Tutorial: CP by Systematic Search Over Real-Number and Floating-Point Domains

Michel RUEHER

University Nice Sophia-Antipolis
I3S – CNRS, France

CP meets CAV

25 June – 29 June 2012

Turunç, Turkey

Outline

Numeric CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$:

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of variables
- $\mathcal{D} = \{D_{x_1}, \ldots, D_{x_n}\}$ is a set of domains
  (D_{x_i} contains all acceptable values for variable x_i)
- $\mathcal{C} = \{c_1, \ldots, c_m\}$ is a set of constraints

Overall scheme

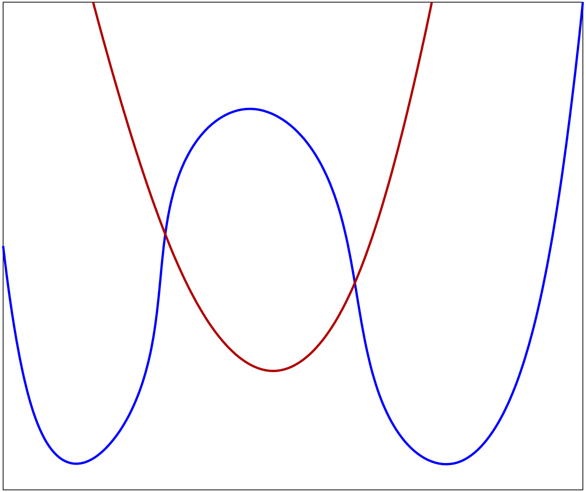
The constraint programming framework is based on a **branch & prune** schema which is best viewed as an iteration of two steps:

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

- The pruning step $\rightarrow$ **reduces an interval** when it can proved that the upper bound or the lower bound does not satisfy some constraint
- The branching step $\rightarrow$ **splits the interval** associated to some variable in two intervals (often with the same width)

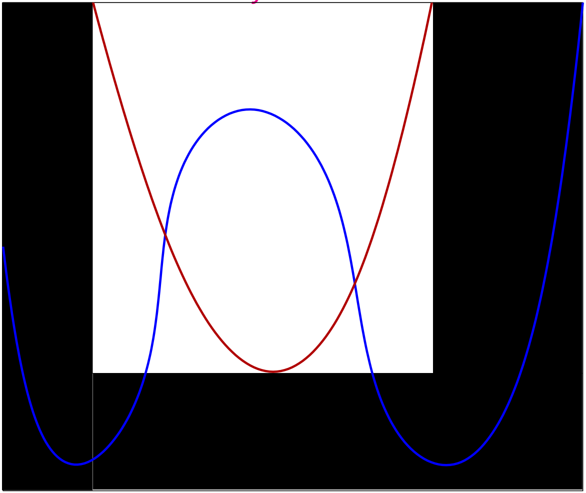
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



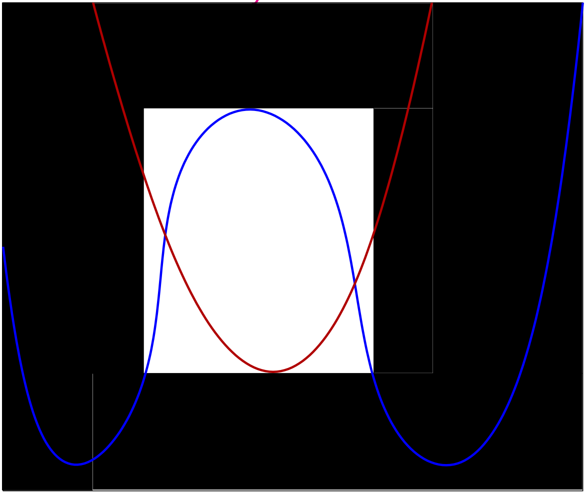
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



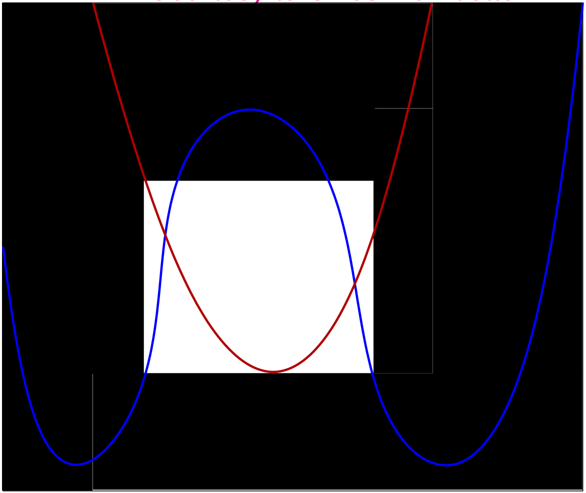
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



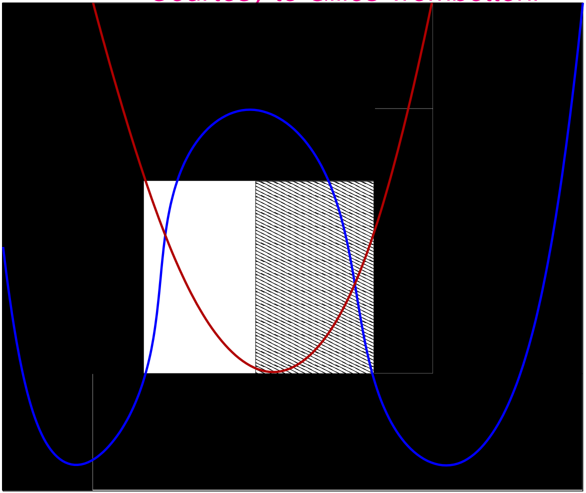
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



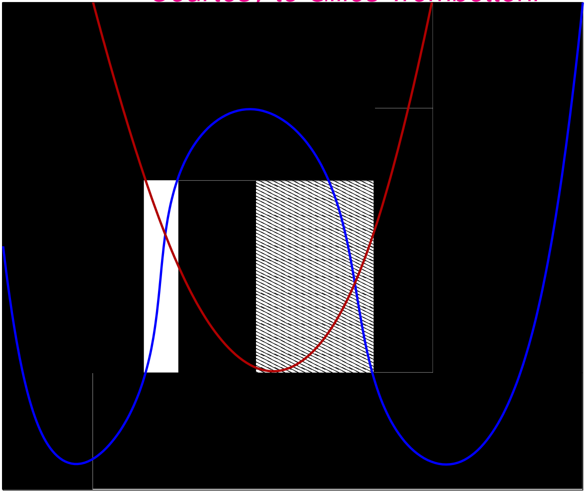
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



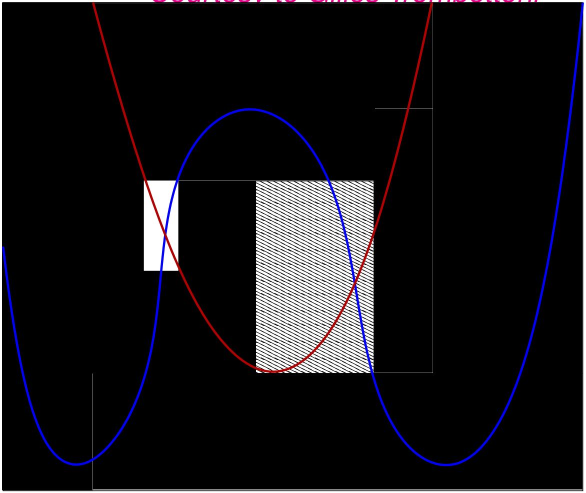
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



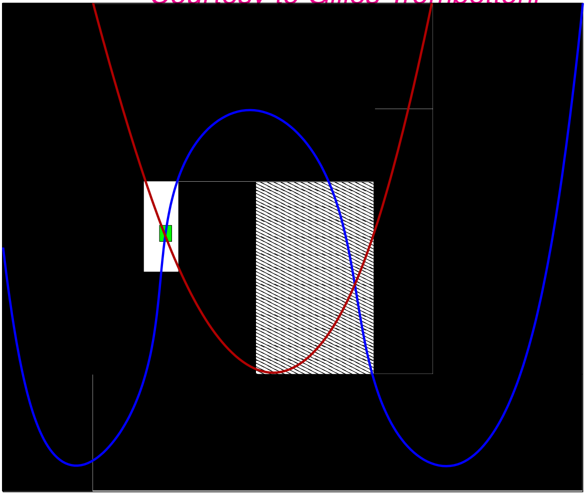
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



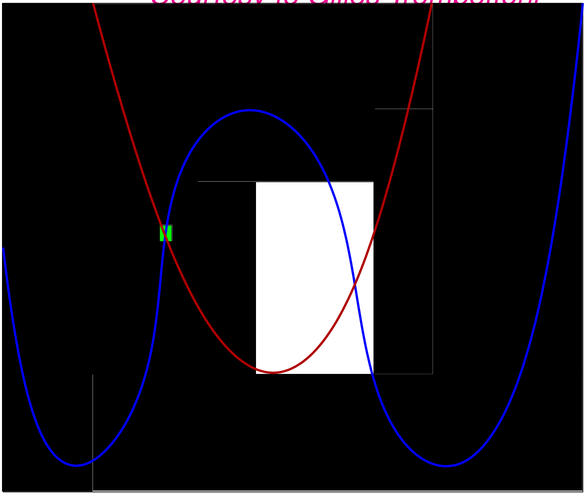
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



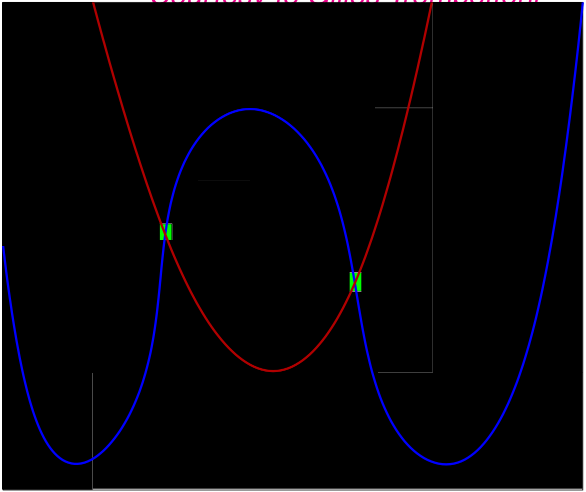
Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



Filtering & Solving process (example)

Courtesy to Gilles Trombettoni



Why do we need intervals?

- **Modelling uncertainty**
  - Error in Measurement or uncertainty in measurements
  - Uncertainty when estimating unknown values
- **Safe Computations** with floating-point numbers
  - Rounding errors
  - Cancellation, ...

What Every Computer Scientist Should Know About Floating-Point Arithmetic, Goldberg, 1991

Problem with floating-point computations

Examples

(in simple precision)

- **Absorption:** $10^7 + 0.5 = 10^7$
- **Cancellation:** $((1 - 10^{-7}) - 1) * 10^7 = -1.192...(\neq -1)$
- Operations are **not associative:** $(10000001 - 10^7) + 0.5 \neq 10000001 - (10^7 + 0.5)$
- **No exact representation:** $0.1 = 0.000110011001100\ldots$

Rump polynomial

- `RumpFunc[x_,y_]:=`$(1335/4 - x^2)y^6 + x^2(11x^2y^2 - 121y^4 - 2) + (11/2)y^8 + x/(2y)$
- Value computed with rational numbers: `RumpFunc[77617, 33096]`$= -\frac{54767}{66192} = -0.827396$
- Value with floating point numbers: **0**
- Value with floating point numbers when $11/2$ is replaced by 5.5 in the polynomial: 1.18059×10^{21}

What is an interval?

An **interval** $[a, b]$ describes a set of real numbers x such that: $a \leq x \leq b$

Assumption:

a and b belong to **finite set** of numbers representable on a computer: **floating-point numbers**, subset of integers, rational numbers, ...

A **Box** denotes a Cartesian product of intervals

→ a box is a vector of intervals that defines the **search space** of problem,
i.e., the space in which are the values of the variables

Interval arithmetic

Interval arithmetic (Moore-1966)
is based on the representation of variables as intervals

Let f be a real-valued function of n unknowns $\{x_1, \ldots, x_n\}$, an **interval evaluation** F of f for given ranges $\mathbf{X} = \{X_1, \ldots, X_n\}$ for the unknowns is an interval Y such that

$$\forall \{v_1, \ldots, v_n\} \in \{X_1, \ldots, X_n\} : \ \underline{Y} \leq f(v_1, \ldots, v_n) \leq \overline{Y}$$

$\underline{Y}, \overline{Y}$: lower and upper bounds for the values of f when the values of the unknowns are restricted to the box $\mathbf{X}$

Interval extension

- 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

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 $\rightarrow [-\infty, +\infty]$

Natural interval extension: Example

Let $f = (x + y) - (y \times x)$ be a real function

Let be $X = [-2, 3]$, $Y = [-9, 1]$

$$
\begin{aligned}
F &= (X \oplus Y) \ominus (Y \otimes X) \\
&= ([-2, 3] \oplus [-9, 1]) \ominus ([-9, 1] \otimes [-2, 3]) \\
&= [-11, 4] \ominus \\
&\qquad [\min(18, -27, -2, 3), \max(18, -27, -2, 3)] \\
&= [-11, 4] \ominus [-27, 18] \\
&= [-29, 31]
\end{aligned}
$$

Interval arithmetic: extension of relations

Let $C : \mathcal{I}^n \rightarrow \mathcal{B}ool$ be a **relation** over the intervals

C is an **interval extension** of the relation $c : \mathcal{R}^n \rightarrow \mathcal{B}ool$ iff:

$$\forall X_1, \ldots, X_n \in \mathcal{I} : \exists v_1 \in X_1 \wedge \ldots \wedge \exists v_n \in X_n \wedge c(v_1, \ldots, v_n) \Rightarrow C(X_1, \ldots, X_n)$$

For instance, $X_1 \doteq X_2 \Leftrightarrow (X_1 \cap X_2) \neq \emptyset$ is an interval extension of the relation $x_1 = x_2$ over the real numbers

Example:

Relation $X_1 \doteq X_2$ holds if $X_1 = [0, 17.5]$ and $X_2 = [17, 27.5]$

Interval extension: properties

- If $\mathbf{0} \notin \mathbf{F}(\mathbf{X})$, then no value exists in the box $\mathbf{X}$ such that $f(X) = 0$ $\rightarrow$
  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**: $Y \subseteq X \Rightarrow F(Y) \subseteq F(X)$
- No distributivity but interval arithmetic is **sub-distributive**: $X(Y + X) \subseteq XY + XZ$

Problems when computing the image of an interval function (1)

- **Outward Rounding** (required for safe computations with floating point numbers)
  $\rightarrow$ ***enlarges intervals***
- **Non continuity** of interval functions: the image of an interval is in general not an interval
  $\rightarrow$ The ***wrapping effect***, which overestimates by a unique vector the image of an interval vector
  Example:
  $$f(x) = \frac{1}{x} \text{ with } X = [-1, 1]$$
  $$F([-1, 1]) = \frac{1}{[-1, 1]} = [-\infty, -1] \cup [1, +\infty]$$
  $$\rightarrow = [-\infty, +\infty]$$

Problems when computing the image of an interval function (2)

- The ***dependency problem***, which is due to the **independence** of the different occurrences of a variable during the interval evaluation of an expression

Examples:

Consider $X = [0, 5]$

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

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

$X(X - 1) = [0, 5]([0, 5] - [1, 1]) = [0, 5][-1, 4] = [-5, 20]$

Interval extension: using different literal forms (1)

- **Factorized form** (Horner for polynomial system) or distributed form
- **First-order Taylor development** of f

$$F_{\text{tay}}(X) = f(x) + J(X).(X - x)$$

with $\forall x \in X$, $J()$ being the Jacobian of f

Interval extension: using different literal forms (2)

- In general, first order Taylor extensions yield a better enclosure than the natural extension on **small intervals**
- Taylor extensions have a **quadratic convergence** whereas the natural extension has a linear convergence
- In general, neither F_{nat} nor F_{tay} won't allow to compute the exact range of a function f

Interval extension: using different literal forms (3)

Consider $f(x) = 1 - x + x^2$, and $X = [0, 2]$

$$f_{tay}([0,2]) = f(x) + (2X - 1)(X - x)$$
$$= f(1) + (2[0,2] - 1)([0,2] - 1) = [-\mathbf{2}, \mathbf{4}]$$

$$f_{nat}([0,2]) = 1 - X + X^2 = [1,1] - [0,2] + [0,2]^2 = [-\mathbf{1}, \mathbf{5}]$$

$$f_{factor}([0,2]) = 1 + X(X - 1) = [1,1] + [0,2]([0,2] - [1,1])$$
$$= [-\mathbf{1}, \mathbf{3}]$$

whereas the range of f over $X = [0, 2]$ is $[0.75, 3]$

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 $X = [x', \overline{x}]$
- A **constraint C_j is AC-like-consistent** if for any variable x_i in $\mathcal{X}_j$, **the bounds $\underline{D_i}$ and $\overline{D_i}$ have a support** in the domains of all other variables of $\mathcal{X}_j$

Local consistencies (2)

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

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

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

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

Local consistencies (3)

- **2B–consistency / Hull–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 but may achieve a **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 $\mathbf{F}(\mathbf{X}, \mathbf{X_1}, \ldots, \mathbf{X_n})$, the optimal interval extension of $f(x, x_1, \ldots, x_n)$

Local consistency filtering (1)

Algorithms that achieve a local consistency filtering are based upon projection functions

- **Solution functions** express the variable x_i in terms of the other variables of the constraint. The solution functions of $x + y = z$ are:
  $f_x = z - y, f_y = z - x, f_z = x + y$
- For complex constraints, **no analytic solution function may exist**
  Example: $x + log(x) = 0$

Local consistency filtering (2)

- **Analytic functions exist** when the variable to express in terms of the others appears **only once** in the constraint
- **Otherwise** $\rightarrow$ **to consider that each occurrence is a different new variable**

  For $x + log(x) = 0$ we obtain
  $\{x_1 + \log(x_2) = 0, x_1 = x_2\}$
  $\rightarrow f_{x_1} = -\log(x_2)$, $f_{x_2} = \exp^{-x_1}$

  - Decomposition does not change the semantics of the initial constraints system
  - ... but **it amplifies the dependency problem**

2B-consistency filtering(1)

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

→ considers that **each occurrence is a different new variable**

→ initial constraints are decomposed into “primitive” constraints

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 !

$\rightarrow$ **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:

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

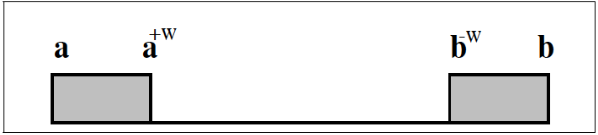
......

... 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** for variable x if:

1. $\exists v \in [a, a^{+w})$ and v is the leftmost zero of $f(x, x_1, \ldots, x_n)$
2. $\exists v' \in (b^{-w}, b]$ and v' is the rightmost zero of $f(x, x_1, \ldots, x_n)$

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

Box–consistency filtering

Transformation of the constraint $C_j(x_{j_1}, \ldots x_{j_k})$ into k **mono-variable constraints** by substituting all variables but one by their intervals

- The two extremal zeros of $C_{j,l}$ can be found by a **dichotomy algorithm** combined with a mono-variable version of the **interval Newton method**
- Box–consistency does not amplify the locality problem but it may generate **a huge number of constraints**

3B–Consistency

3B–Consistency, a relaxation of path consistency

→

checks whether 2B–Consistency can be enforced when the domain of a variable is **reduced to the value of one of its bounds** in the whole system

3B–Consistency (2)

Definition: 3B–Consistency

Let $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP and x a variable of $\mathcal{X}$ with $D_x = [a, b]$.

Let also:

- Let $P_{D_x^1 \leftarrow [a, a^+)}$ be the CSP derived from P by substituting D_x in $\mathcal{D}$ with $D_x^1 = [a, a^+)$
- Let $P_{D_x^2 \leftarrow (b^-, b]}$ be the CSP derived from P by substituting D_x in $\mathcal{D}$ with $D_x^2 = (b^-, b]$

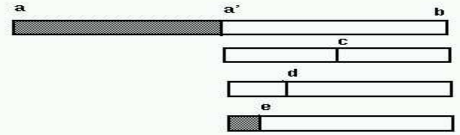
X is 3B–Consistent iff

$$\Phi_{2B}(P_{D_x^1 \leftarrow [a, a^+)}) \neq P_\emptyset \text{ and } \Phi_{2B}(P_{D_x^2 \leftarrow (b^-, b]}) \neq P_\emptyset$$

3B–Consistency (3)

Let $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP and $D_x = [a, b]$, if $\Phi_{2B}(P_{D_x \leftarrow [a, \frac{a+b}{2}]}) = \emptyset$

- then the part $[a, \frac{a+b}{2})$ of D_x will be removed and the filtering process continues on the interval $[\frac{a+b}{2}, b]$
- otherwise, the filtering process continues on the interval $[a, \frac{3a+b}{4}]$.



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 $P \prec P'$

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

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

Implementation issues

- **Standard narrowing algorithm**
- **HC4-Revise** computes the optimal box (under continuity assumptions) when no constraint contains **multiple occurrences** of a variable
- **Box-Revise** computes the optimal box (under continuity assumptions) when each constraint contains at most **one** variable appearing several times

Standard narrowing algorithm (schema) (1)

IN–1 (**in** $\mathcal{C}$, **inout** $\mathcal{D}$)

rangle $Q \leftarrow \{\langle x_i, C_j \rangle | \ C_j \ \in \mathcal{C}$ and $x_i \in Var(C_j)\}$

while $Q \neq \emptyset$

extract $\langle x_i, C_j \rangle$ from Q

$\mathcal{D}' \leftarrow \texttt{narrowing}(\mathcal{D}, x_i, C_j)$

if $\mathcal{D}' \neq \mathcal{D}$ **then**

$\mathcal{D} \leftarrow \mathcal{D}'$

$Q \leftarrow Q \cup \ \{\langle x_l, C_k \rangle | (x_l, x_i) \in Var(C_k)\}$

endif

endwhile

Standard narrowing algorithm (schema) (1)

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

function narrow $(\mathcal{D}, x_i, C_j)$: set of domains
    $m \leftarrow Min_{x_i}(C, D_{x_i})$
    $M \leftarrow Max_{x_i}(C, D_{x_i})$
    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 c, 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:

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

  Use $NEWTON(F_x, I_x)$ (interval extension of Newton's method) to compute extremum functions in function `narrowing`

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 is implemented by the function **HC4Revise** which shrinks the current box with a constraint c

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

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

Global constraints

- Global constraints played a key role in the **success of CP on finite domains**
- QUAD: **a linear approximation**
  - 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
    $\rightarrow$ the coefficient of these linear constraints are updated

Courtesy to Yahia Lebbah, Claude Michel

The QUAD framework

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

Linearisation of x^2

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

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

- $L_1(y,\alpha)$ generates the tangents to $y = x^2$ at $x = \alpha_i$
- $L_1(y,\overline{x})$ and $L_1(y,\underline{x})$: underestimations of y
  $L_2(y)$: overestimation of y

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

Linearisation of x^2

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

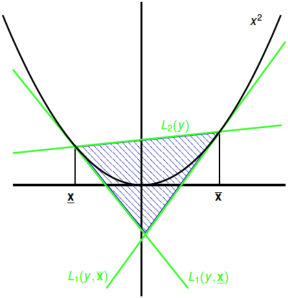
- $L_1(y, \alpha) : y \geq 2\alpha x - \alpha^2$

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

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

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

  $L_2(y) : y \leq x + 20$



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:

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

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

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

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

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

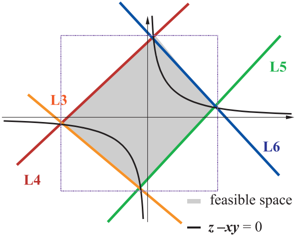
Let's take $z = 5$

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

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

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

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



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**

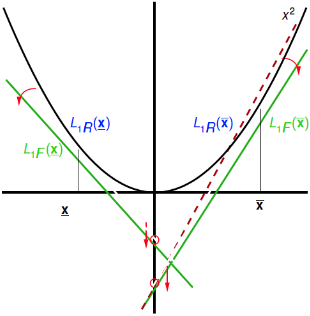
- ▸ **All the computations with floating point numbers require right corrections**

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

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

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

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 :

$$\text{minimise } c^T x$$
$$\text{subject to } \underline{b} \leq Ax \leq \overline{b}$$

- 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
  - $\rightarrow$ ***cheap method to obtain a rigorous bound of the objective***
  - $\rightarrow$ ***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:

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

Quadrification: product term

For the **product term**

$$x_1 x_2 \ldots x_n \qquad (2)$$

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

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

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

Constraints over the floats

$\rightarrow$ **Testing and verifying** floating point computations

- **Problem**: solvers over R ***lose solutions*** over F

Example (double precision, rounding to the nearest):

- over R, $(x + y) + z = x + (y + z)$
  over F, $(x + y) + z \neq x + (y + z)$
- $x < 0 \wedge x + 16.1 = 16.1$
  no solution over R but ... many solutions over F !!
  e.g., $x \in [-1.776356839400250046e^{-15}, 0^{-}]$
- $x * x = 2$
  2 solutions over R, no solution over F

Intervals over F:

$[\underline{x}, \overline{x}]_F$ denotes the ***finite set*** $\{x \in F, \underline{x} \leq x \wedge x \leq \overline{x}\}$

Box over F

- **Observation:** if the ***order of the operations*** is respected, interval computation (outward rounded) provides a ***safe refutation procedure*** over F

- **Procedure:**
  Let $c(x_1, \ldots, x_n)$ be a constraint over F and $x_i' \in [\underline{x_i}, \overline{x_i}]$, if $C(X_1, \ldots, X_{i-1}, [\underline{x}, x_i'], X_i, \ldots, X_n)$ hasn't any solutions, then X_i can be reduce to $[x_i', \overline{x_i}]$

- **Problem:** may be ***slow*** since x_i' has to be computed iteratively (Newton does not apply here)

2B over F

- **Projection functions** of elementary constraints

  $z_F = x_F + y_F$

  direct projection: $\mathbf{z}'_F \leftarrow \mathbf{z}_F \cap (\mathbf{x}_F + \mathbf{y}_F)$

  inverse projections: $\mathbf{x}'_F \leftarrow \mathbf{x}_F \cap (\mathbf{z}_F - \mathbf{y}_F)$

  $\mathbf{y}'_F \leftarrow \mathbf{y}_F \cap (\mathbf{z}_F - \mathbf{x}_F)$
- **Direct projection:** use of interval arithmetic with the known rounding direction (that of the program)
- **Inverse projections:** rounding mode dependant with a rounding mode set to $-\infty$:

$$\mathbf{x}'_F = \mathbf{x}_F \cap [round^{+}_{+\infty}(\underline{z_F}^{-} - \overline{y_F}),\ round_{-\infty}(\overline{z_F} - \underline{y_F})]$$

where

$$round^{+}_{+\infty}(x) = \begin{cases} x^{+} & \text{iff } x \in F, \\ round_{+\infty}(x) & otherwise. \end{cases}$$

2B over F

- **Improvement:**
  Consider $z = x + y$ and $z \in [2^-, 2^-]$
  then x and $y \in [-2^-, 4^-]$

  $\rightarrow$ ***improves filtering speed*** and cuts some slow convergence issues

- **Higher consistencies:** kb-consistencies can be computed by using 2b-consistency

CSP over F

→ **Good approximation** of the “numerical semantics” of arithmetic operations of C programs

→ **Identifying solutions** spaces over the floats that do not contain any solution over the real numbers

Search

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

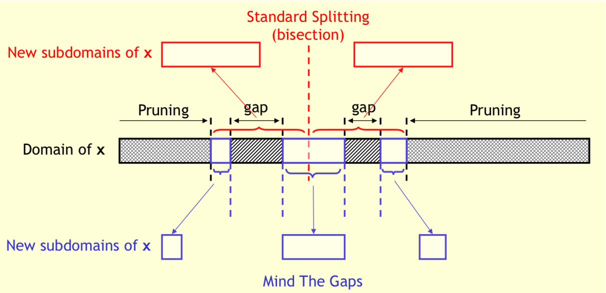
Main heuristics

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**



Conclusion

- **Local consitencies**
  → power-full **refutation capabilities**
- **Main difficulty:**
  → finding a **good trade-off** between pruning and search
- **Applications**
  - Global optimisation: **boosting** safe techniques
  - Program verification:
    → **Refining Approximations**
    → Finding **counterexamples**

Systems

Realpaver:

http://pagesperso.lina.univ-nantes.fr/info/perso/permanents/granvil/realpaver/index.html

Gaol:

http://sourceforge.net/projects/gaol

IBEX

http://www.emn.fr/z-info/ibex/index.html

GlobSol :

http://interval.louisiana.edu/GlobSol/download_GlobSol.html

ICOS :

http://sites.google.com/site/ylebbah/icos

Solvers over F:

- **FPSE** (Mathieu Carlier, INRIA Rennes)
- **COLIBRI** (Bruno Marre, LIST/CEA)
- **FPLib** (Claude Michel, I3S/UNS)