Adaptive Constructive Interval Disjunction

Bertrand Neveu, **Gilles Trombettoni**

Imagine, LIGM, University Paris-Est (France)
LIRMM, University of Montpellier (France)

ICTAI, Washington D.C., November the 4^{th}, 2013

Plan

1. Brief introduction to interval methods
2. 3B-consistency, CID-consistency
3. ACID
4. Experiments

Outline

1. Brief introduction to interval methods
2. 3B-consistency, CID-consistency
3. ACID
4. Experiments

Interval, box, NCSP

Interval, box

- **Interval** $[x_i] = [\underline{x_i}, \overline{x_i}] \equiv \{x_i \in \mathbb{R}, \underline{x_i} \leq x_i \leq \overline{x_i}\}$
  (floating-point bounds)
- A (parallel to axes) **box** $[x]$ is a Cartesian product of intervals: $[x_1] \times ... \times [x_i] \times ... \times [x_n]$
- width or size of an interval: $w([x_i]) := \overline{x_i} - \underline{x_i}$
  width of a box $w([x]) := Max_n(w([x_i]))$
  perimeter of a box: $Peri([x]) := \sum_i w([x_i])$
- Remark: a union of boxes is not a box...
  ... the **Hull** operator approximates the union of a set S of boxes by the smallest box including all points in S.

Numerical CSP (NCSP)

Numerical CSP $P = (x, [x], c)$

- $x = \{x_1, ..., x_i, ..., x_n\}$: real-valued variables
- $[x]$: a box (domain)
- c: a set of (linear or nonlinear) numerical constraints (equations and inequalities)
- solution of P: values of x in $[x]$ that satisfy c

Example of NCSP (AOL-legentil.bch from COPRIN benchmark)

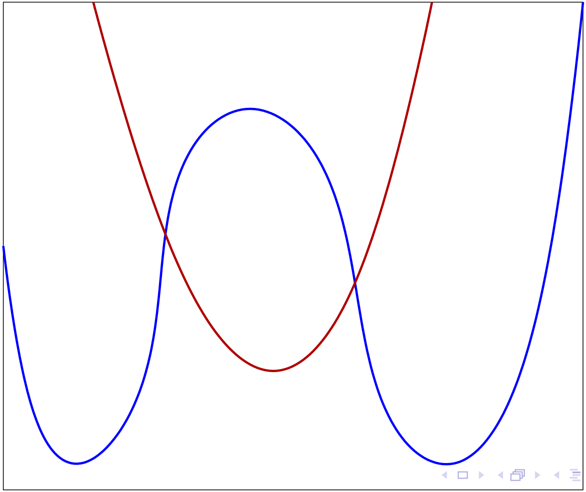
```
Variables

x in [1e-10, pi/2-1e-10];
y in [0, pi/2-1e-10];
z in [-1e8, 1e8];

Constraints

10/3*cos(x)/sin(x)^2+4*(1+tan(x)^2)/cos(y)
  +z*(-50/3*sin(y)*cos(x)/(sin(x)^2*(3.5-5*sin(y)))
  -10/3*cos(x)/sin(x)^2-4*(1+tan(x)^2)/cos(y))=0;

4*tan(x)*sin(y)/cos(y)^2+z*(50/3*cos(y)/(sin(x)
  *(3.5-5*sin(y)))+250/3*sin(y)*cos(y)/(sin(x)
  *(3.5-5*sin(y))^2)-4*tan(x)*sin(y)/cos(y)^2)=0;

50/3*sin(y)/(sin(x)*(3.5-5*sin(y)))+20+10/3/sin(x)
  -4*tan(x)/cos(y)=0;
```

Applications in several fields

- nonconvex constraint systems satisfaction and global optimization
- geolocalization (GPS navigation device)
- robotics (mobile robots, Simultaneous Localization And Mapping, parallel robot design)
- robust parameter estimation (calibration)
- proof of mathematical property
- ...

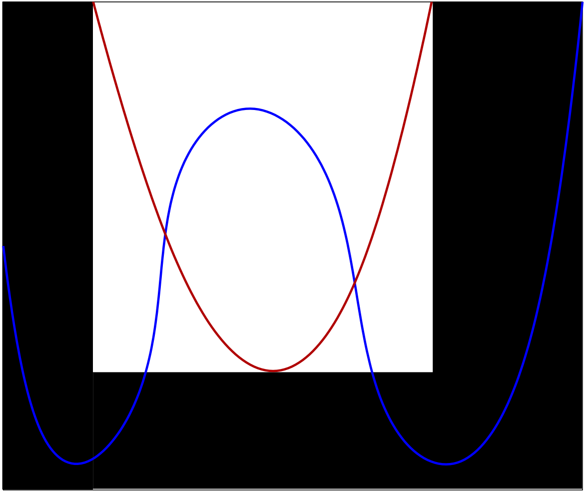
Branch and Contract for solving an NCSP

Solving $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$



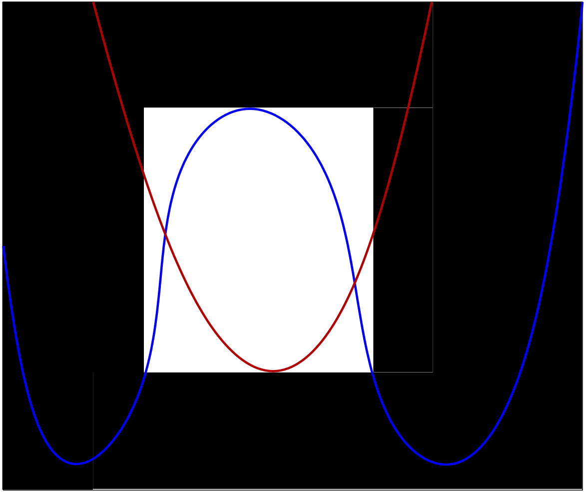
Branch and Contract for solving an NCSP

Solving $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$



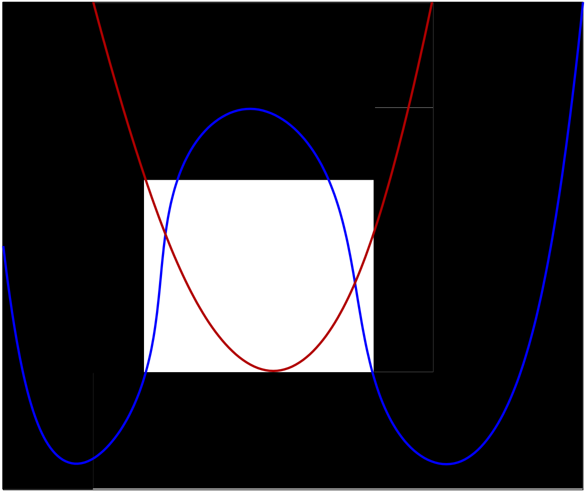
Branch and Contract for solving an NCSP

Solving $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$



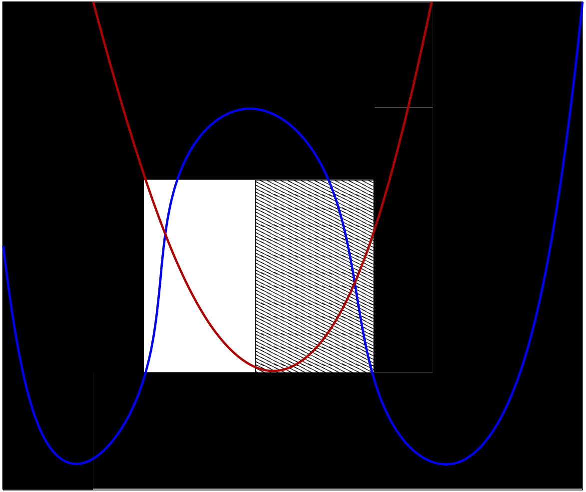
Branch and Contract for solving an NCSP

Solving $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$



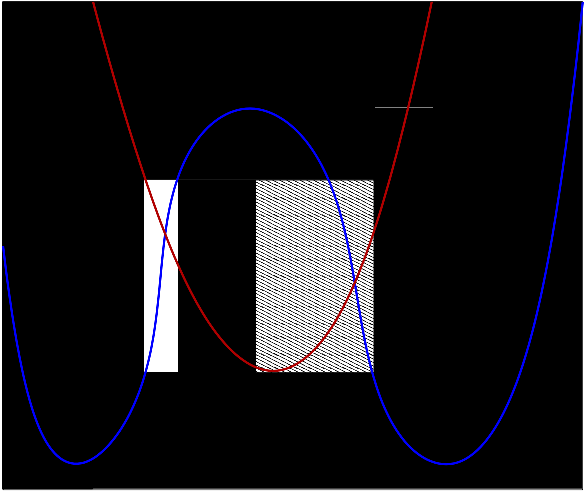
Branch and Contract for solving an NCSP

Solving $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$



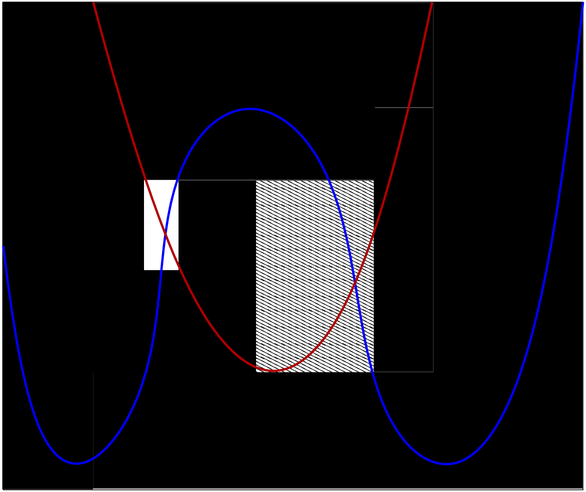
Branch and Contract for solving an NCSP

Solving $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$



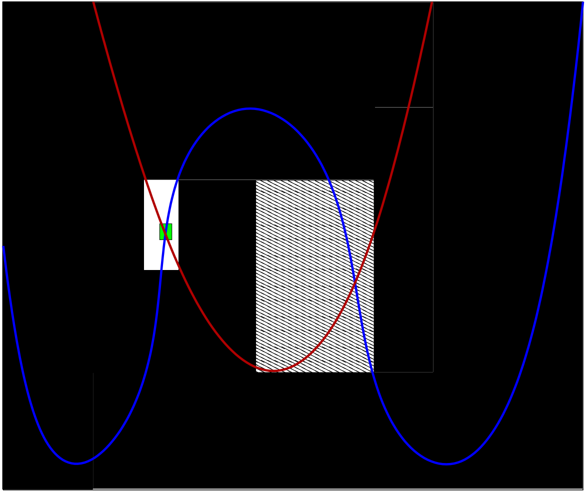
Branch and Contract for solving an NCSP

Solving $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$



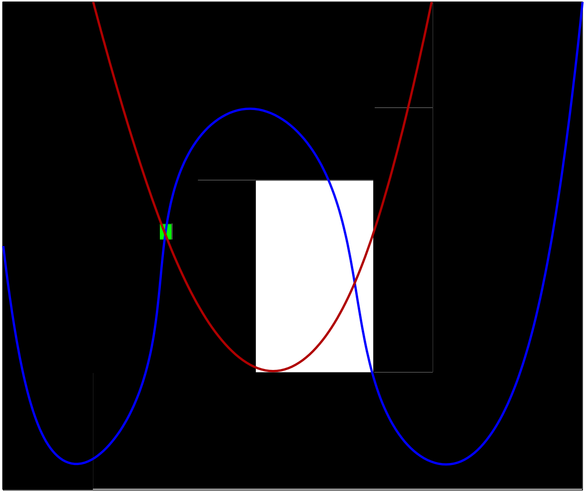
Branch and Contract for solving an NCSP

Solving $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$



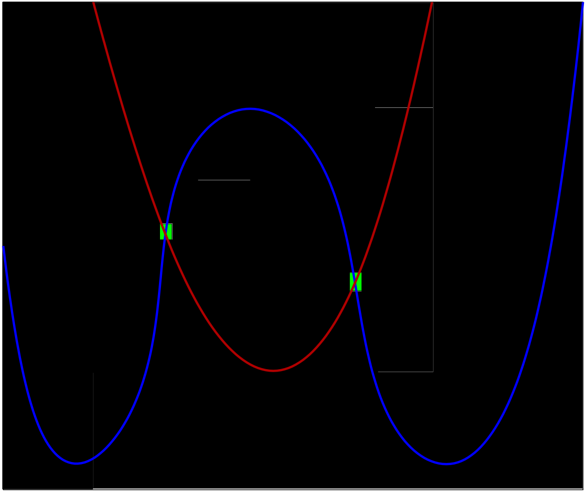
Branch and Contract for solving an NCSP

Solving $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$

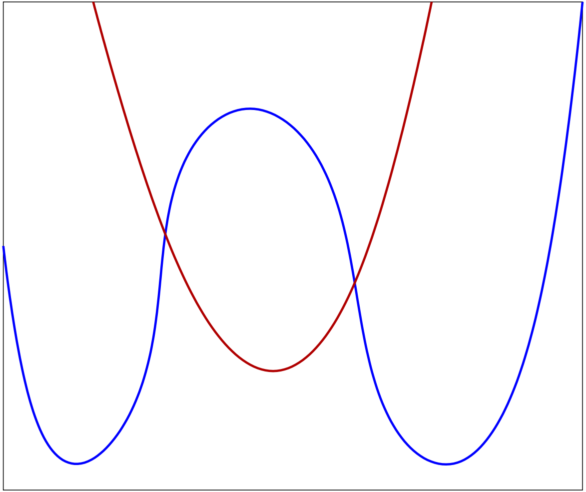


Branch and Contract for solving an NCSP

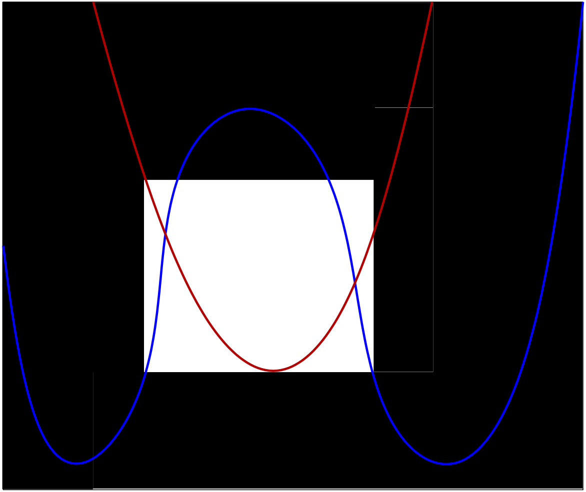
Solving $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$



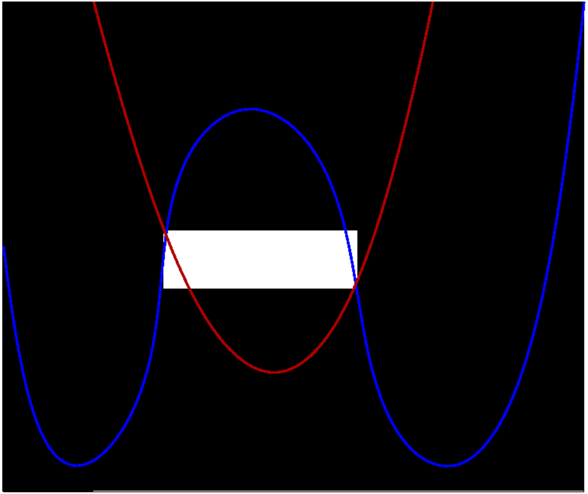
Stronger (CP) consistency



Stronger (CP) consistency



Stronger (CP) consistency



Outline

1 Brief introduction to interval methods

2 3B-consistency, CID-consistency

3 ACID

4 Experiments

SAC and Partition-1-AC consistencies for CSP

SAC [Debruyne+Bessiere, IJCAI 1997]

A value (x_i, v) is SAC-consistent w.r.t. a CSP P iff the subproblem $P_{x_i=v}$ is not arc-inconsistent

$\Rightarrow$ v can be removed from the domain of x_i if $P_{x_i=v}$ is arc-inconsistent (hence has no solution)

Partition-1-AC [Bennaceur+Affane, CP 2001]

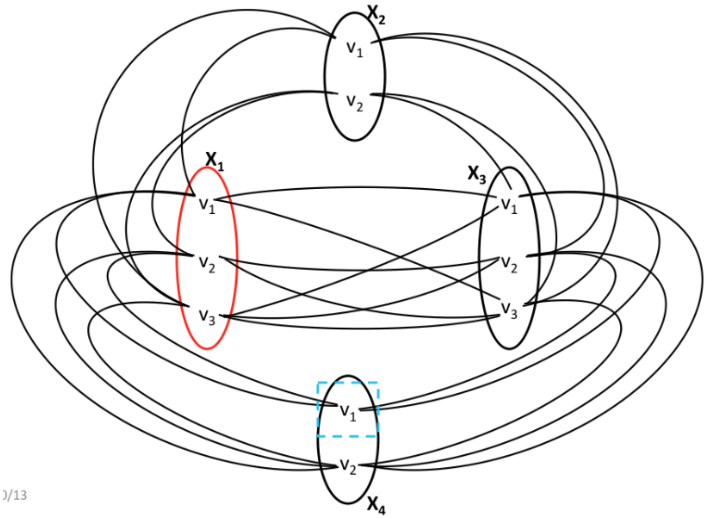
P is P-1-AC iff P is P-1-AC w.r.t. each variable of P

P is P-1-AC w.r.t. a variable x_i of domain $\{v_1, ..., v_k, ...v_n\}$ iff all the values (x_j, v) in P belong to $AC(P_{x_i=v_1}) \cup ... \cup AC(P_{x_i=v_n})$

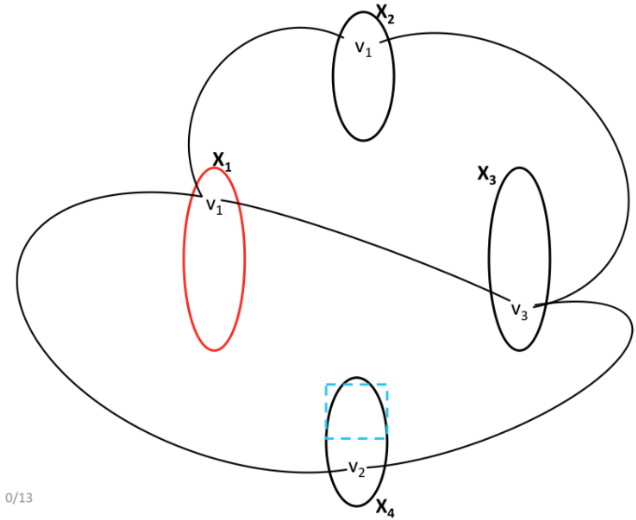
$\Rightarrow$ the procedure VarP-1-AC enforcing $AC(P_{x_i=v_1}) \cup ... \cup AC(P_{x_i=v_n})$ filters the values in P removed by each subproblem $AC(P_{x_i=v_k})$

P-1-AC is stronger than SAC

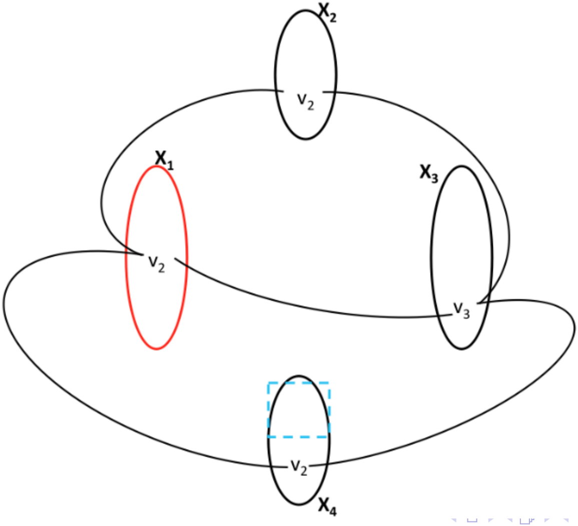
Example of P-1-AC (courtesy by Amine Balafrej)



Example of P-1-AC (courtesy by Amine Balafrej)

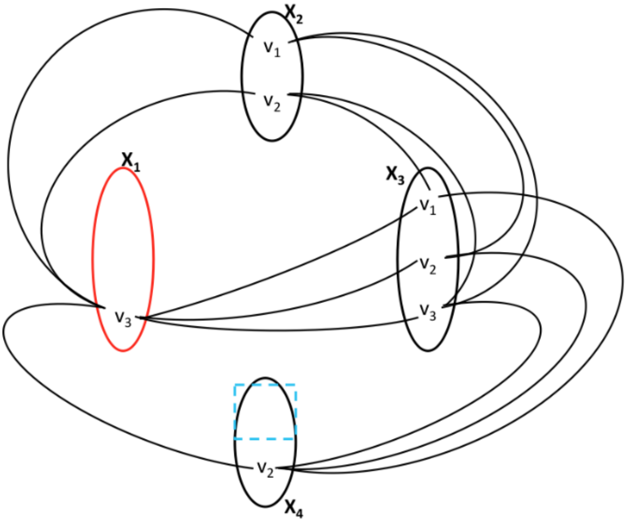


Example of P-1-AC (courtesy by Amine Balafrej)

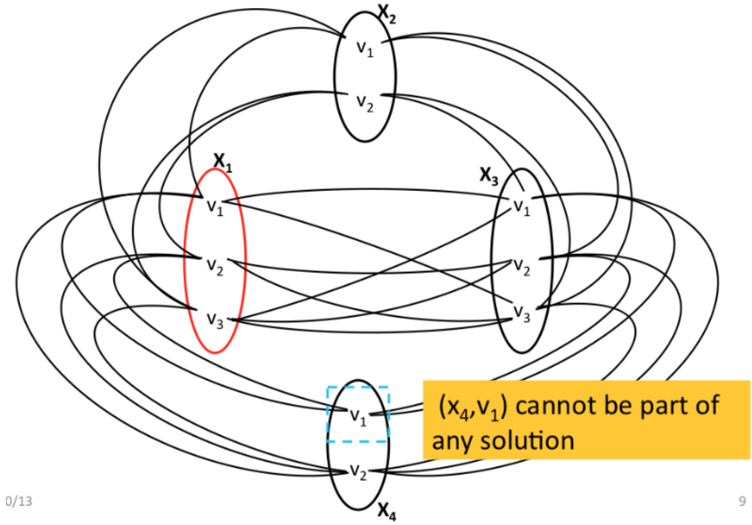


3

Example of P-1-AC (courtesy by Amine Balafrej)

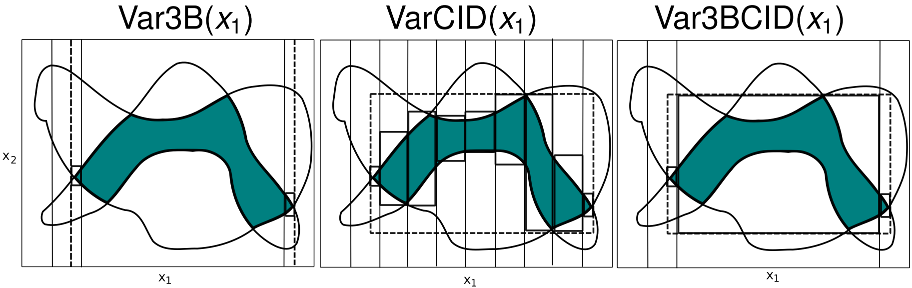


Example of P-1-AC (courtesy by Amine Balafrej)



CID: Adaptation of P-1-AC to NCSP

3B-consistency [Lhomme, IJCAI 1993]
CID-consistency [Trombettoni+Chabert, CP 2007]



Remarks

- Var3BCID can remove values in all dimensions
- CID-consistency only slightly stronger than 3B-consistency in practice
- converges more quickly onto the fixpoint and often in less than n calls to Var3B ($n = $ #variables)
- ACID: new scheme for calling the var3BCID procedure

Outline

1. Brief introduction to interval methods
2. 3B-consistency, CID-consistency
3. ACID
4. Experiments

3B/SAC/CID/3BCID: standard scheme

Two nested loops:

Algorithm `3B` *($P = (x, [x], c)$, ϵ, s_{3B})*

repeat

  $[x]_{old} \leftarrow [x]$

  for every *variable* $x_i \in x$ **do**

    `Var3B` (x_i, s_{3B}, P)

  end

until $w([x]_{old}) - w([x]) \leq \epsilon$

end.

ACID

New scheme with only one loop: a given number numVarCID of variables are “varcided”:

Algorithm `ACID` *($P = (x, [x], c)$, numVarCID, s_{3B}, s_{CID}, h)*

Reorder the variables x with heuristic h

for *i from 1 to numVarCID* **do**

`Var3BCID` (x_i, s_{3B}, s_{CID}, P)

end

end.

New: Use a branching heuristic to order the variables to be varcided

$\Rightarrow$ ACID: filtering algorithm with four parameters: **numVarCID**, s3B/sCID (#slices), **variable selection heuristic** *h*

Main contribution: selecting parameter values

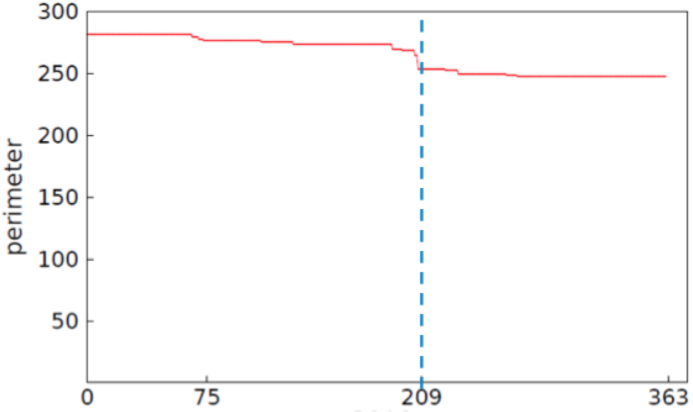
Most parameters experimentally fixed to default values that are robust to modifications:

- heuristic *h*: branching heuristic fixed to smearSumRel
- s_{3B}: #slices fixed to 10
- s_{CID}: #slices fixed to 1

numVarCID is **auto-adapted** during search!

Auto-adaptation of numVarCID

Principle: measure/learn the number of calls to Var3BCID implying a stagnation in filtering (domain size = box perimeter)



#calls to Var3BCID

Auto-adaptation of numVarCID

Divide search into rounds of 1000 nodes each.
Each round interleaves a learning phase (50 nodes)
and an exploitation phase (950 nodes):

- Initialization: k := #variables / 2
- During the search: switch j := #node modulo 1000
- case $j < 50$: short **learning phase**:
  1. achieve 2.k calls to var3BCID
  2. compute/learn the index i in 0..2k corresponding to the last improvement in filtering
     (i s.t. additional filtering from $i + 1$ to 2.k is very small)
- case $j = 50$: k takes the average of the last 50 i values learnt
- case $50 < j < 1000$: long **exploitation phase**:
  Call ACID with numVarCID $= k$

Detailed pseudo-code and other two (less effective) policies are described in the paper.

Outline

1. Brief introduction to interval methods
2. 3B-consistency, CID-consistency
3. ACID
4. Experiments

Protocol

Tests on **constraint satisfaction** instances:
all the 26 instances from the COPRIN benchmark suite solved in $[2, 3600]$ seconds by ACID or a competitor.

Tests on **constrained global optimization** instances:
all the 40 instances from the Coconut benchmark suite solved in $[2, 3600]$ seconds by ACID or a competitor.

Competitors: same satisfaction or optimization strategy differing only in the constraint programming filtering operator:

- HC4: constraint propagation
- 3BCID-fp: standard operator (old scheme with two nested loops)
- 3BCID-n: simplification of ACID with no auto-adaptation:
  the number of vacided variables is fixed to *n*
  (i.e., every variable is varcided once)

Results in constraint satisfaction (synthesis)

	ACID	HC4	3BCID-fp	3BCID-n
#solved instances $<$ 3,600	26	20	23	24
#solved instances $<$ 10,000	26	21	26	26
Average gain factor	1	1.42	1.20	1.09
Maximum gain factor	1	7.69	3.85	1.72
Maximum loss	0	4%	26%	14%
Total time	23,594	$>$72,192	37,494	27,996
Total gain factor	1	>3	1.59	1.19

gain factor = time(X) / time(ACID)

loss = (time(ACID) $-$ time(X)) / time(X)

Details in the paper $\Rightarrow$ ACID is never the worst,
is often (14 on 26) the best, is never far from the best.

Results in constrained optimization (synthesis)

	ACID	HC4	3BCID-fp	3BCID-n
#solved instances	40	40	40	40
Average gain factor	1	1.11	1.30	1.14
Maximum gain factor	1	5.88	3.57	2.85
Maximum loss	0	47%	4%	23%
Total time	9,380	10,289	12,950	11,884
Total gain factor	1	1.10	1.39	1.26

Details in the paper $\Rightarrow$

- ACID is never the worst,
- is the best for 12 on 40 instances,
- not far from the best ($<$ 10% loss in performance) on 23 instances,
- loses between 10% and 47% on 5 instances.

Conclusion

Compared to 3B (close to SAC in CSP) or HC4 (constraint propagation):

- ACID requires no additional parameter (all parameters fixed or auto-adapted)
- ACID brings the best performance on average
- the losses in performance are rare and small
- the gains in performance are sometimes significant

Remark: Differences lowered by other significant algorithmic operators in the strategy: Interval Newton in satisfaction, X-Newton and upperbounding in optimization

$\Rightarrow$ **ACID added to the by-default satisfaction and optimization strategies of Ibex**