Node Selection Heuristics Using the Upper Bound in Interval Branch and Bound

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

Imagine, LIGM, Univ. Paris-Est (France)
U. Montpellier (France)
U. Catolica, Valparaiso (Chile)

Global Optimization WS, Malaga, September, 3^{rd}, 2014

Outline

1. Interval Branch & Bound algorithms
2. New node selection heuristics
3. Experiments

Plan

Intervals and boxes

Intervals

Interval $[x_i] = [\underline{x_i}, \overline{x_i}]$	$\{x_i \in \mathbb{R}, \underline{x_i} \leq x_i \leq \overline{x_i}\}$
$\underline{x_i}$ et $\overline{x_i}$	Floating-point bounds
$\mathbb{IR}$	Set of all the intervals
$m([x_i])$	**Midpoint** of $[x_i]$
$w([x_i]) := \overline{x_i} - \underline{x_i}$	**Width** or size of $[x_i]$

Boxes

Box $[x]$	$[x_1] \times ... \times [x_i] \times ... \times [x_n]$ (explored **search space**)
$w([x])$	$\max_n \; w([x_i])$

Interval arithmetic and (natural) interval extension

Interval arithmetic	$[1,2] + [-5,0] = [-4,2]; \quad [-4,2]^2 = [0,16];$
Interval extension $[f]$	$f(x_1, x_2) := (x_1 + x_2)^2; [f]([1,2],[-5,0]) = [0,16]$

Constrained global optimization

- Continuous constrained optimization (NLP):
  $argmin_{x \in [x] \subset \mathbb{R}^n}\ f(x)\ \ s.t.\ \ g(x) \leq 0 \wedge h(x) = 0$
- Output of standard interval solvers (e.g., GlobSol [Kearfott], Icos [Lebbah, Rueher, Michel]):
  a **tiny box** guaranteed to contain a real-valued vector x minimizing:
  $f(x)\ s.t.\ g(x) \leq 0 \wedge h(x) = 0$.
- IBBA and IbexOpt interval solvers handle relaxed equations (because of their upperbounding algorithms):
  $argmin_{x \in [x] \subset \mathbb{R}^n}\ f(x)\ s.t.\ g(x) \leq 0 \wedge (-\epsilon_{eq} \leq h(x) \leq +\epsilon_{eq})$
- Output: a **floating-point vector** x ϵ-minimizing:
  $f(x)\ s.t.\ g(x) \leq 0 \wedge (-\epsilon_{eq} \leq h(x) \leq +\epsilon_{eq})$
- Remark: Most of deterministic global optimizers are not "exact"/valid/reliable/rigorous.

$\Rightarrow$ Our new node selection heuristics apply to all interval branch and bound algorithms.

Example of allowed operators : Coconut ex6_2_9

```
Variables
  x2, x3, x4, x5 in [1e-7,0.5];

Minimize      (31.4830434782609*x2 + 6*x4)*log(4.8274*x2 + 0.92*x4) -
              1.36551138119385*x2 + 2.8555953099828*x4 + 11.5030434782609*x2*
              log(x2/(4.8274*x2 + 0.92*x4)) + 20.98*x2*log(x2/(4.196*x2 + 1.4*
              x4)) + 7*x4*log(x4/(4.196*x2 + 1.4*x4)) + (4.196*x2 + 1.4*x4)*
              log(4.196*x2 + 1.4*x4) + 1.62*x2*log(x2/(7.52678200680961*x2 +
              0.443737968424621*x4)) + 0.848*x2*log(x2/(7.52678200680961*x2 +
              0.443737968424621*x4)) + 1.728*x2*log(x2/(1.82245052351472*x2 +
              1.4300083598626*x4)) + 1.4*x4*log(x4/(0.504772348000588*x2 + 1.4*
              x4)) + (31.4830434782609*x3 + 6*x5)*log(4.8274*x3 + 0.92*x5) -
              1.36551138119385*x3 + 2.8555953099828*x5 + 11.5030434782609*x3*
              log(x3/(4.8274*x3 + 0.92*x5)) + 20.98*x3*log(x3/(4.196*x3 + 1.4*
              x5)) + 7*x5*log(x5/(4.196*x3 + 1.4*x5)) + (4.196*x3 + 1.4*x5)*
              log(4.196*x3 + 1.4*x5) + 1.62*x3*log(x3/(7.52678200680961*x3 +
              0.443737968424621*x5)) + 0.848*x3*log(x3/(7.52678200680961*x3 +
              0.443737968424621*x5)) + 1.728*x3*log(x3/(1.82245052351472*x3 +
              1.4300083598626*x5)) + 1.4*x5*log(x5/(0.504772348000588*x3 + 1.4*
              x5)) - 35.6790434782609*x2*log(x2) - 7.4*x4*log(x4) -
              35.6790434782609*x3*log(x3) - 7.4*x5*log(x5);

Subject to    x2 + x3 = 0.5;    x4 + x5 = 0.5;
```

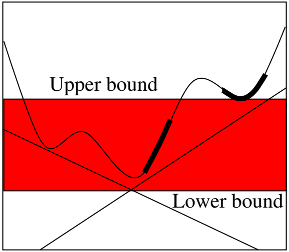
Example of allowed operators : Coconut ex7_2_3

```
Variables
  x1                  in [100,10000];
  x2, x3              in [1000,10000];
  x4, x5, x6, x7, x8 in [10,1000];

Minimize x1 + x2 + x3;

Subject to

 833.33252*x4/x1/x6 + 100/x6 - 83333.333/(x1*x6) <= 1;
 1250*x5/x2/x7 + x4/x7 - 1250*x4/x2/x7 <= 1;
 1250000/(x3*x8) + x5/x8 - 2500*x5/x3/x8 <= 1;

 0.0025*x4 + 0.0025*x6 <= 1;
 -0.0025*x4 + 0.0025*x5 + 0.0025*x7 <= 1;
 -0.01*x5 + 0.01*x8 <= 1;
```

Interval Branch & Bound: ingredients

Interval *Branch & Bound*

1. **node selection**: the box with the smallest lower bound
2. **combinatorial exploration**: the selected box is bisected on one dimension
3. **contraction**: reduction of the box with no loss of solution
4. search for a good lower bound f_{min} of the cost (**lower bounding**): no feasible point has a cost better than the lower bound
5. search for a feasible point with a "good" cost $\widetilde{f}$ (**upper bounding**)



Objective function

Interval Branch & Bound: pseudo-code

Algorithm *IntervalBranch&Bound(B, g, f)*

while $B \neq \emptyset$ **and** $\widetilde{f} - f_{min} > \epsilon_{obj}$ **do**

$[x]$:= bestBox (B, criterion); $B := B \setminus \{[x]\}$
$([x]_1, [x]_2)$:= bisect ($[x]$)
$[x]_1$:= Contract&Bound($[x]_1$, g, f)
$[x]_2$:= Contract&Bound($[x]_2$, g, f)
$B := B \cup \{[x]_1\} \cup \{[x]_2\}$
$f_{min} := \min_{[x] \in B} [x].lb$

end

end.

The procedure Contract&Bound

Added to the system: a cost variable x_{obj} and a constraint $f(x) = x_{obj}$ (for the contraction)

At each node of the search tree:

Algorithm *Contract&Bound*([x], g, f, ...)

- $g' := g \cup \{x_{obj} \leq \widetilde{f} - \epsilon_{obj}\}$
- $[x]$:= contraction($[x]$, $g' \cup \{f(x) = x_{obj}\}$, f)
- **if** $[x] \neq \emptyset$ **then**
  - // Upperbounding:
  - $(x_{ub}$, cost$)$:= FeasibleSearch($[x]$,g')
  - **if** $cost < \widetilde{f}$ **then** $\widetilde{f}$:= cost
- **end**
- **return** $[x]$, x_{ub}

end.

Improving $\underline{x_{obj}}$ by contraction $\equiv$ lower bounding

Plan

1. Interval Branch & Bound algorithms
2. New node selection heuristics
3. Experiments

Motivation

Two phases in a B&B

1. $f^* < \widetilde{f}$: find the optimal solution
2. $|f^* - \widetilde{f}| \leq \epsilon_{obj}$: prove $\widetilde{f}$ is the ϵ-optimal solution
   $\Rightarrow$ **all** the nodes must be handled, **in any order**!
   $\Rightarrow$ the node selection matters only in the first phase

Motivation

The best node to select is the one that improves the most the **upperbound**.

Upper and lower bounds of the objective in a node/box

All the proposed node selection heuristics are based on lower and upper bounds of the objective function in nodes/boxes.

Notation

- $[x].lb$: lower bound of the objective in the box $[x]$
- $[x].ub$: upper bound of the objective in the box $[x]$

Remark: $[x].ub$ does **not** generally correspond to a **feasible point**.
Easy way to compute $[x].lb$ and $[x].ub$:
interval evaluation:

- $[x].lb := \underline{[f]([x])}$
- $[x].ub := \overline{[f]([x])}$

Upper and lower bounds of a node/box

- Claim: better node selection if using more accurate bounds $[x].lb$ and $[x].ub$.

  $\Rightarrow$ Simple idea: using the contraction on $[x_{obj}]$ of the Contract&Bound procedure:

- $[x].lb$: $\underline{x_{obj}}$ is improved by constraint programming contractors and linear relaxation.
- $[x].ub$: $\overline{x_{obj}}$ can also be improved by constraint programming contractors
  (and by linear relaxation - using an additional call to the Simplex algorithm).

$\Rightarrow$ These computations take into account the feasible space!

Computing $[x].ub$

Algorithm *Contract&Bound([x], g, f, ...)*

$g' := g \cup \{x_{obj} \leq \widetilde{f} - \mathbf{0.9}\epsilon_{obj}\}$
$[x] :=$ contraction$([x], g' \cup \{f(x) = x_{obj}\}, f)$
if $[x] \neq \emptyset$ **then**
  $(x_{ub}, \text{cost}) :=$ FeasibleSearch$([x], g')$
  if $cost < \widetilde{f}$ **then**
    $\widetilde{f} := \text{cost}$
    $\mathbf{[x].ub := \widetilde{f}} - \epsilon_{\mathbf{obj}}$
  else
    $[x].ub := \overline{x_{obj}}$
  end
end
return $[x], x_{ub}$
end.

Computing $[x].ub$: priority among boxes

The label $[x].ub$ is:

1. $< \widetilde{f} - \epsilon_{obj}$ if the contraction procedure improved $\overline{x_{obj}}$
2. $= \widetilde{f} - \epsilon_{obj}$, if the box is a descendant of the box having the current best feasible point of cost $\widetilde{f}$
3. $= \widetilde{f} - 0.9\,\epsilon_{obj}$ if the box was handled after the last update of best cost
4. $> \widetilde{f} - 0.9\,\epsilon_{obj}$ otherwise

Node selection criteria

- **LB**: well known criterion used by best-first search Branch&Bound and selecting the box $[x]$ in the set B with a minimum $[x].lb$

  Optimistic criterion: we hope to find a solution with cost f_{min}

- **LB+UB**: selects $[x] \in B$ minimizing $[x].lb + [x].ub$
  $\equiv$ minimize $[x].lb$ and $[x].ub$ with the same weight
  $\equiv$ minimize the middle of the cost interval

Node selection criteria: LBvUB

LBvUB

At each node selection, a **random choice** is made for selecting the node using:

- UB, with probability p, or
- LB, with probability $1 - p$.

Intuition: Using two criteria avoids the drawbacks of the two criteria individually, i.e.:

- **LB drawback**: choosing promising boxes with no feasible point
- **UB drawback**: going deeply in the search tree where only slightly better solutions will be found trapped inside a local minimum

Remark: UB (resp. LB) is used to tie breaks between nodes chosen by LB (resp.UB).

Data structure for storing the nodes in LBvUB

Each node $[x]$ in the set B of nodes is labeled with two values: $[x].lb$ and $[x].ub$. Several implementation choices:

1. **One heap** sorted on $[x].lb \Rightarrow$ selection of the best $[x].ub$ in linear time
   Observation: 10% of the total time spent on the heap management when $|B| > 50{,}000$
2. For LBvUB, when $|B|$ exceeds 50,000,
   change the probability p to 0.1
   (fewer calls to the $[x].ub$ minimization criterion)
3. **Two heaps**, one for LB, one for UB
4. Variant: **periodic reconstruction** (PR) of the two heaps (every 50 or 100 iterations)
   Goal: break ties among nodes having the same $[x].lb$ and $[x].ub$

Summary of the LBvUB node selection heuristic

- Values $[x].lb$ and $[x].ub$ computed by **contraction** taking into account the **feasible region**
- At each node, **random choice** between LB and UB criteria.
- **Two heaps**
  - One heap for sorting nodes according $[x].lb$
  - A second heap for sorting nodes according $[x].ub$
  - A diversification device (e.g., heap reconstruction) to break ties
- That's it!

Plan

1. Interval Branch & Bound algorithms
2. New node selection heuristics
3. Experiments

Implementation in Ibex

Implemention with the **Ibex** free C++ library
(Ibex: Interval-Based EXplorer)
and its **IbexOpt** global optimization strategy.

Project leader: Gilles Chabert (EMN/LINA, Nantes)
Other contributors:

Ignacio Araya	U. Catolica, Valparaiso
Bertrand Neveu	LIGM, Paris
Jordan Ninin	ENSTA, Brest
Gilles Trombettoni	LIRMM, U. Montpellier

Ingredients in IbexOpt [IbexTeam AAAI 2011]

- **Branching heuristic**: variant of the smear function ([Kearfott 2010])
- Contraction and lower bounding:
  - Interval constraint programming contractors: HC4 [Messine thesis 1997, Benhamou+al. ICLP 1998], Mohc [IbexTeam AAAI 2010], ACID [IbexTeam CP 2007]
  - Interval-based polyhedral relaxation algorithms: X-Taylor [IbexTeam CPAIOR 2012], ART (affine arithmetic) [Ninin+Messine 2010]
- Upper bounding based on **inner region extraction in the feasible space**: inHC4, in-XTaylor [IbexTeam JOGO 2014]
- **No** lagrangian method, **no** convexity analysis (for the moment)

Benchmark and main results

Benchmark

All the instances from the series 1 and 2 of the Coconut constrained global optimization benchmark that

- are solved in a runtime of [1, 3600] seconds by one competitor (cost precision $\epsilon_{obj} = 1e\text{-}8$)
- using the best branching strategy among **smear sum** absolute (ssa) or relative (ssr), **smear max** (sm), **largest interval first** (lf) and **round robin** (rr)
- have [6, 50] variables

$\Rightarrow$ 82 instances

Main results

Our best node selection strategy:

- obtains a gain of about 40% w.r.t. LB in total time and 23% on average
- obtains a significant gain on several instances
- is robust: no significant loss w.r.t. LB

Sample of 82 instances

Name	Branching	Name	Branching	Name	Branching	Name	Branching
ex2_1_9	ssr	ex8_2_1	ssa	linear	ssr	hs088	lf
ex3_1_1	ssr	ex8_4_4	ssr	meanvar	ssr	hs093	ssr
ex5_3_2	ssr	ex8_4_5	lf	process	ssr	hs100	ssr
ex5_4_3	ssr	ex8_4_6	ssr	ramsey	lf	hs103	ssr
ex5_4_4	ssa	ex8_5_1	ssr	sambal	rr	hs104	lf
ex6_1_1	ssr	ex8_5_2	ssr	srcpm	sm	hs106	lf
ex6_1_3	ssr	ex8_5_6	ssr	avgasa	ssr	hs109	ssr
ex6_1_4	ssr	ex14_1_2	ssr	avgasb	ssr	hs113	lf
ex6_2_6	ssr	ex14_1_6	ssr	batch	ssa	hs114	rr
ex6_2_8	ssr	ex14_1_7	ssr	dipigri	ssr	hs117	ssa
ex6_2_9	ssr	ex14_2_1	ssr	disc2	ssr	hs119	ssa
ex6_2_10	ssr	ex14_2_3	ssr	dixchlng	lf	makela3	ssr
ex6_2_11	ssr	ex14_2_7	ssr	dualc1	ssr	matrix2	lf
ex6_2_12	ssr	alkyl (rr)	lf	dualc2	ssr	mistake	ssa
ex7_2_3	ssr	bearing	ssr	dualc5	ssr	odfits	ssr
ex7_2_4	lf	hhfair	ssr	genhs28	lf	optprloc	ssr
ex7_2_8	lf	himmel16	ssr	haifas	ssr	pentagon	ssr
ex7_2_9	lf	house	ssr	haldmads	lf	polak5	ssr
ex7_3_4	ssr	hydro	ssr	himmelbk	lf	robot	lf
ex7_3_5	ssr	immun	rr	hs056	lf		
ex8_1_8	ssr	launch	ssr	hs087	ssr		

Main results

Gain/Loss w.r.t. LB $= \frac{time(LB)}{time(new\ \ heuristic)}$

CPU time and number of nodes gains/losses w.r.t. the standard LB.

Criterion	#heaps	PR	time max loss	time max gain	time avg gain	time total gain	nodes avg gain	nodes total gain
LB+UB	1	no	0.17	23.8	1.20	1.47	1.25	1.59
LB+UB	1	100	0.14	10.2	1.21	1.54	1.28	1.70
LBvUB	1	no	0.52	9.17	1.17	1.58	1.20	1.75
LBvUB-01	1	no	0.52	21.7	1.18	1.64	1.19	1.67
LBvUB	2	no	0.47	13.7	1.27	1.51	1.23	1.45
LBvUB	2	50	0.46	21.7	1.28	1.67	1.26	1.83
LBvUB	2	100	0.48	15.9	1.30	1.68	1.26	1.76

PR : frequency of periodic reconstruction of heaps

LBvUB-01: LBvUB with $p = 0.1$ when the number of nodes exceeds 50,000.

Main results

Details of the gains/losses in CPU time w.r.t. LB

strategy	# heaps	PR	gain > 5	gain $[2, 5]$	gain $[1.2, 2]$	gain $[1.05, 1.2]$	equiv. $[0.95, 1.05]$	loss $[0.8, 0.95]$	loss $[0.5, 0.8]$	loss < 0.5
LB+UB	1	no	4	5	28	23	17	4	0	1
LB+UB	1	100	4	10	28	23	12	4	1	1
LBvUB	1	no	2	6	22	22	20	7	3	1
LBvUB-01	1	no	2	6	22	23	18	8	3	1
LBvUB	2	no	4	7	29	26	10	5	0	1
LBvUB	2	50	4	7	29	21	15	5	0	1
LBvUB	2	100	4	8	29	25	11	4	0	1

PR : frequency of periodic reconstruction of heaps

LBvUB-01: LBvUB with $p = 0.1$ when the number of nodes exceeds 50,000.

Main differences between LB and LBvUB

CPU times in second; timeout 10,000 seconds

Name	#variables	LB time	LB avg. time	LBvUB time	LBvUB avg. time	remark
disc2	28	7/10 runs	2278	[50, 693]	142	quadratic
srcpm	38	[22, 372]	244	[19, 24]	22	convex
launch	39	[92, 1414]	670	[57, 117]	82	fast Baron
bearing	14	[6, 124]	29	[5, 16]	8	IbexOpt modified
immun	22	[9, 24]	16	[3, 6]	5	convex
hhfair	14	[8, 15]	12	[4, 5]	4.5	domains bounded
ex5_4_4	27	[310, 324]	317	[140, 160]	149	Baron 1e-6: 159
ex14_1_7	10	[425, 479]	450	[189, 273]	232	
robot	14	[803, 956]	908	[469, 630]	543	trigonometric
hs088	33	[264, 332]	299	[241, 910]	619	
ex7_3_4	12	[2.4, 2.8]	2.6	[2.7, 3.3]	3	

Observation (to be confirmed): significant gains seem to be obtained on problems where IbexOpt has difficulties for finding good feasible points.

Other experiments

- The probability p in LBvUB has a slight impact on performance, provided $p \in [0.2, 0.8]$
  $\Rightarrow$ p has been fixed to 0.5.
- LBvUB with one additional call to the Simplex algorithm on $\overline{x_{obj}}$
  $\Rightarrow$ similar results
- To be more symetric between lower bound and upper bound of a box, we tried to call an independent procedure in Contract&Bound, just for computing $[x].ub$ and $[x].lb$:

  The procedure contracts the extended system (with $x_{obj} = f(x)$) but without the constraint $x_{obj} < \widetilde{f} - \epsilon_{obj}$

  $\Rightarrow$ avg. gain in #nodes = 1.03; total gain in #nodes = 1.16

Criteria C3, C5, C7 by Markot et al.

Criteria C3, C5, C7 by Markot et al.

- Maximize criterion $C_3 = \frac{f^* - lb}{ub - lb}$ with $[lb, ub] = [f]([x])$
- Maximize criterion $C_5 = C_3 \times fr$ (fr a feasibility ratio)
- Minimize $C_7 = \frac{lb}{C_5}$.
- The three criteria give worse results than LB on the tested instances, but...

LBvC3

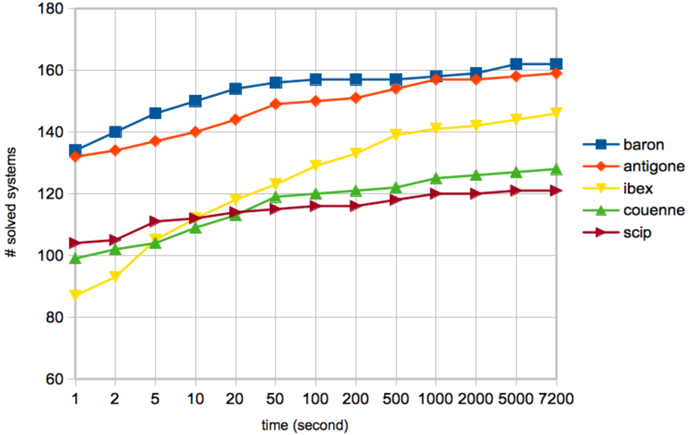
LBvC3 gives interesting results:

Criterion	max loss	max gain	avg gain	total gain
LB	1	1	1	1
LBvC3 (time)	0.44	7.4	1.16	1.26
LBvC3 (nodes)			1.05	1.48
LBvUB (time)	0.48	15.9	1.30	1.66

Performance profile

Comparison on the 176 constrained global optimization instances appearing in the Antigone experiments (JOGO 2014) (no trigonometric operators).
Systems in series 1 (resp. 2) of the Coconut benchmark having $[1, 50]$ (resp. $[6, 50]$) variables.

Several incorrect results obtained by SCIP and Couenne.



Results on the instances not tested in Antigone's JOGO article

Name	Type	Time (s)
ex7_2_5		0.03
ex7_2_6		0.07
ex7_2_7		0.16
ex7_2_8		9.86
ex7_2_9		> 7200
ex7_2_10		0.03
ex8_1_1	trigonometric	0.004
ex8_1_ 2	trigonometric	0.01
ex8_1_ 8		0.25
hs056	trigonometric	2.89
hs087	trigonometric	0.37
hs109	trigonometric	1.53
robot	trigonometric	1018

Conclusion

Promising node selection heuristics based on

- good accuracy of lower bound $[x].lb$ and upper bound $[x].ub$ of each node,
  (cheap overhead)
- LBvUB: The criterion $[x].lb$ or $[x].ub$ is selected randomly at each node for avoiding the drawbacks of each individually.

Good results obtained on representative and difficult instances.

Some other ingredients should be studied like the feasibility ratio proposed by Markot et al. in criterion C5.

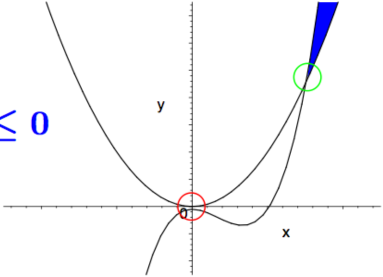
Thank you

Questions?

Lack of rigor of deterministic branch and bounds

Consider the following optimisation problem:

$$\begin{array}{ll} \min & x \\ \text{s. t.} & y - x^2 \geq 0 \\ & y - x^2 * (x-2) + 10^{-5} \leq 0 \\ & x, y \in [-10, +10] \end{array}$$



Baron 6.0 and Baron 7.2 find 0 as the minimum ...

(Courtesy by Yahia Lebbah and Michel Rueher)