

Monotone Hull-Consistency: An Interval Contractor Using Monotonicity

Ignacio Araya, Bertrand Neveu, **Gilles Trombettoni**

INRIA Sophia Antipolis, University of Nice–Sophia, CERTIS, COPRIN research team, France

SWIM, June the 10th, 2009

Outline

- 1 Motivations and background
- 2 The MoHC algorithm
- 3 Experiments

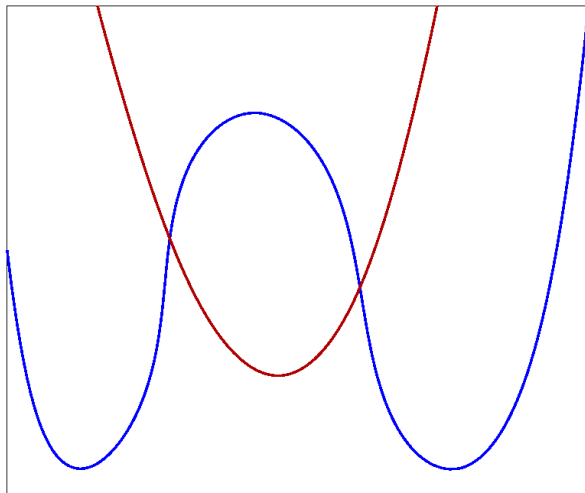
Solving systems of constraints with interval methods

A **constraint** designs an **equation** or an inequality over the reals.

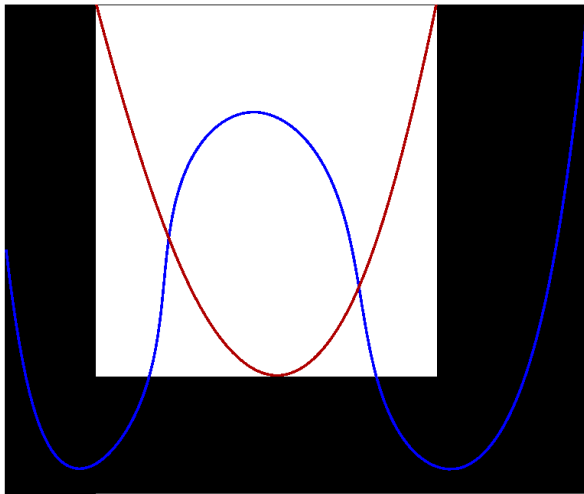
Interval-based **branch and prune/contract** strategy for finding **all** the real solutions to a system of constraints inside an initial box $[b_0]$:

- $L \leftarrow \{[b_0]\}$
- While $L \neq \emptyset$ do:
 - 1 Choose the first box $[b]$ in L .
 - 2 **Contract** $[b]$ with **interval constraint programming (ICP) contractors**.
 - 3 **Interval Newton**: Contract $[b]$ and **certify/guarantee** unicity of solution inside $[b]$.
 - 4 If $[b] \neq \emptyset$ and $Diam([b]) < \epsilon$, then $[b]$ is “solution”.
 - 5 **Branch**: If $[b] \neq \emptyset$ and $Diam([b]) > \epsilon$, then: $[b]$ is **split/bisected** on one dimension (variable) $\Rightarrow [b_1]$ and $[b_2]$.
 - 6 $L \leftarrow L \cup \{[b_1]\} \cup \{[b_2]\}$

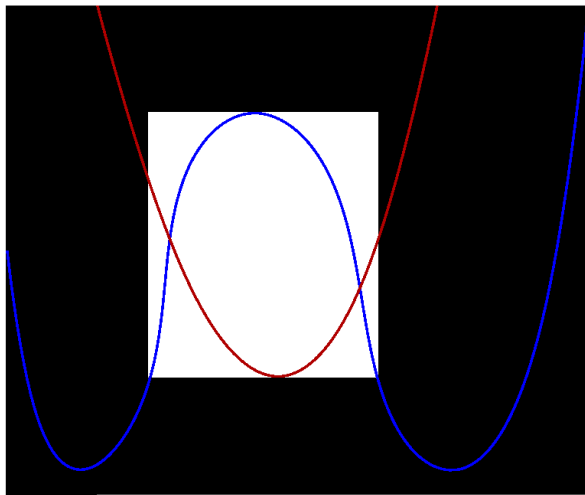
Example of “Branch and prune/contract” scheme



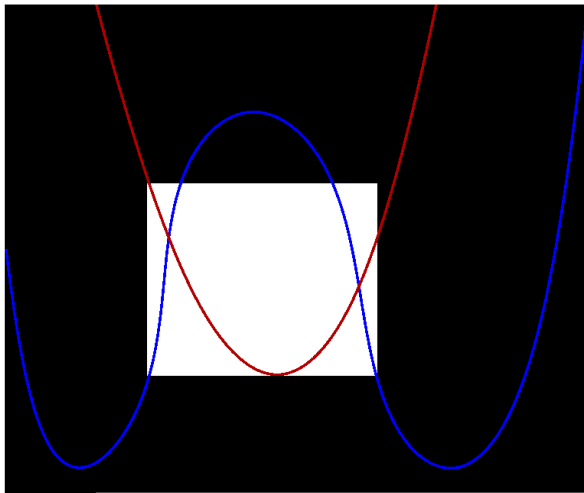
Example of “Branch and prune/contract” scheme



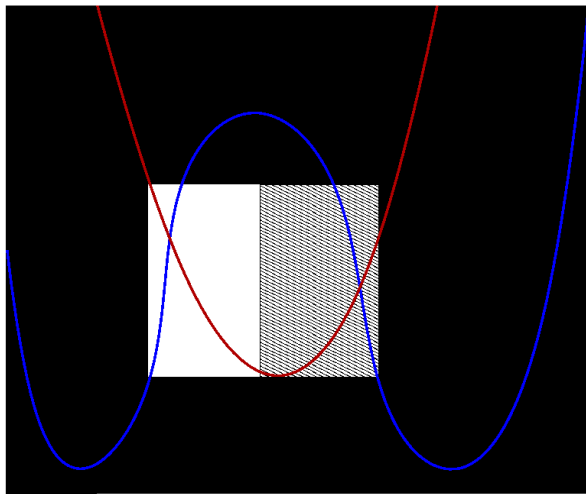
Example of “Branch and prune/contract” scheme



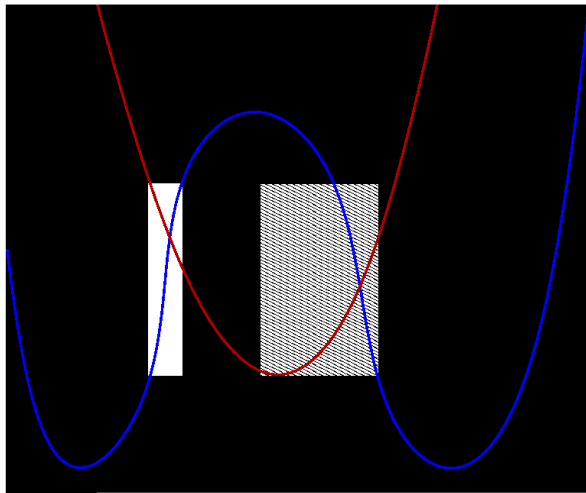
Example of “Branch and prune/contract” scheme



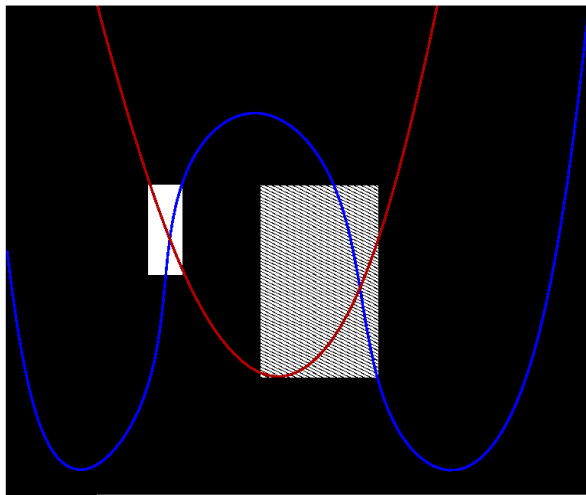
Example of “Branch and prune/contract” scheme



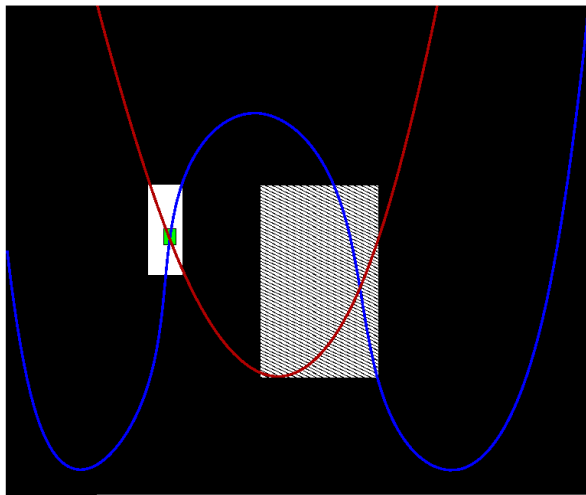
Example of “Branch and prune/contract” scheme



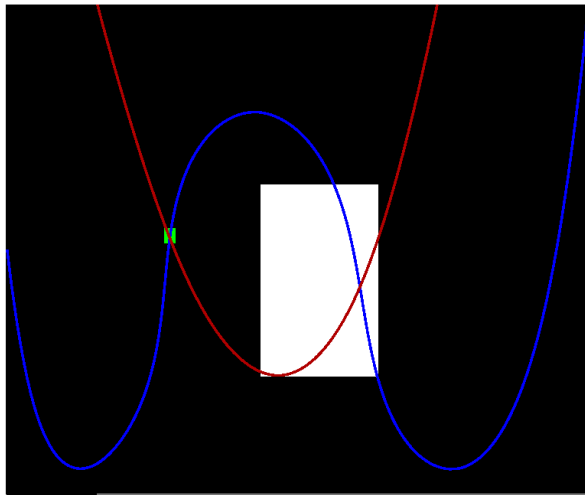
Example of “Branch and prune/contract” scheme



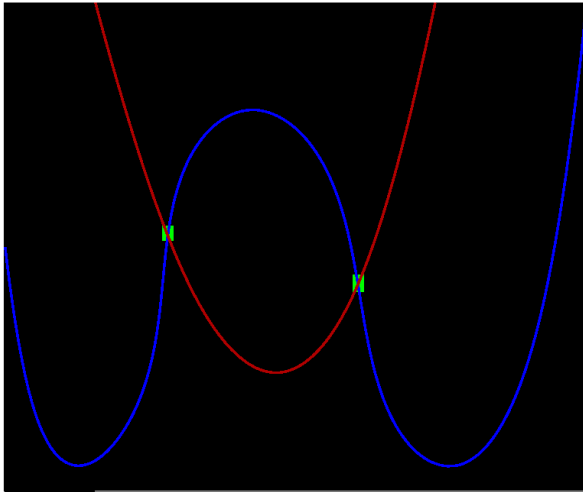
Example of “Branch and prune/contract” scheme



Example of “Branch and prune/contract” scheme



Example of “Branch and prune/contract” scheme



Contraction and constraint propagation in ICP

Contraction

- Definition: Narrowing of the current box (on the bounds) with no loss of solution (in polynomial time)
- Contraction algorithm \equiv contractor

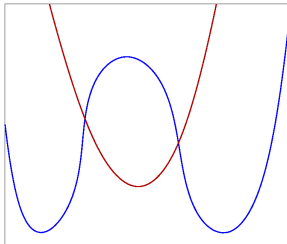
Constraint propagation

- Two procedures: `ConstraintPropagation(Revise)`
- **Revise** procedure: the current box is contracted w.r.t. **one** constraint.
- **Propagation** procedure: intersection of individual constraint boxes with an **incremental** propagation.

Different *Revise* procedures

Hull-consistency

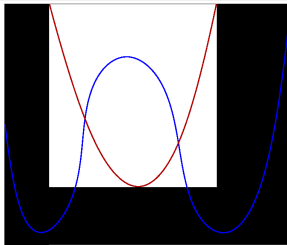
- Computing the optimal/sharpest/smallest box that contains all the solutions to **one** constraint is difficult.
- Computing the optimal box
≡ Verifying the **2B/hull-consistency**.
- The difficulty comes from **multiple occurrences of variables**:
the **dependency problem**.



Different *Revise* procedures

Hull-consistency

- Computing the optimal/sharpest/smallest box that contains all the solutions to **one** constraint is difficult.
- Computing the optimal box
≡ Verifying the **2B/hull-consistency**.
- The difficulty comes from **multiple occurrences of variables**:
the **dependency problem**.



Different *Revise* procedures

- HC4-Revise (in fact, a variant called TAC-Revise) computes the optimal box when the constraint contains **no** multiple occurrences of variables.
- Compared to HC4-Revise, Box-Revise is more costly but computes a sharpest box when the constraint contains **one** variable appearing several times.
- A new algorithm called MoHC-Revise handles better the dependency problem, when **several** variables occur several times.

MoHC-Revise better exploits the monotonicity of interval functions.

Exploiting monotonicity in image calculation

Example (with an univariate function)

- $[y] = [f](x) = x^3 - 3x^2 + x$
Natural extension: $[f]([3, 4]) = [-18, 41]$
- $f'(x) = 3x^2 - 6x + 1$. $f'([3, 4]) = [3, 30] > 0$.
 $\Rightarrow [f]$ is monotonic (increasing) in $[x] = [3, 4]$.
- $[y] = [f(3), f(4)] = [3, 20]$.

\Rightarrow If $[f]$ (depending on several variables) is monotonic in $[x]$, then the dependency problem related to x disappears.

“General case”

- $[y] = [f](x_i, x_d, z)$ s.t. $[f]$ is increasing in $[x_i]$, decreasing in $[x_d]$.
- $[y] = [[f]_{min}(x_i, x_d, z), \overline{[f]_{max}(x_i, x_d, z)}]$, with:
 - $[f]_{min}(x_i, x_d, z) = [f](\underline{[x_i]}, \overline{[x_d]}, [z])$
 - $[f]_{max}(x_i, x_d, z) = [f](\overline{[x_i]}, \underline{[x_d]}, [z])$

Outline

- 1 Motivations and background
- 2 The MoHC algorithm
- 3 Experiments

Revise procedure of MOnotonic Hull-Consistency

MoHC-Revise ($c, X = x_1, \dots, x_k, \tau_{mohc}, \#slices; \text{in-out } [b]$)

HC4-Revise($c, [b]$)

```

if MultipleOccurrences( $c$ ) and ratioMoHC[ $c$ ] <  $\tau_{mohc}$  then
    ( $[f]_{min}, [f]_{max}, \text{Gradient}$ )  $\leftarrow$  preProcessing( $c$ )
    MinMaxRevise( $c, X, [b], [f]_{min}, [f]_{max}$ )
    MonotonicBoxNarrow( $c, X, [b], \#slices, [f]_{min}, [f]_{max}, \text{Gradient}$ )
end if
    
```

- Gradient is a table of size k : $\text{Gradient}[i] = \frac{\partial f}{\partial x_i}([b])$
- preProcessing computes Gradient, performs **occurrence grouping** (see companion presentation), and calculates the corresponding $[f]_{min}$ and $[f]_{max}$ functions based on monotonicity.
- MinMaxRevise narrows the intervals of vars appearing **once** (and those on which f is not monotonic).
- MonotonicBoxNarrow narrows the intervals of variables appearing **several times** (on which f is monotonic).

One word about τ_{mohc} and $\text{ratioMoHC}[c]$

- The user-defined parameter $\tau_{mohc} \in [0, 1]$ allows the monotonicity-based procedures to be called more or less often.

- For every constraint c :

$$\text{ratioMoHC}[c] = \frac{\text{Diam}([\underline{f}]_{\min}([b]), \overline{[f]}_{\max}([b]))}{\text{Diam}([f]([b]))}$$

$\text{ratioMoHC}[c]$ tries to predict whether the monotonicity-based treatment that follows is promising: $\text{ratioMoHC}[c] < \tau_{mohc}$

- $\text{ratioMoHC}[c]$ is computed only once after every bisection (i.e., choice point).

The MinMaxRevise procedure

MinMaxRevise ($c, X, [b], [f]_{min}, [f]_{max}$)

HC4-Revise($[f]_{min} \leq 0, [b]$)

HC4-Revise($[f]_{max} \geq 0, [b]$)

HC4-Revise($[f]_{min}/[f]_{max}$) can narrow only the variables:

- appearing once in c ,
- the interval of which $[f]$ is **not** monotonic on.

The MonotonicBoxNarrow procedure

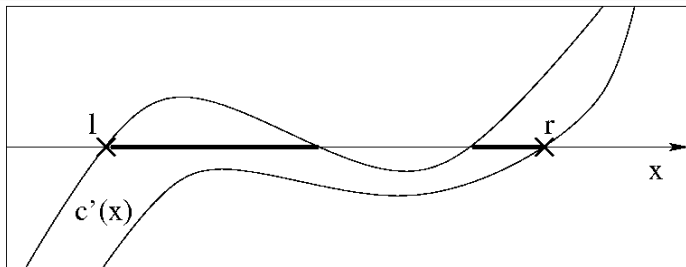
MonotonicBoxNarrow ($c, X, [b], \#slices, [f]_{min}, [f]_{max}, \text{Gradient}$)

```

for all variable  $x_i \in X$  with multiple occurrences in  $c$  do
  if  $\text{Gradient}[i]([b]) \geq 0$  /*  $[f]$  is increasing in  $[x_i]$  */ then
    if  $[f]_{min}([b]) \neq 0$  then  $\text{DichoLeftNarrow}(x_i, [f]_{max} \geq 0, [b])$ 
    if  $[f]_{max}([b]) \neq 0$  then  $\text{DichoRightNarrow}(x_i, [f]_{min} \leq 0, [b])$ 
  else if  $\text{Gradient}[i]([b]) \leq 0$  /*  $[f]$  is decreasing in  $[x_i]$  */ then
    if  $[f]_{max}([b]) \neq 0$  then  $\text{DichoLeftNarrow}(x_i, [f]_{min} \leq 0, [b])$ 
    if  $[f]_{min}([b]) \neq 0$  then  $\text{DichoRightNarrow}(x_i, [f]_{max} \geq 0, [b])$ 
  end if
end for
    
```

- DichoLeftNarrow contracts $[x_i]$ by the left side in $O(\log_2(\#slices))$ evaluation of $[f]_{min}$ or $[f]_{max}$.
- DichoRightNarrow contracts $[x_i]$ by the right side in $O(\log_2(\#slices))$.

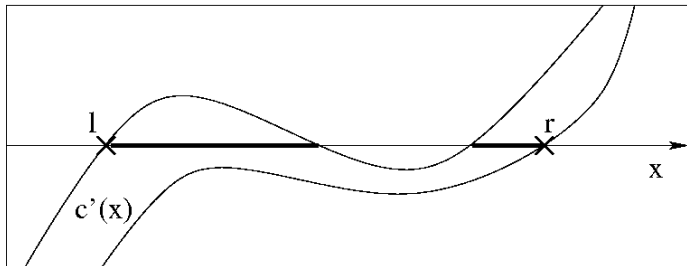
LeftNarrow versus DichoLeftNarrow



Principle of `LeftNarrow` (used in the standard `Box` algorithm):

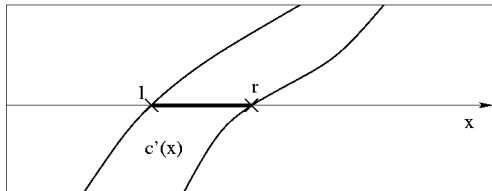
- Consider the univariate constraint $c'(x) = [f](x, [y_1], \dots, [y_k]) = 0$: all the variables except x are replaced by their current interval.
- Compute l , the leftmost zero of c' by splitting $[x]$ in atomic slices $[x_i]$ of size ϵ :
- If $0 \notin [f]([x_i], [y_1], \dots, [y_k])$, then $[x_i]$ is eliminated from $[x]$ and one considers the next slice.

LeftNarrow versus DichoLeftNarrow



- If $0 \in [f]([x_i], [y_1], \dots, [y_k])$, then $[x_i]$ **maybe** (due to overestimation) contains a solution, so that one stops and returns $[x_i]$ as lower approximation of l .
- A “dichotomic” slicing version calls $O(\#slices)$ image calculations.

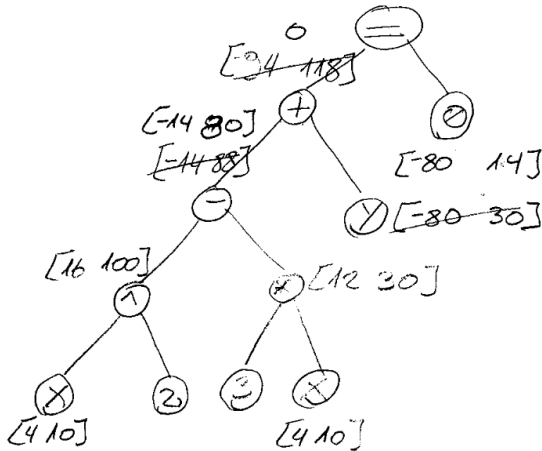
LeftNarrow versus DichoLeftNarrow



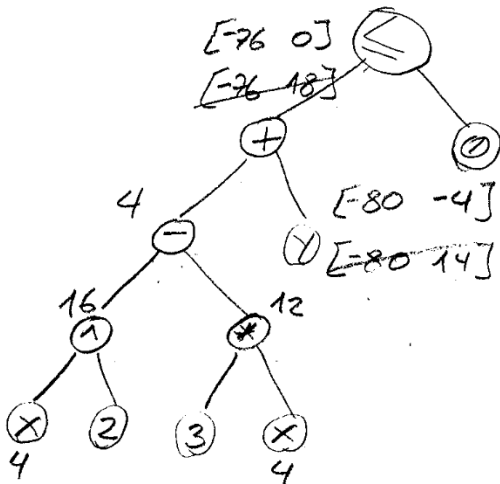
Principle of DichoLeftNarrow, for an increasing function:

- Compute l , the leftmost zero of c by splitting $[x]$ in atomic slices $[x_i]$ of size ϵ :
- If $[f]_{\max}(\overline{[x_i]}, [y_1], \dots, [y_k]) < 0$, then $[x_i]$ is eliminated from $[x]$ and one considers the next slice.
- If $[f]_{\max}(\overline{[x_i]}, [y_1], \dots, [y_k]) \geq 0$, then $[x_i]$ **certainly** contains a “solution”, so that one stops and returns $\underline{[x_i]}$ as lower approximation of l .
- A really dichotomic slicing version calls $O(\log_2(\#slices))$ image calculations.

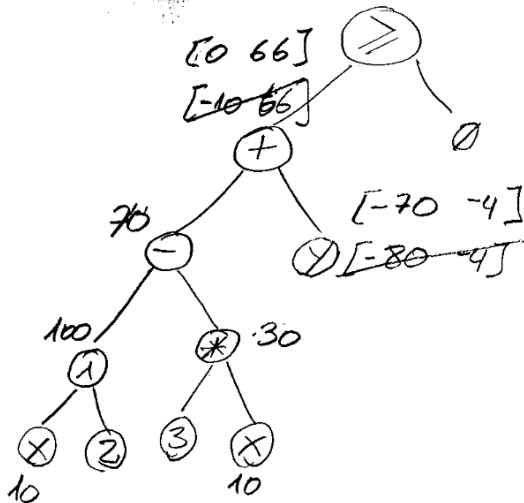
Example: HC4-Revise on $x^2 - 3 \times x + y = 0$



Example: Min-Revise on $x^2 - 3 \times x + y = 0$



Example: Max-Revise on $x^2 - 3 \times x + y = 0$



MoHC and Hull-consistency

Let $c = [f](X, Y) = 0$ with multiple occurrences of $x \in X$ and single occurrence of $y \in Y$.

Theorem 1

If $[f]$ is monotonic on every $[x]$ and on every $[y]$, then:
MoHC-Revise called until fixpoint computes the hull-consistency of c .

Remark: No call to `MonotonicBoxNarrow` for $y \in Y$!

Theorem 2

If $[f]$ is monotonic on every $[x]$ and, inside MoHC-Revise,
HC4-Revise \rightarrow TAC-Revise [Chabert et al., SAC'2005],
then: MoHC-Revise' called until fixpoint computes the hull-consistency of c .

Outline

- 1 Motivations and background
- 2 The MoHC algorithm
- 3 Experiments**

Protocol

- 17 instances from the COPRIN benchmarks containing constraints with multiple occurrences.
- Implementation of two variants of MoHC in the Ibex C++ library (Chabert).
- Strategy: Between two choice points:
 - 1 Monotonicity test,
 - 2 $3BCID(\text{MoHC})$ or $3BCID(\text{LazyMoHC})$.
- Comparison with $3BCID(\text{HC4})$.
- All tests performed on a same computer.

Results obtained by MoHC

MoHC-Revise ($c, X = x_1, \dots, x_k, \tau_{mohc} = 0.9, \#slices; \text{in-out } [b]$)

HC4-Revise($c, [b]$)

if MultipleOccurrences(c) **and** ratioMoHC[c] < 0.9 **then**

 ($[f]_{min}, [f]_{max}, \text{Gradient}$) \leftarrow preProcessing(c)

 MinMaxRevise($c, X, [b], [f]_{min}, [f]_{max}$)

 MonotonicBoxNarrow($c, X, [b], \#slices, [f]_{min}, [f]_{max}, \text{Gradient}$)

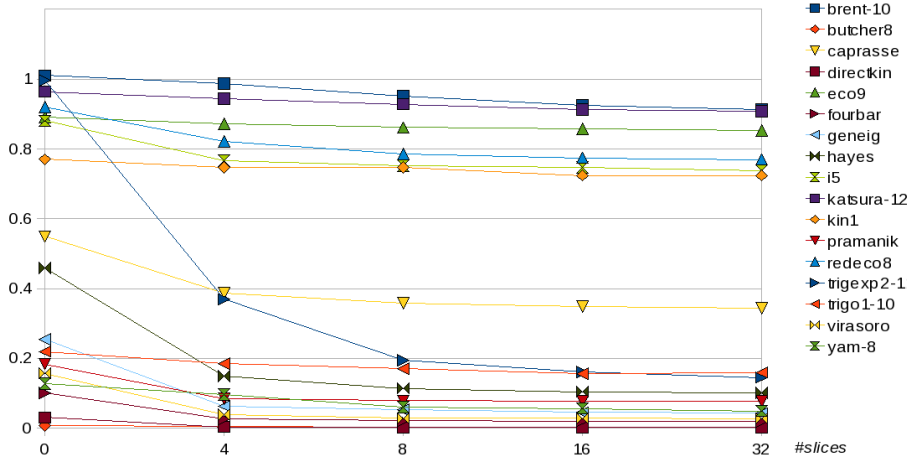
end if

Results obtained by MoHC: # choice points

bench	0	4	8	16	32
brent-10	3963	3871	3733	3629	3579
butcher8	1.4E+6	3.5E+5	2.8E+5	2.3E+5	2.1E+5
caprasse	719	507	469	457	451
eco9	5519	5399	5333	5307	5283
fourbar	97997	27049	21339	20305	19703
geneig	41136	10393	8760	7371	6953
hayes	8153	2655	2035	1867	1799
i5	9353	8135	7985	7941	7843
katsura	4097	4015	3945	3881	3857
kin1	67	65	65	63	63
pramanik	12771	5913	5499	5419	5373
redco8	2245	2005	1917	1891	1879
trigexp2-11	15131	5609	2961	2465	2217
trigol-10	565	477	441	403	411
virasoro	405329	100627	74295	74295	72414
yam-8	385	291	185	173	151
directkin	43898	5923	4009	3399	3151

Results obtained by MoHC: # choice points

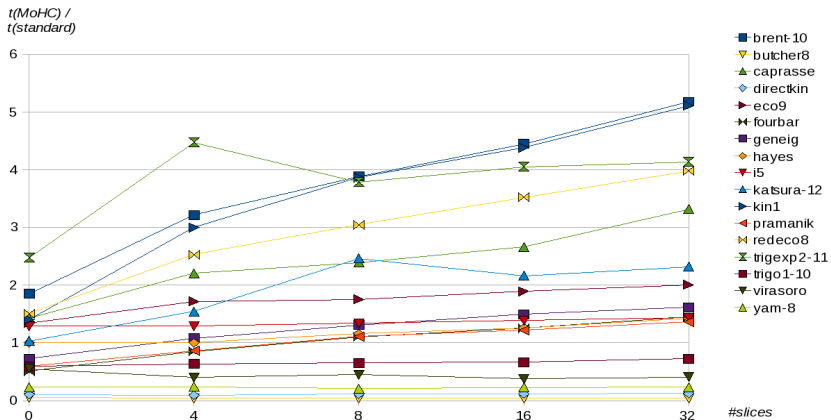
$\frac{\#nodes(MoHC)}{\#nodes(standard)}$



Results obtained by MoHC: CPU time

bench	0	4	8	16	32
brent-10	35	60.75	73.4	84.09	97.95
butcher	5914.5	2576	2121	1998	1871
caprasse	3.91	6.02	6.53	7.28	9.06
eco9	18.61	23.83	24.34	26.26	27.8
fourbar	547	912.9	1179.07	1341.91	1550.04
geneig	282.3	421	511	583	630.73
hayes	41.6	41.77	48.2	52.6	59.61
i5	72.16	72.04	74.73	77.52	80.13
katsura-12	80.17	120.13	191.82	168.51	180.63
kin1	2.75	5.88	7.58	8.6	10.03
pramanik	21.36	31.04	39.87	43.69	48.99
redco8	9.47	15.91	19.17	22.15	25.01
trigexp2-11	225	406.91	344.61	368.81	376.12
trigol-10	89.16	95.62	98.19	100.78	109.48
virasoro	3967.46	2865.94	3230.65	2700.83	2915.22
yam-8	2.68	2.78	2.41	2.6	2.69
directkin	1787	1665	1962	2036	2197

Results obtained by MoHC: CPU time



First conclusions

- MoHC is sometimes fruitful, sometimes counterproductive.
- But, more precisely: MoHC inside 3BCID, with $\tau_{mohc} = 90\%$ and with no call to interval Newton seems to be a not so good algorithm.
- 3B also computes slices. It is known that 3B(HC4) better contracts than Box. This may explain why MoHC performs redundant job.
- We should test MoHC without 3BCID and making vary τ_{mohc} (or finding another prediction test). MonotonicBoxNarrow should use univariate interval Newton.
- We should compare MoHC (without 3BCID) to HC4 and Box.
- Before that \Rightarrow LazyMoHC!

LazyMoHC

Definition: MoHC with no call to MonotonicBoxNarrow.

MoHC-Revise ($c, X = x_1, \dots, x_k, \tau_{mohc}$; **in-out** $[b]$)

```

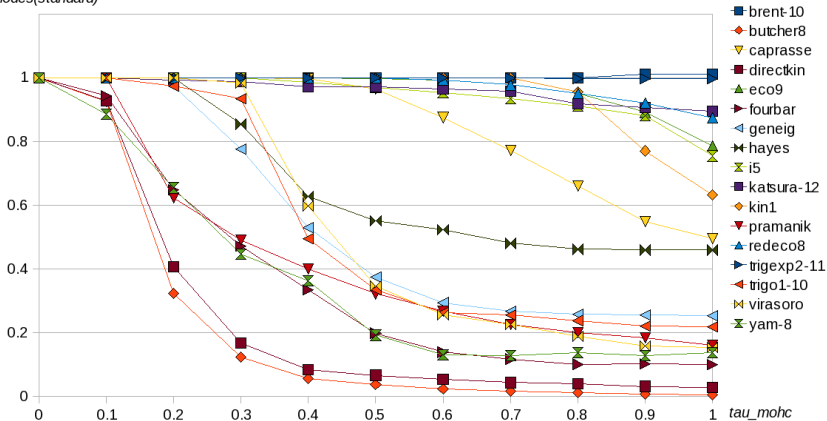
HC4-Revise( $c, [b]$ )
if MultipleOccurrences( $c$ ) and ratioMoHC[ $c$ ] <  $\tau_{mohc}$  then
    ( $[f]_{min}, [f]_{max}, \text{Gradient}$ )  $\leftarrow$  preProcessing( $c$ )
    MinMaxRevise( $c, X, [b], [f]_{min}, [f]_{max}$ )
    /*
    MonotonicBoxNarrow( $c, X, [b], \#slices, [f]_{min}, [f]_{max}, \text{Gradient}$ )
    */
end if
    
```

Results obtained by LazyMoHC: # choice points

bench	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	
brent-10	3923	3923	3923	3923	3923	3923	3923	3923	
butcher	1.8E+8	1.7E+8	5.9E+7	2.3E+7	1.0E+7	6.6E+6	4.3E+6	2.9E+6	2.
caprasse	1309	1309	1309	1309	1305	1261	1145	1011	
directkin	1395697	1294789	567893	234578	115760	92345	75049	60123	5
eco9	6193	6193	6193	6193	6193	6177	6145	6051	
fourbar	965343	910347	625170	455780	322469	191806	133665	113420	9
geneig	161211	161173	157094	125250	85491	60207	47405	42931	4
hayes	17763	17763	17763	15171	11149	9771	9277	8527	
i5	10621	10621	10621	10603	10475	10301	10111	9911	
Katsura-12	4251	4245	4213	4195	4127	4125	4097	4067	
kin1	87	87	87	87	87	87	87	87	
pramanik	69259	69259	43191	33939	27753	22297	18487	15677	1
redco8	2441	2441	2441	2441	2441	2441	2423	2385	
trigexp2-11	15187	15187	15187	15187	15187	15187	15187	15187	1
trigol-1	2565	2565	2495	2397	1271	857	673	655	
virasoro	2562665	2562665	2562665	2519714	1531829	885605	657181	577135	48
yam-8	3017	2673	1973	1349	1095	579	393	383	

Results obtained by LazyMoHC: # choice points

$\frac{\#nodes(MoHC)}{\#nodes(standard)}$



Results obtained by LazyMoHC: CPU time

bench	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
brent-10	18.89	18.95	19.04	19	19.42	18.94	19.41	20.63	27.25
butcher	281859	255231	210188	77480	31866	18332	9334	7530	6471
caprasse	2.73	2.68	2.68	2.68	2.88	3.09	3.61	3.67	3.37
directkin	17507	15410	12159	6231	3878	2710	2364	2215	1989
eco9	13.85	14.12	13.92	14.31	13.92	13.98	14.17	14.35	15.41
fourbar	1069	1051	851	712	657	601	572	552	545
geneig	390	391.86	394.51	363.21	286.67	259.55	250	262.67	273.94
hayes	41.63	41.69	41.71	37.09	33.12	31.68	31.23	31.29	33.53
i5	55.85	57.41	57.34	57.71	56.3	54.99	53.72	58.6	65.33
katsura-12	77.8	78.3	77.64	79	79.85	80.83	80.17	95.6	144
kin1	1.96	1.96	1.96	1.96	1.96	1.96	1.96	1.97	2.29
pramanik	35.9	35.89	30.83	28	26.76	24.49	23.18	21.86	21.24
redco8	6.28	6.28	6.37	6.34	6.31	6.36	6.47	6.77	7.59
trigexp2-11	90.9	93.15	95.84	92.89	91.86	93.12	93.51	92.45	145.34
trigol-10	150.9	151.48	146.82	141.73	104.85	83.69	71.32	78.75	83.64
virasoro	7173	7251	6980	6211	4143	3780	4034	3551	3598
yam-8	11.66	10.39	7.94	5.51	4.67	2.92	2.35	2.44	2.67

Results obtained by LazyMoHC: CPU time

