

Remark on 2B(w)-Consistency filtering:

$$
\begin{array}{rcl}
\Phi_{2B(w_1)}(P) & \rightarrow & D_x = [a_1, b_1] \\
\Phi_{2B(w_2)}(P) & \rightarrow & D_x = [a_2, b_2] \\
w_1 < w_2 & \not\rightarrow & [a_1, b_1] \subset [a_2, b_2]
\end{array}
$$

A large w can prevent a projection function of a constraint c_j to reduce the domain of some variable x and ... thus allowing a reduction more significant later!

Example:

$f_1 : y \leftarrow 0.71 \times x \quad f_2 : y \leftarrow 0.6 \times x + z$

$D_x = [0, 10] \quad Dy = [-10, 20] \quad D_Z = [0, 0.9]$

- If $w = 2$, f_1 can shrink D_y to $[0, 7.1]$ and thus f_2 cannot achieve any reduction
- IF $w = 3$, f_1 cannot achieve any reduction but f_2 can shrink D_y to $[0, 6.9]$

Box–consistency (1)

- Box–consistency is a **coarser approximation** of arc consistency than 2B-Consistency
- Box–consistency generates a system of uni-variate functions that can be solved with **Newton's method**
- Box–consistency does not amplify the locality problem but may generate a **huge number of constraints**

Box–consistency (2)

Variable x is Box–Consistent for constraint $f(x, x_1, \ldots, x_n) = 0$ if the bounds of the domain of x correspond to the **leftmost** and the **rightmost** **zero** of the **optimal interval extension** $F(X, X_1, \ldots, X_n)$ of $f(x, x_1, \ldots, x_n)$

Definition: Box–consistency

Let $(\mathcal{X}, \mathcal{D}, \mathcal{C})$be a CSP and $C \in \mathcal{C}$ a k-ary constraint over $(X_1, \ldots, X_k)$

C is Box–Consistent if, for all X_i the following relations hold :

1. $C(X_1, \ldots, X_{i-1}, [\underline{X_i}, \underline{X_i}^+), X_{i+1}, \ldots, X_k)$
2. $C(X_1, \ldots, X_{i-1}, (\overline{X_i}^-, \overline{X_i}], X_{i+1}, \ldots, X_k)$

Box–consistency filtering (1)

Filtering by Box–consistency is defined in the following way:

- Transformation of each constraint $\mathbf{C_j(x_{j_1}, \ldots x_{j_k})}$ into $\mathbf{k}$ mono-variable constraints $\mathbf{C_{j,l}}$ by substituting all variables but one by their intervals
- The two extremal zeros of $\mathbf{C_{j,l}}$ can be found by a **dichotomy algorithm** combined with a mono-variable version of the **interval Newton method**

Relations between numeric CSP

- $\mathcal{D}' \subseteq \mathcal{D}$ means $D'_{x_i} \subseteq D_{x_i}$ for all $i \in 1..n$
- CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is **smaller** than $P' = (\mathcal{X}, \mathcal{D}', \mathcal{C})$ if $\mathcal{D} \subseteq \mathcal{D}'$, we note $P \prec P'$
- $P_\emptyset$ denotes the class of empty CSP (CSP with at least one empty domain)
  By convention $P_\emptyset$ is the smallest CSP.

Relation between Box–consistency and 2B-consistency (1)

General case: $\Phi_{2B}(P) \preceq \Phi_{Box}(P)$

Particular case: $\Phi_{2B}(P) \equiv \Phi_{Box}(P)$

if **no variable has multiple occurrences** in any constraint

Relation between Box–consistency and 2B–consistency (2)

A 2B-Consistent CSP is Box–consistency but a Box–consistent CSP may not be 2B-Consistent

Example:

$c\ : x_1 + x_2 - x_1 = 0, D_{x_1} = [-1, 1], D_{x_2} = [0, 1]$
is not be 2B-Consistent for x_2 but is Box–consistency for x_2 because
$([-1, 1] \oplus [0, 0^+] \ominus [-1, 1]) \cap [0, 0]$
and $([-1, 1] \oplus [1^-, 1] \ominus [-1, 1]) \cap [0, 0]$ are not empty

Box versus 2B(decomp)

2B-consistency on the decomposed system is weaker than Box–consistency on the initial system

$$\Phi_{Box}(P) \preceq \Phi_{2B}(\mathbf{P_{decomp}})$$

Proof:
For local consistencies CSP P_{decomp} is a relaxation of P
$\rightarrow$ 2B–consistency (P) $\preceq$ 2B–consistency (P_{decomp}).
Since there aren't any multiple occurrences of variables in P_{decomp}, $\Phi_{Box}(P_{decomp}) \equiv \Phi_{2B}(P_{decomp})$
and thus $\Phi_{Box}(P) \preceq \Phi_{2B}(P_{decomp})$

Example

Let be $c : x_1 + x_2 - x_1 - x_1 = 0$ and $D_{x_1} = [-1, 1]$, $D_{x_2} = [0.5, 1]$

c is not Box–consistency because $[-1, -1^+] \oplus [0.5, 1] \ominus [-1, -1^+] \ominus [-1, -1^+] \cap [0, 0] \neq \emptyset$

$decomp(c)$ is 2B-Consistent for D_{x_1} and D_{x_2}

2B-consistency, 3B-consistency, k-consistencies

- 3B-consistencies achieves a total consistency if the constraint has only two variables
- kB-consistence provides a decision procedure for a constraint system of $k-1$ variables
- Relation between $\Phi_{2B}(P)$ and $\Phi_{3B}(P_{decomp})$:
  - Relation $\Phi_{2B}(P) \preceq \Phi_{3B}(P_{decomp})$ does not hold
  - Relation $\Phi_{3B}(P_{decomp}) \preceq \Phi_{2B}(P)(P)$ does not hold

Implementation issues

- **2B–consistency, Box–consistency, 3B–consistency**
- **HC4-Revise** computes the optimal box (under continuity assumptions) when the constraint contains **no multiple occurrences** of some variable
- **Box-Revise** computes the optimal box (under continuity assumptions) when the constraint contains **one** variable appearing several times
- **Mohc-Revise** better handles the dependency problem, when **several variables occur several times**

Courtesy to Gilles Trombettoni

Standard narrowing algorithm (schema) (1)

IN–1 (**in** $\mathcal{C}$, **inout** $\mathcal{D}$)
$\quad Q \leftarrow \{< x_i, C_j > \mid C_j \in \mathcal{C}$ and $x_i \in Var(C_j)\}$
$\quad$ **while** $Q \neq \emptyset$
$\qquad$ extract $< x_i, C_j >$ from Q
$\qquad \mathcal{D}' \leftarrow \texttt{narrowing}(\mathcal{D}, x_i, C_j)$
$\qquad$ **if** $\mathcal{D}' \neq \mathcal{D}$ **then**
$\qquad\quad \mathcal{D} \leftarrow \mathcal{D}'$
$\qquad\quad Q \leftarrow Q \cup \{< x_l, C_k > \mid (x_l, x_i) \in Var(C_k)\}$
$\qquad$ **endif**
$\quad$ **endwhile**

Standard narrowing algorithm (schema) (1)

$\longrightarrow$ Computation of extremum functions in function `narrowing` of algorithm `IN-1`

1 **function narrow** $(\mathcal{D}, x_i, C_j)$: set of domains
2 $\quad m \leftarrow Min_{x_i}(C, D_{x_i})$
3 $\quad M \leftarrow Max_{x_i}(C, D_{x_i})$
4 $\quad$ **return** $\mathcal{D}[D_{x_i} \leftarrow [m, M]]$

2B-consistency filtering

Algorithm schema

- Generate **projection functions** for each variable of each constraint
- Use **interval extension** of the projection functions to compute $Min_{x_i}(C, D_{x_i})$ and $Max_{x_i}(C, D_{x_i})$

Computing Box–consistency filtering

La function $narrow(c, \mathcal{D})$ (generic algorithm `IN`) reduces the variable domains of c until c is Box–consistency :

- For each variable x of constraint , a **uni-variate interval function** is generated by replacing all variables but x by their domains
- Searching the leftmost zero and the rightmost zero of these uni-variate functions on intervals that are of the form:

$$C(D_{x_1}, .., D_{x_{i-1}}, x, D_{x_{i+1}}, ..., D_{x_k}) = \tilde{0}.$$

$\rightarrow$ Computation of extremum functions in function `narrowing` of algorithm `IN-1`

Function LNAR (1)

Function LNAR computes the leftmost zero F_x for variable x with an initial domain I_x

- LNAR first reduces the interval I_x with function $NEWTON(F_x, I_x)$ (interval extension of Newton's method)
- If $NEWTON(F_x, I_x)$ cannot shrink enough I_x to make it Box–consistency , the domain is divided to check whether the left bound of I_x actually contains a zero
- Function $SPLIT$ splits interval I in two intervals I_1 and I_2; I_1 being the left part of the initial interval

Function LNAR (2)

function LNAR (IN: F_x, I_x**, return** *Interval*)
    r $\leftarrow$ right(I_x)
    if $0 \notin F_x(I_x)$ then **return** $\emptyset$
        else $I_x \leftarrow NEWTON(F_x, I_x)$
            if $0 \in F_x([left(I_x), left(I_x)^+])$ then **return** $[left(I_x), r]$
                else $SPLIT(I_x, I_1, I_2)$
                    $L_1 \leftarrow LNAR(F_x, I_1)$
                    if $L_1 \neq \emptyset$ then **return** $[left(L_1), r]$
                        else **return** $[left(LNAR(F_x, I_2)), r]$
                    endif
            endif
    endif

Figure: Function LNAR

Algorithm HC4

Goal

Limit the loss of information due to the decomposition of the constraints required by 2B–consistency filtering

Principle of algorithm HC4

- **HC4** works on a CSP where each constraint is represented by its **syntax tree**
  (no explicit decomposition: the nodes of the tree are primitive constraints)
- **HC4:** standard propagation scheme
- A projection Π_x^c is implemented by the function **HC4Revise**. At each node of the tree, **HC4Revise** calls a primitive projection ("wired" procedure)

BC4: similar to HC4, adapted for **Box-consistency filtering**

Algorithm HC4-Revise

Algorithm **HC4-Revise** shrinks the current box with a constraint c.

Principle of **HC4-Revise** : shrinks each **occurrence** of a variable by isolating it in c

Implementation of HC4-Revise

- Double exploration of the syntax tree of c.
- Synthesis : **evaluation** (over intervals) at each node of the tree
- Heritage : **elementary projection** at each node of the tree

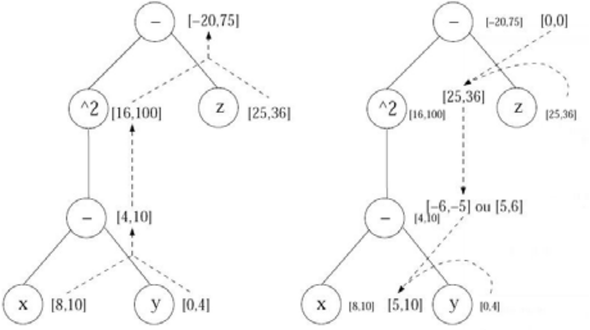
Algorithm HC4Revise

Principle of algorithm HC4Revise *on one constraint c*

- Double exploration of the syntax tree of c.
- Synthesis : **evaluation** (over intervals) at each node of the tree
- Heritage : **elementary projection** at each node of the tree

Example

Evaluation of $(x - y)^2 = z$ with $X = [8, 10], Y = [0, 4], Z = [25, 36]$ **Projection** over x



What computes HC4Revise ?

For any constraint c without multiple occurrences of variables, **HC4Revise** computes **Hull-consistency**, the optimal projection of c

The double-path exploration is enough if it handles unions of intervals in the tree:

- Without multiple occurrences of variables, c is a tree
- If **gaps** are collected ($\longrightarrow$ domain is an union of intervals), projection functions compute **arc-consistency**

Mohc: motivations

Hull-consistency: **smallest box** containing all the solutions to **one** constraint
$\longrightarrow$ Difficult to compute because of the **dependency problem**

- **HC4-Revise** $\longrightarrow$ optimal box (under continuity assumptions) when the constraint contains **no** multiple occurrence of variables.
- **Box-Revise** $\longrightarrow$ optimal box (under continuity assumptions) when at most **one** variable occurs several times
- **Mohc-Revise** better handles the dependency problem, when **several** variables occur several times $\longrightarrow$ exploits the **monotonicity** of functions

Mohc: exploiting monotonicity of functions

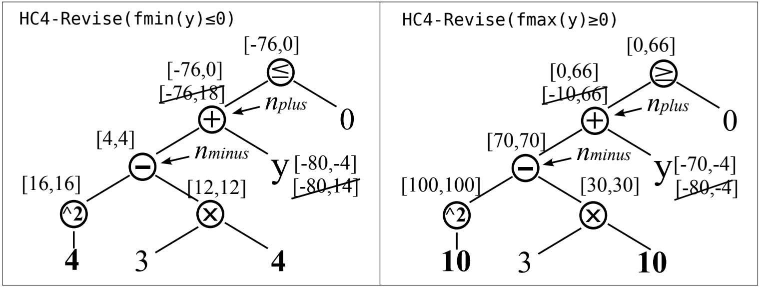
(Simplified) definition of monotonicity-based extension

- Consider $f(x_i, x_d, w)$ such that: f is increasing w.r.t. x_i, and f is decreasing w.r.t. x_d in $X_i \times X_d \times W$
- $W = [\underline{F_{min}(W)}, \overline{F_{max}(W)}]$, with:
  - $f_{min}(w) = f(\underline{X_i}, \overline{X_d}, w)$
  - $f_{max}(w) = f(\overline{X_i}, \underline{X_d}, w)$

$\rightarrow$ If f is monotonic w.r.t. to x in a given box, then the dependency problem related to x **disappears**

Mohc, example

Initial box: $X = [4, 10]$, $Y = [-80, 14]$



QUAD: A global constraint based on safe linear relaxation

- A **tight linear relaxation** of the quadratic constraints adapted from a classical **RLT techniques** (Sherali-Tuncbilek 92,Sherali-Adams 99)
- Use of **LP algorithm** to narrow the domain of each variable
  $\longrightarrow$ the coefficient of these linear constraints are updated

Courtesy to Yahia Lebbah, Claude Michel

QUAD: Motivations (1)

- **Quadratic equations and inequations are widely** used to model distance relations in numerous applications (kinematics, robotics,chemistry)
- Classical (local) filtering algorithms are unable to achieve a significant pruning because these constraints are **handled independently**

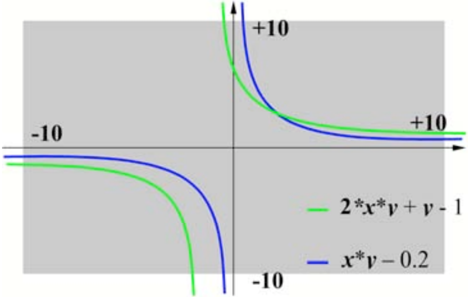
QUAD: Motivations (2)

- **A constraint is handled as a black-box** by local consistencies (2B–filtering,BOX–filtering)
  - No way to catch the dependencies between constraints
  - Splitting is behind the success for small dimensions
- **Higher consistencies** (KB–filtering,Bound–filtering) $\rightarrow$ **visiting numerous combinations**

QUAD: Motivations (3)

- ***2x*y + y = 1***
- ***x*y = 0.2***

  x, y *in [-10, +10]*



- ***Box-consistency : no reduction***

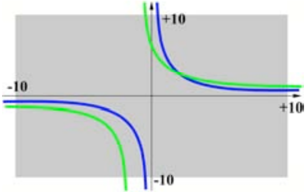
  $0 \in [-211, 209]$ $\quad$ $0 \in [-211, 189]$

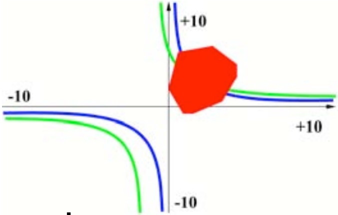
  $0 \in [-100.2, 99.8]$ $\quad$ $0 \in [-100.2, 99.8]$

- ***2B-consistency : no reduction***

  (division by zero when computing the projection functions)

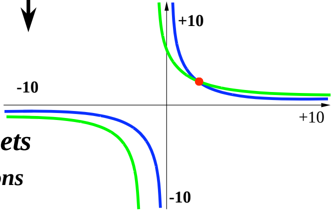
Quad Filtering





$2*x*y + y - 1$

$x*y - 0.2$



- *Quad : based on more precise* ***convex-sets***
  - *Generation of 3* ***linear systems of inequations***
  - **12** *Calls to the* ***simplex***
  - *Provides* ***exact solutions***

QUAD: Safe use of Linear Relaxation

- **A global constraint** to handle a tight approximation of the constraint system with an LP solver

- ***Combines***

  - **safe and rigorous** linear relaxations

  - **local consistencies** and **interval methods**

The QUAD framework

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

Linearisation of x^2

- $\mathbf{f(x) = x^2}$ with $\underline{\mathbf{x}} \leq \mathbf{x} \leq \overline{\mathbf{x}}$ is approximated by :

$$
\begin{aligned}
L_1(y,\alpha) &\equiv [(x-\alpha)^2 \geq 0]_l \text{ where } \alpha \in [\underline{x}, \overline{x}] \\
L_2(y) &\equiv (\underline{x} + \overline{x})x - y - \underline{x} * \overline{x} \geq 0
\end{aligned}
$$

- $[(\mathbf{x} - \alpha_\mathbf{i})^\mathbf{2} = \mathbf{0}]_\mathbf{l}$ generates the tangents to $\mathbf{y} = \mathbf{x^2}$ at $\mathbf{x} = \alpha_\mathbf{i}$
- $L_1(y, \overline{x})$ and $L_1(y, \underline{x})$: underestimations of $\mathbf{y}$
  $L_2(y)$: overestimation of $\mathbf{y}$

QUAD **only computes** $\mathbf{L_1(y, \overline{x})}$ and $\mathbf{L_1(y, \underline{x})}$

Linearisation of x^2

Example 1: relaxation of $\mathbf{x^2}$ with $\mathbf{x} \in [-\mathbf{4}, \mathbf{5}]$

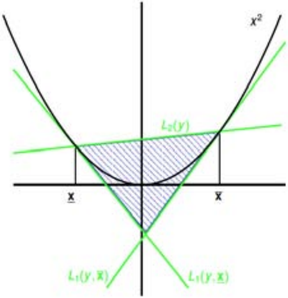
- $L_1(y, \alpha) \equiv y \geq 2\alpha x - \alpha^2$

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

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

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

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



Linearisation of xy

Relaxation of **xy**

$L_3(z) \equiv [(x - \underline{x_i})(y - \underline{x_j}) \geq 0]_l$

$L_4(z) \equiv [(x - \underline{x_i})(\overline{x_j} - y) \geq 0]_l$

$L_5(z) \equiv [(\overline{x_i} - x)(y - \underline{x_j}) \geq 0]_l$

$L_6(z) \equiv [(\overline{x_i} - x)(\overline{x_j} - y) \geq 0]_l$

Example 2:

$\mathbf{z} = \mathbf{xy}$ with $\mathbf{x} \in [-\mathbf{5}, +\mathbf{5}], \mathbf{y} \in [-\mathbf{5}, +\mathbf{5}]$

$L3(z) : z + 5x + 5y + 25 \geq 0$

$L4(z) : -z + 5x - 5y + 25 \geq 0$

$L5(z) : -z - 5x + 5y + 25 \geq 0$

$L6(z) : z - 5x - 5y + 25 \geq 0$

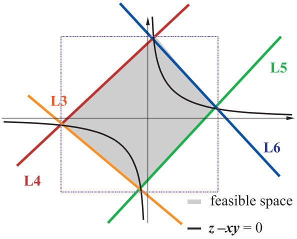
Let's take $z = 5$

$L3(z) : y \geq -x - 6$

$L4(z) : y \leq 4 - x$

$L5(z) : y \geq x - 4$

$L6(z) : y \leq 6 - x$



QUAD filtering algorithm (1)

Function QUAD_filtering(IN: $\mathcal{X}$, $\mathcal{D}$, $\mathcal{C}$, ϵ) **return** $\mathcal{D}'$

1. **Reformulation**
   $\rightarrow$ linear inequalities $[\mathcal{C}]_R$ for the nonlinear terms in $\mathcal{C}$
2. **Linearisation/relaxation of the whole system** $[\mathcal{C}]_L$
   $\rightarrow$ a linear system $LR = [\mathcal{C}]_L \cup [\mathcal{C}]_R$
3. $\mathcal{D}'$:= $\mathcal{D}$
4. **Pruning**

QUAD filtering algorithm (2)

- **Pruning**

While reduction of some bound $> \epsilon$ and $\emptyset \notin \mathcal{D}'$ **Do**

1. **Update the coefficients** of $[\mathcal{C}]_R$ according to $\mathcal{D}'$
2. **Reduce the lower and upper bounds** $\underline{x}'_i$ and $\overline{x}'_i$ of each **initial** variable $x_i \in \mathcal{X}$ by computing min and max of x_i subject to LR with a LP solver
3. **Propagate reductions** with local consistencies, newton

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
$\Rightarrow$ **floating point numbers**

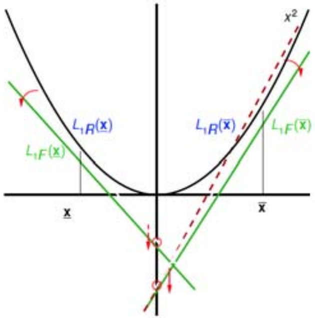
► **All the computations with floating point numbers require right corrections**

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



Safe approximations of L_1

[$L_{1F}(y, \alpha)$ approximations]

Let $\alpha \in F$ and

$$\mathbf{L_{1F}}(\mathbf{y}, \alpha) \equiv \begin{cases} y - \lfloor 2\alpha \rfloor x + \lceil \alpha^2 \rceil \geq 0 & \text{iff } \alpha \geq 0 \\ y - \lceil 2\alpha \rceil x + \lceil \alpha^2 \rceil \geq 0 & \text{iff } \alpha < 0 \end{cases}$$

$\forall x \in \mathbf{x}$, and $y \in [0, max\{\underline{\mathbf{x}}^2, \overline{\mathbf{x}}^2\}]$,

if $L_1(y, \alpha)$ holds, then $L_{1F}(y, \alpha)$ holds too

Generalisation to n-ary linearisations

Let $\sum_{i=1}^{n} a_i x_i + b \geq 0$

then $\forall x_i \in \mathbf{x}_i$:

$$\sum_{i=1}^{n} \overline{a}_i x_i + \sup(\overline{b} + \sum_{i=1}^{n} \sup(\sup(\mathbf{a}_i \underline{x}_i) - \overline{a}_i \underline{x}_i)) \geq \sum_{i=1}^{n} a_i x_i + b \geq 0$$

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 $x_R \in R^n$
- CPLEX 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 and Shcherbina
  - → ***cheap method to obtain a rigorous bound of the objective***
  - → ***rigorous computation of the certificate of infeasibility***

Quadrification: Power terms

A **power term** of the form x^n can be approximated by $n+1$ **inequalities** with a procedure proposed by Sherali and Tuncbilek , called “bound-factor product RLT constraints”
It is defined by the following formula:

$$[x^n]_R \;=\; \{[(x-\underline{x})^i(\overline{x}-x)^{n-i} \geq 0]_L, i = 0...n\} \tag{1}$$

Quadrification: product term (1)

For the **product term**

$$x_1 x_2 \ldots x_n \tag{2}$$

The **Quadrification** step brings back the multi-linear term into a **set of quadratic terms** as follows:

$$\begin{array}{cccc}
\hline
\underbrace{x_1 x_2 \ldots x_n} & = & \underbrace{x_1 \ldots x_{d1}} & \underbrace{x_{d1+1} \ldots x_n} \\
x_{1\ldots n} & = & x_{1\ldots d1} \times & x_{d1+1\ldots n} \\
\hline
 & & \underbrace{x_1 \ldots x_{d2}} & \underbrace{x_{d2+1} \ldots x_{d1}} \\
x_{1\ldots d1} & = & x_{1\ldots d2} \times & x_{d2+1\ldots d1} \\
\hline
 & & \underbrace{x_{d1+1} \ldots x_{d3}} & \underbrace{x_{d3+1} \ldots x_n} \\
x_{d1+1\ldots n} & = & x_{d1+1\ldots d3} \times & x_{d3+1\ldots n} \\
\hline
\ldots & & & \\
\hline
\end{array}$$

where $x_{i\ldots j} = [x_i x_{i+1} \ldots x_j]_L$.

Quadrification: product term (2)

For instance, consider the term $x_1 x_2 x_3 x_4 x_5$. The proposed quadrification process would operate in the following way:

$$\underbrace{x_1 x_2 x_3 x_4 x_5}_{y_1} = \underbrace{x_1 x_2 x_3}_{y_2} \times \underbrace{x_4 x_5}_{y_3}$$

$$y_2 = \underbrace{x_1 x_2}_{y_4} \times \underbrace{x_3}_{x_3}$$

$$y_3 = \underbrace{x_4}_{x_4} \times \underbrace{x_5}_{x_5}$$

$$y_4 = \underbrace{x_1}_{x_1} \times \underbrace{x_2}_{x_2}$$

Search

- Main **heuristics**
- Mind the **Gaps**

Main heuristics

In the search tree, **the choice of the next variable to bisect is very important**

Three heuristics are commonly used:

- **Round robin**
- **Select first the largest interval**
- **Smear** function (Kearfott 1990)
  - For each (f, x), in the current box $[B]$: compute $smear(f, x) = |\frac{\partial f}{\partial x}([B])| \times Diam([x])$;
  - For some variable x: $smear(x) = \sum_j (smear(f_j, x))$ (or $Max_j(smear(f_j, x)))$;
  - Bisect the variable with the strongest impact.

Standard splitting vs Mind The Gaps

- **Collect** gaps while filtering (HC4 Revise)
- **Eliminate** non relevant gaps
- **Select** relevant gaps
- Generate **sub problems**