Journal of Global Optimization manuscript No.
(will be inserted by the editor)

Node Selection Strategies in Interval Branch and Bound Algorithms

Bertrand Neveu · Gilles Trombettoni · Ignacio Araya

Received: date / Accepted: date

Abstract We present in this article new strategies for selecting nodes in interval Branch and Bound algorithms for constrained global optimization. For a minimization problem the standard best-first strategy selects a node with the smallest lower bound of the objective function estimate. We first propose new node selection policies where an *upper bound* of each node/box is also taken into account. The good accuracy of this upper bound achieved by several contracting operators leads to a good performance of the node selection rule based on this criterion. We propose another strategy that also makes a trade-off between diversification and intensification by greedily diving into potential feasible regions at each node of the best-first search. These new strategies obtain better experimental results than classical best-first search on difficult constrained global optimization instances.

1 Introduction

The paper deals with continuous global optimization (nonlinear programming) deterministically handled by interval Branch and Bound (B&B). Several works have been performed for finding good branching strategies [10], but little work for the node selection itself. The interval solvers generally follow a *best*

Ignacio Araya is supported by the Fondecyt Project 11121366.

B. Neveu
Imagine LIGM Université Paris–Est, France
E-mail: Bertrand.Neveu@enpc.fr

G. Trombettoni
LIRMM, University of Montpellier, CNRS, France
E-mail: Gilles.Trombettoni@lirmm.fr

I. Araya
Pontificia Universidad Católica de Valparaíso, Chile
E-mail: rilianx@gmail.com

first search strategy, with some studies for limiting its exponential memory growth [6,20].

To our knowledge, the only node selection strategies for interval B&B algorithms were proposed by Casado et al. in [6], Csendes in [9] and Markot et al. in [14]. One criterion to maximize is suitable for unconstrained global minimization, called C_3 in [14]:

$$C_3 := \frac{f^* - lb}{ub - lb}$$

$[lb, ub] = [f]([x])$ is the interval obtained by an *interval evaluation* of the real-valued objective function f in the current box $[x]$, i.e. $[lb, ub]$ is a range interval that includes all real images of any point in $[x]$ by f. f^* is the minimal objective value of any feasible point in the studied domain. Since f^* is generally not known, $\widetilde{f}$, the objective value of the best feasible point found so far, can be used as an approximation of f^*. This criterion favors boxes with tight objective range $[lb, ub]$ and nodes with good lb. For constrained optimization, another criterion (to maximize), numbered C_5, is equal to $C_3 \times fr$. It takes into account a *feasibility ratio* fr computed from all the inequality constraints. The criterion C_7 proposes to minimize $\frac{lb}{C_5}$.

Other node selection strategies have been studied for deterministic (while non interval-based) B&B algorithms. Depth-first, breadth-first or best-first search strategies are the most commonly used strategies in existing optimization codes. It is important to understand that depth-first search favors the exploitation (intensification) of the search space because the search goes down towards the feasible solutions inside the region visited. Best-first search favors the exploration (diversification) in that nodes explored consecutively can belong to significantly different regions. Also, best-first search selects a node with small lower bound, so with the greatest potential improvement of the objective value.

Different node selection strategies are proposed in the Solving Constraint Integer Programs (SCIP) optimization code [22]. Depth-first, breadth-first and best-first search strategies are available. With the option `restartdfs`, the SCIP solver performs a depth first search, but periodically (i.e., every $k = 100$ backtracking steps) selects the best node. The most sophisticated strategy, called Upper Confidence bounds for Trees (UCT) [13], is used for handling the mixed integer linear programming subclass. This strategy keeps all the nodes of the search tree, including the closed ones, and maintains a label (UCT score) for each of them depending on its evaluation and on the number of times it has been visited. At each iteration, the algorithm traverses again the search tree from the root until an open node (i.e., a leaf) is reached by following, at each level, the child whose UCT score is the higher. The score of the visited nodes are decreased in order to diversify the search (i.e., to avoid that the same path is followed every time). A conclusion found in [22] is that UCT is costly but can improve the performances when limited to the first nodes of the search.

The node selection strategy of the `Baron` global optimization solver is briefly described in [24]. The default node selection rule switches between

the node with the minimal violation and the one with a smallest lower bound. When memory limitations become stringent, `Baron` temporarily switches to a depth-first search. The *violation* value of a node is defined by the summation of the violation errors contributed by each variable during node processing. These variable violation errors are computed from the violations of the nonconvex constraints by the solution of the relaxed (convex) problem.

Node selection strategies have also been studied for convex Mixed Integer Non Linear Programs (MINLP). The default strategy of the `Bonmin` solver [5] selects the best lower bound. Depth-first and breadth-first search are also available. Finally, a *dynamic* strategy starts with depth-first search and turns to best-first search once three integer feasible solutions have been found.

The K-Best-First Search [11] (KBFS) has been proposed for selecting nodes in the open list for combinatorial (discrete) optimization problems. KBFS(k) performs node expansion cycles iteratively. In each cycle, the k best open nodes are expanded. Their children are generated and added to the open nodes list before a new cycle is performed. Thus, KBFS makes a tradeoff between a breadth-first and a best-first search. If $k = 1$, then KBFS(1) resorts to a best-first search. KBFS(∞) resorts to breadth-first search since it expands all the nodes at the same level before going to the next level.

This paper proposes new node selection policies in interval Branch and Bound algorithms for constrained or unconstrained optimization.

After presenting the necessary background in Section 2, a first node selection rule is described in Section 3. This approach uses, not only a lower bound of a given node, but also an upper bound of the optimal cost. This upper bound is made more accurate by using contraction methods that eliminate infeasible sub-regions. The adopted policy selects randomly the node with a smallest lower bound or the node with a smallest *upper bound* (to be defined more precisely further).

A second approach makes a simple tradeoff between exploration and exploitation. A best-first interval B&B runs a greedy diving at each node to better focus on feasible regions (see Section 4).

Section 5 carries out an empirical study of different variants of these novel strategies and shows that they perform well on difficult non convex constrained optimization problems belonging to the Coconut benchmark [23].

2 Background and Interval B&B Algorithms

An interval $[x_i] = [\underline{x_i}, \overline{x_i}]$ defines the set of reals x_i s.t. $\underline{x_i} \leq x_i \leq \overline{x_i}$. A *box* $[x]$ is a Cartesian product of intervals $[x_1] \times ... \times [x_i] \times ... \times [x_n]$.

The paper deals with continuous global optimization under inequality constraints defined by:

$$\min_{x \in [x]} f(x) \quad s.t. \quad g(x) \leq 0$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the real-valued objective (non convex) function and $g : \mathbb{R}^n \to \mathbb{R}^m$ is a vector-valued (non convex) function.[1] $x = (x_1, ..., x_i, ...x_n)$ is a vector of variables varying in a domain (i.e., a box) $[x]$. A point x is said to be *feasible* if it satisfies the constraints.

An interval (or spatial) B&B scheme for continuous constrained global optimization (also known as nonlinear programming) is described below. Algorithms 1 and 2 correspond to the algorithms implemented in `IbexOpt` [26] and `IBBA` [15,21] solvers and help understanding the node selection strategies that follow. However, these strategies can be embedded in any interval Branch and Bound algorithm.

The algorithm is launched with the vector of constraints g, the objective function f and with the input domain initializing a list $Boxes$ of boxes to be handled. ϵ_{obj} is the absolute or relative precision required on the objective function and is used in the stopping criterion. Therefore, the algorithm computes a feasible point of cost $\widetilde{f}$ such that no other solution exists with a cost lower than $\widetilde{f} - \epsilon_{obj}$. We add a variable x_{obj} in the problem (to the vector x of variables) corresponding to the objective function value, and a constraint $f(x) = x_{obj}$.

The B&B algorithm maintains two main types of information during the iterations:

- $\widetilde{f}$: the value of the best feasible point $x_{\widetilde{f}}$ found so far, and
- f_{min}: the minimal value of the lower bounds $\underline{x_{obj}}$ of the nodes $[x]$ to explore, i.e. the minimal $\underline{x_{obj}}$ in the list of open nodes.
  In other terms, in every box $[x]$, there is a guarantee that no feasible point exists with an objective function value lower than $\underline{x_{obj}}$.

The procedure `SelectBox` selects the next node to handle. If the criterion selects a box $[x]$ with a minimal lower bound estimate of the objective function ($\underline{x_{obj}}$), the B&B algorithm will implement the standard best-first search.

The selected box $[x]$ is then split into two sub-boxes $[y]$ and $[z]$ along one dimension (selected by any branching strategy). Both sub-boxes are then handled by the `Contract&Bound` procedure (see Algorithm 2).

A constraint $x_{obj} \leq \widetilde{f} - \epsilon_{obj}$ is first added to the problem for decreasing the upperbound of the objective function in the box. Imposing $\widetilde{f} - \epsilon_{obj}$ as a new upperbound (and not only $\widetilde{f}$) aims at finding a solution *significantly* better than the current best feasible point. The procedure then *contracts* the handled box without loss of feasible part [8]. In other words, some infeasible parts at the bounds of the domain are discarded by constraint programming (CP) [28,3,19] and convexification [25,17] algorithms. This contraction works on the extended box including the objective function variable x_{obj} and on

[1] We restrict the problem to inequality constraints because our `IbexOpt` software, in which the new strategies proposed have been developed, must perform a rigorous upperbounding in a feasible region with a nonempty interior. However, the node selection rules proposed in this paper can also apply to the more general optimization problem having equality constraints.

Algorithm `IntervalBranch&Bound` *(f, g, x, box, ϵ_{obj}, ϵ_{sol})*

$f_{min} \leftarrow -\infty$; $\widetilde{f} \leftarrow +\infty$;
$f_{min}^{sb} \leftarrow +\infty$ /* The lowest cost of the nodes having reached the ϵ_{sol} precision. */
$Boxes \leftarrow \{box\}$
while $Boxes \neq \emptyset$ **and** $\widetilde{f} - f_{min} > \epsilon_{obj}$ **and** $\frac{\widetilde{f}-f_{min}}{|f|} > \epsilon_{obj}$ **do**
  $[x] \leftarrow$ `SelectBox` ($Boxes$, criterion); $Boxes \leftarrow Boxes \setminus \{[x]\}$
  $([y],[z]) \leftarrow$ `Bisect` ($[x]$)
  $([y], x_{\widetilde{f}}, \widetilde{f}, Boxes) \leftarrow$ `Contract&Bound` ($[y]$, f, g, x, ϵ_{obj}, $x_{\widetilde{f}}$, $\widetilde{f}$, $Boxes$)
  $([z], x_{\widetilde{f}}, \widetilde{f}, Boxes) \leftarrow$ `Contract&Bound` ($[z]$, f, g, x, ϵ_{obj}, $x_{\widetilde{f}}$, $\widetilde{f}$, $Boxes$)
  $(f_{min}^{sb}, Boxes) \leftarrow$ `UpdateBoxes` ($[y]$,ϵ_{sol}, f_{min}^{sb},$Boxes$)
  $(f_{min}^{sb}, Boxes) \leftarrow$ `UpdateBoxes` ($[z]$,ϵ_{sol}, f_{min}^{sb}, $Boxes$)
  $f_{min} \leftarrow \min(f_{min}^{sb}, \min_{[x]\in Boxes} \underline{x_{obj}})$

Algorithm 1: Interval-based Branch and Bound

Algorithm `Contract&Bound` *($[x]$, f, g, x, ϵ_{obj}, $x_{\widetilde{f}}$, $\widetilde{f}$, $Boxes$)*

$g' \leftarrow g \cup \{x_{obj} \leq \widetilde{f} - \epsilon_{obj}\}$
$[x] \leftarrow$ `Contraction` ($[x]$, $g' \cup \{f(x) = x_{obj}\}$)
if $[x] \neq \emptyset$ **then**
  // Upperbounding:
  $(x_{\widetilde{f}}, \text{cost}) \leftarrow$ `FeasibleSearch` ($[x]$, f, g', ϵ_{obj})
  if $cost < \widetilde{f}$ **then**
    $\widetilde{f} \leftarrow$ cost
    $Boxes \leftarrow$ `FilterOpenNodes`($Boxes$, $\widetilde{f} - \epsilon_{obj}$)
return ($[x]$, $x_{\widetilde{f}}$, $\widetilde{f}$, $Boxes$)

Algorithm 2: The `Contract&Bound` procedure run at each node of the B&B algorithm

the associated constraint, so improving $\underline{x_{obj}}$ amounts to improving the lower bound of the objective function image (lowerbounding).

The last part of the procedure carries out upperbounding. `FeasibleSearch` calls one or several heuristics searching for a feasible point $x_{\widetilde{f}}$ that improves the best cost $\widetilde{f}$ found so far. If the upperbound of the objective value is improved, the `FilterOpenNodes` procedure performs a type of garbage collector on all the open nodes by removing from $Boxes$ all the nodes having $\underline{x_{obj}} > \widetilde{f} - \epsilon_{obj}$.

In Algorithm 1, after the calls to `Contract&Bound`, the last calls to `Update-Boxes` (see Algorithm 3) generally push the two sub-boxes in the set $Boxes$ of open nodes. However, if the size of a box reaches the precision ϵ_{sol}, the box will be no more studied (i.e., bisected) and only the minimal value f_{min}^{sb} of the objective function of these small discarded boxes is updated.

Note that if the search tree is traversed in best-first order, then an exponential memory may be required to store the nodes to handle.

Algorithm `UpdateBoxes` *([x], ϵ_{sol}, f^{sb}_{min}, Boxes)*
  if $[x] \neq \emptyset$ **then**
    if $w([x]) > \epsilon_{sol}$ **then**
      `Push` ([x], *Boxes*)
    else
      $f^{sb}_{min} \leftarrow min(f^{sb}_{min}, \underline{x_{obj}})$ /* or $\min(f^{sb}_{min}, \underline{[f]([x])})$ */
  return (f^{sb}_{min}, *Boxes*)

Algorithm 3: Routine for updating the list of open nodes or the minimal objective function value of small boxes no more studied.

Let us recall that $\underline{x_{obj}}$ is a lower bound of the objective function in the box $[x]$ (no feasible point below $\underline{x_{obj}}$) and $\overline{x_{obj}}$ is an upper bound of the minimum of the objective function in the box $[x]$. Remark that $\overline{x_{obj}}$ does *not* generally correspond to the objective value of a feasible point. *Interval arithmetic* [18] provides an easy way to compute $\underline{x_{obj}}$ and $\overline{x_{obj}}$. With interval arithmetic, we replace the standard mathematical operators in f by their interval counterpart to compute $[f]([x])$. Therefore, $\underline{[f]([x])}$ computes a lower bound $\underline{x_{obj}}$ while $\overline{[f]([x])}$ computes an upper bound $\overline{x_{obj}}$. In the next section, we compute better lower and upper bounds that also take into account the constraints, i.e. the feasible region.

3 New Selection Rule Using Upper and Lower Bounds

In optimization, the selection of the next node to expand is crucial for obtaining a good performance. The best node we can choose is such that it will improve the most the upperbound. Indeed, the upperbound improvement reduces *globally* the feasible space due to the constraint: $x_{obj} \leq \widetilde{f} - \epsilon_{obj}$. There exist two phases in a Branch and Bound:

1. a phase where the algorithm tries to find the optimal solution, and
2. a second phase where one has to prove that this solution is optimal, which requires one to expand all the remaining nodes.

Therefore the node selection matters only in the first phase.

We define in this section new strategies aggregating two criteria for selecting the box to be subdivided:

1. `LB`: The well known criterion used by best-first search and selecting the open node with the smallest $\underline{x_{obj}}$. This criterion is optimistic since we hope to find a solution with cost $\overline{f_{min}}$, in which case the search would end.
2. `UB`: This criterion selects the node having the smallest upperbound, i.e. the open node with the smallest $\overline{x_{obj}}$. Thus, if a feasible point is found in a sub-box, it will more likely improve the best cost found so far.

This first UB criterion is symmetric to the LB one. For every box, $\underline{x_{obj}}$ and $\overline{x_{obj}}$ are computed by the Contract&Bound procedure and label the node before storing it in the set of open boxes. Remember that a variable x_{obj} representing the value of the objective function has been introduced. In this case, $\underline{x_{obj}}$ and $\overline{x_{obj}}$ are simply the bounds of the interval $[x_{obj}]$ after contraction by Algorithm 2. Constraint programming techniques like 3BCID [27] can improve $[x_{obj}]$ by handling and discarding small slices at the bounds of $[x_{obj}]$ (shaving process).

Note that the contraction of the problem including x_{obj} produces generally an even better computation of $\overline{x_{obj}}$ than using a natural interval evaluation of the objective function (i.e., $\overline{x_{obj}} = ub$ computed by $[lb, ub] \leftarrow [f]([x])$). In our work, $\overline{x_{obj}}$ constitutes an upper bound on the minimal value of the objective value satisfying the constraints and lower than $\widetilde{f} - \epsilon_{obj}$. Indeed, since we have added the constraint $x_{obj} = f(x)$, the contraction on x due to the constraints $g(x) \leq 0$ are propagated on x_{obj}. Therefore, a slice at the bounds of $[x_{obj}]$ may be eliminated because it corresponds to a *non* feasible region.

We propose two ways to aggregate these two criteria.

- LB+UB
  This strategy selects the node $[x]$ with the smallest value of the sum $\underline{x_{obj}} + \overline{x_{obj}}$. This corresponds to a minimization of both criteria with the same weight, that is, a minimization of the middle of the interval of the objective function estimate in the box.
- LBvUB: alternating criteria
  In this second strategy, the next box to handle is chosen using *one* of the two criteria. A random choice is made by the SelectBox function at each node selection, with a probability UBProb of choosing UB.
  If UB (resp. LB) is chosen and several nodes have the same cost $\overline{x_{obj}}$ (resp. $\underline{x_{obj}}$), then we use the other criterion LB (resp. UB) to break ties.
  Experiments showed that the performance is not sensitive to a fine tuning of the UBProb parameter provided it remains between 0.2 and 0.8, so that the parameter has been fixed to 0.5. The experiments presented in Section 5 highlight the positive impact of this criterion on performance.

3.1 Improving the UB Criterion by Favoring Feasible Regions

The $\overline{x_{obj}}$ value of each open box was computed when handled by Contract&Bound (before being pushed into the heap of open nodes). Thanks to the modifications brought to Contract&Bound (see Algorithm 4), all $\overline{x_{obj}}$ values fall in four main categories. This explains how the UB criterion selects the next box to be subdivided. The UB value $\overline{x_{obj}}$ is:

1. lower than $\widetilde{f} - \epsilon_{obj}$ if the contraction procedure reduced $\overline{x_{obj}}$ in the box,
2. equal to $\widetilde{f} - \epsilon_{obj}$, if the box is a descendant of the box containing the current best feasible point $\widetilde{f}$,

3. equal to $\widetilde{f} - 0.9\,\epsilon_{obj}$ if the box was handled after the last update of $\widetilde{f}$,
4. greater than $\widetilde{f} - 0.9\,\epsilon_{obj}$ in the remaining case.

```
Algorithm Contract&Bound ([x], f, g, x, ε_obj, x_f̃, f̃)
    g' := g ∪ {x_obj ≤ f̃ − 0.9ε_obj}
    [x] ← Contraction ([x], g' ∪ {f(x) = x_obj})
    if [x] ≠ ∅ then
        /* Upperbounding: */
        (x_f̃, cost) ← FeasibleSearch ([x], f, g', ε_obj)
        if cost < f̃ then
            f̃ ← cost
            x_obj ← f̃ − ε_obj
            Boxes ← FilterOpenNodes(Boxes, f̃ − ε_obj)
    return ([x], x_f̃, f̃)
```

Algorithm 4: Modification of the `Contract&Bound` procedure for improving the `UB` criterion

As shown in the `Contract&Bound` pseudocode, this modification favors the boxes that are issued by a bisection of boxes where the current best feasible point was found.

The good experimental results obtained by the `LBvUB` strategy (see Section 5) suggest that it is important to invest both in intensification (`UB`) and in diversification (`LB`). In other words, the use of a second criterion allows the search to avoid the drawback of using one criterion alone, i.e. (for `LB`) choosing promising boxes with no feasible point and (for `UB`) going deeply in the search tree where only slightly better solutions will be found trapped inside a local minimum.

3.2 Implementation of the Set of Open Nodes

In our implementation, the set *Boxes* of open nodes was initially implemented by a heap data structure, i.e. a binary tree, ordered by the `LB` criterion, thus performing a best-first search. For the `LBvUB` strategy, several implementations have been tested to select the nodes according to both criteria (`LB` and `UB`). Several implementation choices have been tested.

One heap sorted by $\underline{x_{obj}}$

In this case, the node selection using `UB` comes at a linear cost in the number of open nodes. In practice, on the tested instances, the time spent on the heap management takes about 10% of the total time when the number of open nodes exceeds 50,000.

Thus, we designed a variant that still uses one heap and checks $|Boxes|$: If $|Boxes|$ exceeds 50,000, the `UBProb` probability is changed to 0.1; otherwise `UBProb` remains equal to 0.5. This variant leads to fewer calls to the $\overline{x_{obj}}$ minimization criterion.

Two heaps

Finally, we tried a cleaner implementation, with two heaps, one for `LB`, one for `UB`.[2] In addition, each element in one heap has a pointer to the corresponding node in the other heap. All the operations, such as additions or removals, must be performed twice, but are achieved in time $log_2(|Boxes|)$. Only the heap filtering process, achieved by the `FilterOpenNodes` procedure (launched by `Contract&Bound`) each time a better feasible point is found, remains in linear time.

We ran experiments to test the different policies. It appeared that the implementation using two heaps gave the best results. All the results of an `LBvXX` strategy shown in the experimental part use this implementation.

4 New Selection Rule Based on a "Feasible Diving" Procedure

We have designed a second node selection strategy that makes a tradeoff between exploitation and exploration. The interval B&B algorithm described in Algorithm 5 performs a best-first search that significantly differs from Algorithm 1. The node selection strategy is a variant of the `LB` criterion that selects a node $[x]$ with a smallest $\underline{x_{obj}}$. We just add other criteria to break ties:

1. the highest node in the tree is selected,
2. in case of equality on the two criteria ($\underline{x_{obj}}$ and then depth in the tree), one selects the node generated first.

At each node of this interval B&B search, we now call a probing procedure based on a greedy depth-first search to better intensify the search. This *Feasible Diving* procedure is described in Algorithm 6. From the selected node, `FeasibleDiving` builds a tree in depth-first search and keeps the more promising node at each iteration. The box in the node is bisected along one dimension (chosen by the same branching strategy as in the B&B algorithm, e.g. the *SmearSum relative* branching strategy [26]) and the two sub-boxes are handled by the `Contract&Bound` procedure (see Algorithm 2). The sub-box of $[x]$ with the smallest x_{obj} is handled in the next step while the other sub-box is pushed into the global heap *Boxes* implementing the list of open nodes. Contrarily to a standard depth-first search, `FeasibleDiving` does not perform any backtracking. That is why we call it a greedy or diving algorithm.

The `FeasibleDiving` procedure tries to dive inside the feasible region (or tries to prove that no such a feasible region exists) thanks to the `Contraction`

[2] The heaps contain references (pointers) to the actual nodes.

Algorithm `IntervalBranch&BoundBis` *(f, g, x, box, ϵ_{obj}, ϵ_{sol})*

$f_{min} \leftarrow -\infty$; $\widetilde{f} \leftarrow +\infty$
$f^{sb}_{min} \leftarrow +\infty$ /* The lowest cost of the nodes having reached the ϵ_{sol} precision. */
$Boxes \leftarrow \{box\}$
while $Boxes \neq \emptyset$ **and** $\widetilde{f} - f_{min} > \epsilon_{obj}$ **and** $\frac{\widetilde{f}-f_{min}}{|f|} > \epsilon_{obj}$ **do**
  $[x] \leftarrow$ `SelectBox` ($Boxes$, criterion); $Boxes \leftarrow Boxes \setminus \{[x]\}$
  $([x], x_{\widetilde{f}}, \widetilde{f}, f^{sb}_{min}, Boxes) \leftarrow$ `FeasibleDiving` ($[x]$, g, f, x, ϵ_{sol}, $x_{\widetilde{f}}$, $\widetilde{f}$, f^{sb}_{min}, $Boxes$)
  if $[x] \neq \emptyset$ **and** $w([x]) < \epsilon_{sol}$ **then** $f^{sb}_{min} \leftarrow min(f^{sb}_{min}, \underline{x_{obj}})$
  $f_{min} \leftarrow \min(f^{sb}_{min}, \min_{[x]\in Boxes} \underline{x_{obj}})$

Algorithm 5: Interval-based Branch and Bound calling Feasible Diving at each node

Algorithm `FeasibleDiving` *($[x]$, g, f, x, ϵ_{sol}, $x_{\widetilde{f}}$, $\widetilde{f}$, f^{sb}_{min}, $Boxes$)*

while $[x] \neq \emptyset$ **and** $w([x]) > \epsilon_{sol}$ **do**
  $([y], [z]) \leftarrow$ `Bisect` ($[x]$)
  $([y], x_{\widetilde{f}}, \widetilde{f}) \leftarrow$ `Contract&Bound` ($[y]$, f, g, x, ϵ_{obj}, $x_{\widetilde{f}}$, $\widetilde{f}$)
  $([z], x_{\widetilde{f}}, \widetilde{f}) \leftarrow$ `Contract&Bound` ($[z]$, f, g, x, ϵ_{obj}, $x_{\widetilde{f}}$, $\widetilde{f}$)
  $([x]_{best}, [x]_{worst}) \leftarrow$ `Sort` ($[y]$, $[z]$, *criterion*) /* $\underline{[x]_{best}} < \underline{[x]_{worst}}$ */
  $(f^{sb}_{min}, Boxes) \leftarrow$ `UpdateBoxes` ($[x]_{worst}$, ϵ_{sol}, f^{sb}_{min}, $Boxes$)
  $[x] \leftarrow [x]_{best}$
return ($[x]$, $x_{\widetilde{f}}$, $\widetilde{f}$, f^{sb}_{min}, $Boxes$)

Algorithm 6: The `Feasible Diving` procedure run at each node of a best-first interval B&B algorithm.

and `FeasibleSearch` (upperbounding) procedures performed at each iteration. In other terms, if we follow the $[x]_{best}$ node from the root of the tree built by `FeasibleDiving`, slices of $[x]_{best}$ with no feasible points are discarded by the `Contraction` procedure (potentially leading to en empty box guaranteed to contain no feasible point) and feasible points are searched for inside $[x]_{best}$ at each node. That is why we call this procedure Feasible Diving.

Overall, we should highlight that this B&B alternates a diversification phase that selects the nodes in best-first order and an intensification phase, performed by `FeasibleDiving`, where the nodes are selected by a greedy diving.

It is mentioned in [4] and [5] that a procedure similar to `FeasibleDiving` has been proposed for mixed integer linear programming solvers. While not detailed in the literature, this *Probed Diving* procedure seems to be used by the IBM Ilog CPLEX MIP solver with some success.

5 Numerical Experiments

We have run experiments on a sample of constrained global optimization instances. Section 5.1 reports the implementation and the protocol used. Section 5.2 justifies that completely forgetting the `LB` criterion leads to poor results in practice. Section 5.3 compares node selection rules randomly choosing at each node between the `LB` criterion and another one. Section 5.4 compares `FeasibleDiving` and `LBvUB` to the standard best-first search `LB` strategy.

5.1 Experimental Protocol and Implementation

All the variants of our `IbexOpt` global optimization solver [26] have been implemented using the Interval-Based EXplorer (in short: `Ibex`) free `C++` library [7, 8]. The main ingredients in `IbexOpt` include:

- For contraction and lower bounding:
  - Interval constraint programming contractors such as the well-known HC4 constraint propagation algorithm [3,15] and the more recent ACID contractor [19].
  - Interval-based polyhedral relaxation algorithms based on `X-Taylor` [1] and `ART` [16,21] (affine arithmetic relaxation technique).
- For the upper bounding phase:
  An original approach extracts an *inner region* (box or polytope) in the feasible space using the `inHC4` and `in-XTaylor` heuristics [2]. Inside an inner region, one candidate point is picked by different techniques (monotonicity analysis of the objective function, random selection – for non monotonic dimensions, minimization of an affine approximation of the objective function) and the upperbound is updated accordingly.

Note that a Lagrangian method and a convexity analysis are not implemented yet in the current version of `IbexOpt` due to the difficulty of rendering these methods rigorous.

We selected all the instances from the series 1 and 2 of the Coconut constrained global optimization benchmark [23] that:

- are solved between 1 and 3600 seconds by one competitor (objective function precision ϵ_{obj} = `1e-6`),
- have between 6 and 50 variables,
- are solved by `IbexOpt` using the best branching strategy among *Smear sum absolute* (ssa) or *relative* (ssr), *Smear max* (sm), *Largest interval first* (lf) and *Round robin* (rr).

This gives the 82 instances listed in Annex A (Table 3) with the chosen branching strategy, the same for all methods.

Note that the equations $h(x) = 0$ in these benchmarks have been relaxed by inequalities $-10^{-8} \leq h(x) \leq +10^{-8}$ because `IbexOpt` needs to achieve upperbounding in nonempty feasible regions [2].

All the tests have been carried out using the recent version 2.1.10 of `Ibex` on a `X86-64` processor under `Linux Ubuntu`.

5.2 Bad Performance Without the LB Criterion

This section highlights that the only promising strategies, i.e. that give better performance results than `LB` (best-first search), still run `LB` at certain nodes of the search tree.

Indeed, `LB` can solve all the 82 instances within the timeout of one hour. `LBvUB` can also solve the 82 instances and gives better results generally. Details are provided in the next sections.

However, the `LB+UB` strategy could not solve 4 instances in less than one hour. For the remaining 78 instances, `LB+UB` gave good results compared to `LB`, with a total time gain ratio (i.e., $time(LB)/time(LB+UB)$) of 1.33 and an average time gain ratio of 1.58.[3] The same phenomenon appeared with the pure C_3, C_5 and C_7 strategies where a few instances could not be solved before the timeout.

These results make us believe that it is useful to sometimes choose the node minimizing the `LB` ($\underline{x_{obj}}$) criterion in order to diversify the search.

5.3 Comparing Variants of `LBvXX` Selection Rules

We denote by `LBvXX` a node selection rule that randomly chooses between `LB` and another criterion `XX` at each node of the B&B.

All the experiments test a strategy `LBvXX` (with a probability 0.5 of choosing one of both criteria for the next node selection). The first criterion is `LB` and the `XX` criterion can be:

- UB_0: `UB` with no bias for the descendants (sub-boxes in the B&B tree) of the node where the latest $\widetilde{f}$ was found.
- UB_1: `UB` with the modification of the `Contract&Bound` procedure described in Algorithm 4 that favors the descendants of the node where the latest $\widetilde{f}$ was found.
- FUB: `UB` with an even stronger bias for the descendants of the nodes where a feasible point was found. One such descendant with the smallest $\overline{x_{obj}}$ is selected, if such one node exists; otherwise (i.e., if all the subtrees of nodes where feasible points were found are entirely explored) another node with the smallest $\overline{x_{obj}}$.
- MID: evaluation of the objective function in the middle of the box, thus forgetting the feasible region.

[3] A total time gain ratio means that the total CPU time required by `LB` for solving the 78 instances is divided by the total CPU time required by `LB+UB` for solving the same instances. An average time gain ratio means that a division between the respective CPU times is achieved on each instance before computing an average of the 78 results.

– C_3, C_5 or C_7: the criteria described in [14] and mentioned in introduction. We modified these criteria for improving their performance. The $[lb, ub]$ computation is not obtained by interval evaluation but by contracting $[x_{obj}]$, like for our criteria.

Table 1 shows the synthetic comparison of all these variants. We report the time and number of nodes gain ratios comparing each strategy s and the reference strategy LB. We take into account the CPU time and the number of nodes obtained using a strategy s on an instance i (these measurements are in fact an averaged value over 10 trials obtained with different seeds, due to the random choices made by `IbexOpt`):

$$timeRatio(i, s) = time(LB, i)/time(s, i)$$

We have also added a more sophisticated normalized measure of the gains. Each instance gets a normalized time ratio between 0 and 1 comparing a strategy s and the reference strategy LB. This ratio gives a value between 0 and 0.5 if LB is better, and a value between 0.5 and 1 if s provides better results. We then compute an aggregate normalized gain ratio over the instances. Therefore a value of this normalized ratio greater than 1 denotes a gain, a value smaller than 1 denotes a loss.

– $NormalizedTimeRatio = time(LB, i)/(time(s, i) + time(LB, i))$
– $AverageNormalizedTimeRatio(s)$ is an average value of the $timeRatio(i, s)$ for i.
– aggregate normalized gain ratio (reported value for s): $AverageNormalizedTimeRatio(s)/(1-AverageNormalizedTimeRatio(s))$

	UB_0	UB_1	FUB	MID	C_3	C_5	C_7
Total Time gain	1.33	1.47	1.46	1.22	1.31	0.78	1.18
Average Time gain on the 82 instances	1.50	1.45	1.41	1.35	1.33	0.87	1.07
Average Time gain on instances > 10 sec	2.03	1.92	1.85	1.67	1.65	0.88	1.24
Norm. Time gain on the 82 instances	1.17	1.17	1.15	1.10	1.16	0.82	0.98
Norm. Time gain on instances > 10 sec	1.27	1.34	1.23	1.14	1.32	0.80	1.08
Total Node gain	1.49	1.63	1.64	0.88	1.52	0.83	1.23
Average Node gain	1.78	1.69	1.66	1.60	1.60	1.01	1.12

Table 1 Comparison of different LBvXX strategies selecting a node using LB or "XX" criterion with a probability 0.5.

We observe that only `LBvC5` gives results significantly worse than `LB`. All the other `LBvXX` variants give good results. `LBvC3` is the best variant among Markot's et al. criteria. $\texttt{LBvUB}_1$ gives the best results in total time and normalized gain. It is the version of `LBvUB` tested below.

5.4 Detailed Comparison Between LB, LBvUB and FD

Figure 1 shows a diagram comparing LB, LBvUB (i.e., the best variant $\texttt{LBvUB}_1$) and FD. The scatterplots highlight that:

- LBvUB dominates LB on most instances,
- FD significantly dominates LB,
- FD generally dominates LBvUB.

These comparisons are shown in a more synthetic way in Table 2.

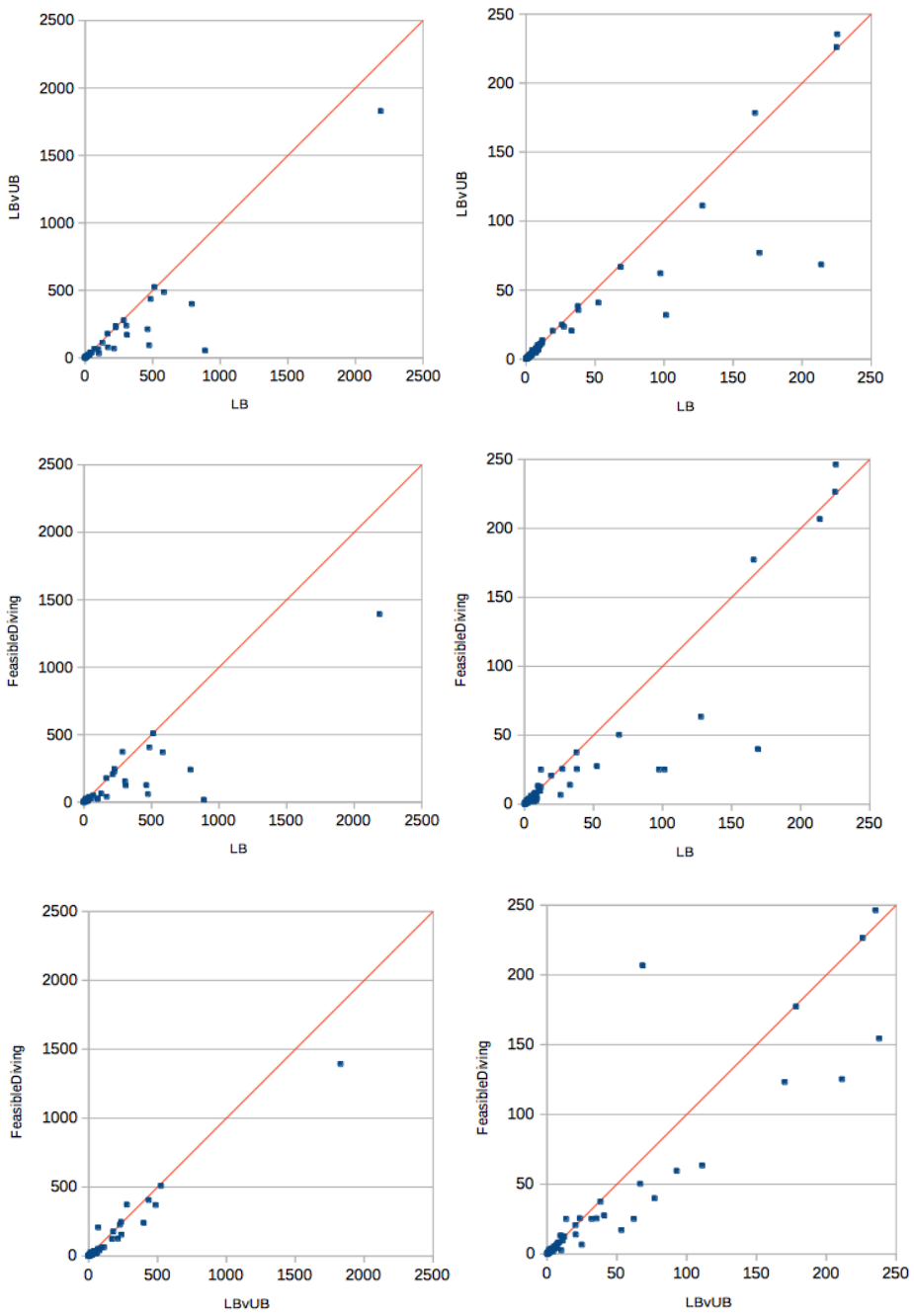
Gain/loss ranges	gain > 5	gain $[2, 5]$	gain $[1.2, 2]$	gain $[1.05, 1.2]$	equiv. $[0.95, 1.05]$	loss $[0.8, 0.95]$	loss $[0.5, 0.8]$	loss $[0.2, 0.5]$	loss <0.2
LBvUB vs LB	2	6	16	12	24	19	3	0	0
FD vs LB	2	19	15	7	15	12	11	1	0
FD vs LBvUB	0	6	30	8	18	10	9	1	0

Table 2 Detailed gains/losses of a given strategy s_1 w.r.t. another strategy s_2. The gains are expressed by $\frac{time(s_2)}{time(s_1)}$. The entries report the number of instances showing a gain in a given range.

Remarks

- FeasibleDiving also obtained very good results on other problems.
  It is the case on an instance called concon in the Coconut benchmark when the ssr by-default branching strategy is used.[4] FeasibleDiving can solve the instance in 10 seconds on 8 of the 10 trials while all the other tested variants always reach a timeout of 1000 seconds.
  The Coconut instance called chem could be added to the list of 82 instances. chem can be solved within one hour only when the B&B uses the round robin branching strategy. With this branching strategy, FeasibleDiving can solve chem to optimality in 2790 seconds, whereas LB requires 4988 seconds and LBvUB requires 4884 seconds.
- Remember that the B&B algorithm calls an upperbounding procedure at each node explored by FeasibleDiving. Indeed, each iteration of FeasibleDiving calls Contract&Bound that itself calls the Contraction and FeasibleSearch procedures. It is possible to very often call FeasibleSearch in IbexOpt because it corresponds to two relatively cheap heuristics (called InHC4 and In-XTaylor in [2]). This could explain the good results obtained using FeasibleDiving.

[4] These results should be seen in perspective since the instance can be solved to optimility in a fraction of a second using a largest-first branching strategy.



(e) Scatterplots for all the instances (f) Zoom on instances solvable within 250 sec

Fig. 1 Pairwise comparison between `LB`, `LBvUB` and `FeasibleDiving`. The coordinates of each point represents the CPU time (in second) required by competitors.

6 Conclusion

The node selection policy is a promising line of research to improve performance of interval B&B algorithms. In this article we have proposed two main new strategies that obtained good experimental results on difficult instances.

Both approaches are based on a best-first interval B&B algorithm. The first node selection strategy, called `LBvUB`, switches between the selection of a node with a smallest lower bound (`LB`) and with a smallest upper bound (`UB`). The `UB` criterion is biased to slightly favor descendants of nodes where feasible points have been found. The lower and upper bounds are made accurate by box contraction operations. Note that the $\texttt{LBvC}_3$ node selection rule randomly choosing between `LB` and a version of the C_3 criterion proposed by Markot et al. also leads to promising results.

The second node selection strategy greedily dives into potential feasible regions at each node of a best-first search using a `FeasibleDiving` procedure. Each node handled by this procedure is contracted and explored for finding a feasible point inside.

Both node selection strategies outperform the standard best-first interval B&B algorithm, and `FeasibleDiving` generally obtains the best performance results. We think that this node selection strategy benefits from a synergy with the cheap upperbounding procedures available in `IbexOpt` and used by the `FeasibleDiving` procedure.

References

1. Araya, I., Trombettoni, G., Neveu, B.: A Contractor Based on Convex Interval Taylor. In: Proc. CPAIOR, *LNCS*, vol. 7298, pp. 1–16. Springer (2012)
2. Araya, I., Trombettoni, G., Neveu, B., Chabert, G.: Upper Bounding in Inner Regions for Global Optimization under Inequality Constraints. J. Global Optimization (JOGO) **60**(2), 145–164 (2014)
3. Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.F.: Revising Hull and Box Consistency. In: Proc. ICLP, *LNCS*, vol. 5649, pp. 230–244. Springer (1999)
4. Bixby, R., Rothberg, E.: Progress in Computational Mixed Integer Programming – A Look Back from the Other Side of the Tipping Point. Annals of Operations Research **149**, 37–41 (2007)
5. Bonami, P., Kilink, M., Linderoth, J.: Algorithms and Software for Convex Mixed Integer Nonlinear Programs. Tech. Rep. 1664, U. Wisconsin (2009)
6. Casado, L., Martinez, J., Garcia, I.: Experiments with a New Selection Criterion in a Fast Interval Optimization Algorithm. Journal of Global Optimization **19**, 247–264 (2001)
7. Chabert, G.: Interval-Based EXplorer (2015). `www.ibex-lib.org`
8. Chabert, G., Jaulin, L.: Contractor Programming. Artificial Intelligence **173**, 1079–1100 (2009)
9. Csendes, T.: New Subinterval Selection Criteria for Interval Global Optimization. Journal of Global Optimization **19**, 307–327 (2001)
10. Csendes, T., Ratz, D.: Subdivision Direction Selection in Interval Methods for Global Optimization. SIAM Journal on Numerical Analysis **34**(3) (1997)
11. Felner, A., Kraus, S., Korf, R.E.: KBFS: K-Best-First Search. Annals of Mathematics and Artificial Intelligence **39** (2003)
12. Kearfott, R., Novoa III, M.: INTBIS, a Portable Interval Newton/Bisection Package. ACM Trans. on Mathematical Software **16**(2), 152–157 (1990)

13. Kocsis, L., Szepesvari, C.: Bandit based Monte-Carlo Planning. In: Proc. ECML, *LNCS*, vol. 4212, pp. 282–293. Springer (2006)
14. Markot, M., Fernandez, J., Casado, L., Csendes, T.: New Interval Methods for Constrained Global Optimization. Mathematical Programming **106**, 287–318 (2006)
15. Messine, F.: Méthodes d'optimisation globale basées sur l'analyse d'intervalle pour la résolution des problèmes avec contraintes. Ph.D. thesis, LIMA-IRIT-ENSEEIHT-INPT, Toulouse (1997)
16. Messine, F., Laganouelle, J.L.: Enclosure Methods for Multivariate Differentiable Functions and Application to Global Optimization. Journal of Universal Computer Science **4**(6), 589–603 (1998)
17. Misener, R., Floudas, C.: ANTIGONE: Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations. J. Global Optimization (JOGO) **59**(2–3), 503–526 (2014)
18. Moore, R.E.: Interval Analysis. Prentice-Hall (1966)
19. Neveu, B., Trombettoni, G., Araya, I.: Adaptive Constructive Interval Disjunction: Algorithms and Experiments. Constraints Journal **DOI: 10.1007/s10601-015-9180-3**, Accepted for publication (2015)
20. Ninin, J., Messine, F.: A Metaheuristic Methodology Based on the Limitation of the Memory of Interval Branch and Bound Algorithms. Journal of Global Optimization **50**, 629–644 (2011)
21. Ninin, J., Messine, F., Hansen, P.: A Reliable Affine Relaxation Method for Global Optimization. 4OR-Quaterly Journal of Operations Research (2014). Accepted to publication. DOI: 10.1007/s10288-014-0269-0
22. Sabharwal, A., Samulowitz, H., Reddy, C.: Guiding Combinatorial Optimization with UCT. In: Proc. CPAIOR, *LNCS*, vol. 7298, pp. 356–361. Springer (2012)
23. Shcherbina, O., Neumaier, A., Sam-Haroud, D., Vu, X.H., Nguyen, T.V.: Benchmarking Global Optimization and Constraint Satisfaction Codes. In: COCOS, Workshop on Global Constraint Optimization and Constraint Satisfaction (2002). `www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html`
24. Tawarmalani, M., Sahinidis, N.V.: Global Optimization of Mixed-Integer Nonlinear Programs: A Theoretical and Computational Study. Mathematical Programming **99**(3), 563–591 (2004)
25. Tawarmalani, M., Sahinidis, N.V.: A Polyhedral Branch-and-Cut Approach to Global Optimization. Mathematical Programming **103**(2), 225–249 (2005)
26. Trombettoni, G., Araya, I., Neveu, B., Chabert, G.: Inner Regions and Interval Linearizations for Global Optimization. In: Proc. AAAI, pp. 99–104 (2011)
27. Trombettoni, G., Chabert, G.: Constructive Interval Disjunction. In: Proc. CP, *LNCS*, vol. 4741, pp. 635–650. Springer (2007)
28. Van Hentenryck, P., Michel, L., Deville, Y.: Numerica : A Modeling Language for Global Optimization. MIT Press (1997)

A List of the 82 constrained global optimization instances tested

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		

Table 3 Benchmark tested. The systems have been selected from the Coconut benchmark library (series 1 and 2) according to a protocol described in Section 5. The best branching strategy has been chosen for each system, the same for all methods, among: largest interval first (lf), round robin (rr), Smear max (sm), Smear sum absolute (ssa) (see [12]) and Smear sum relative (ssr) (see [26]).