Scientific Computing and Validated Numerics
Alefeld, Frommer, Lang (Eds.)
pp. 221–227

ISBN 3-05-501737-4

On Branching Rules in Second-Order Branch-and-Bound Methods for Global Optimization

Dietmar Ratz

Abstract

This paper investigates different branching rules, i.e. rules for selecting the subdivision direction, in interval branch-and-bound algorithms for global optimization. Earlier studies ([2],[9]) dealt with the subdivision direction selection in methods which do not use second-order information about the objective function. The investigated model algorithm (similar to that in [3]) now uses the enclosure of the Hessian matrix to incorporate a concavity test for box-discarding and an interval Newton Gauss-Seidel step to reduce the widths of the boxes resulting from the underlying generalized bisection method.

Four different branching rules are investigated, and a wide spectrum of test problems is used for numerical tests. The results indicate that there are substantial differences between the rules with respect to the performance of the model algorithm. This is clarified by comparing the required CPU times, the numbers of function and derivative evaluations, and the necessary amounts of storage space.

0 Introduction

Let $f : D \to I\!R$ be a twice continuously differentiable function, and let $D \supseteq [x] \in I I\!R^n$. We address the problem of finding all points x^* in the interval vector $[x]$ such that

$$f(x^*) = \min_{x \in [x]} f(x).$$

We are interested in both the global minimizers x^* and the minimum value $f^* = f(x^*)$.

We use the branch-and-bound method described in [3] and [8] with several modifications. Our method starts from an initial box $[x] \in I I\!R^n$, subdivides $[x]$, stores the subboxes in a list L, and discards subintervals which are guaranteed not to contain a global minimizer, until the desired accuracy of the intervals in the list is achieved. The tests we use to discard or to prune pending subboxes are cut-off test, monotonicity test, concavity test, and interval Newton Gauss-Seidel step. For details on these tests and on the method itself, see [3].

1 Global Optimization Algorithm

In the following, we give a simplified algorithmic description and an overview of our global optimization method. We use the notations from [3].

Algorithm 1: GlobalOptimize $(f, [x], \varepsilon, L_{\text{res}}, [f^*])$

1. $\widetilde{f} := \overline{f_\diamond}(m([x]))$; $[y] := [x]$; $L := \{\,\}$; $L_{\text{res}} := \{\,\}$;
2. **repeat**
   (a) FindOptmlComponents $([y], k_1, k_2)$; Branch $([y], k_1, k_2, [U]_1, [U]_2, [U]_3, [U]_4)$;
   (b) **for** $i := 1$ **to** 4 **do**
      i. **if** $\widetilde{f} < \underline{f}([U]_i)$ **then next**$_i$;
      ii. **if** MonotonicityTest $(\nabla f([U]_i))$ **then next**$_i$;
      iii. **if** ConcavityTest $(\nabla^2 f([U]_i))$ **then next**$_i$;
      iv. IntervalNewtonGaussSeidelStep $(f, [U]_i, \nabla^2 f([U]_i), [V], p)$;
      v. **for** $j := 1$ **to** p **do if** $\widetilde{f} \geq \underline{f}([V]_j)$ **then** $L := L \uplus ([V]_j, \underline{f_V})$;
   (c) **while** $(L \neq \{\,\})$ **do**
      i. $([y], \underline{f_y})$:= PopHead (L);
      ii. $\widetilde{f} := \min\{\widetilde{f}, \overline{f_\diamond}(m([y]))\}$; CutOffTest $(L, \widetilde{f})$;
      iii. **if** Accept $(f, [y], \varepsilon)$ **then** $L_{\text{res}} := L_{\text{res}} \uplus ([y], \underline{f_y})$ **else goto** 2(a);

   until $(L = \{\,\})$;
3. $([y], \underline{f_y})$:= Head (L_{res}); $[f^*] := [\underline{f_y}, \widetilde{f}]$; **return** $L_{\text{res}}, [f^*]$;

Algorithm 1 first computes an upper bound $\widetilde{f}$ for the global minimum value and initializes L and L_{res}. The main iteration (Step 2) starts with a multisection of $[y]$. Then we apply a range check, the monotonicity test, the concavity test, and the interval Newton step to the multisected boxes $[U_1]$, $[U_2]$, $[U_3]$, and $[U_4]$. The interval Newton step results in p boxes, to which we apply a range check. If the actual box $[V]_j$ is still a candidate for a minimizer, we store it in L in Step 2(b)v. Note that the boxes are stored as pairs $([y], \underline{f_y})$ in list L sorted in *nondecreasing* order with respect to the lower bounds $\underline{f_y} = \underline{f([y])}$ and in *decreasing* order with respect to the ages of the boxes in L (cf. [8]).

In Step 2(c), we remove the first element from the list L, i.e. the element of L with the smallest $\underline{f_y}$ value, and we perform the cut-off test. Then, if the desired accuracy is achieved for $[y]$, we store $[y]$ in the result list L_{res}. Otherwise, we go to the branching step. When the iteration stops because the pending list L is empty, we compute a final enclosure $[f^*]$ for the global minimum value and return L_{res} and $[f^*]$.

The method can be improved by incorporating an approximate local search procedure to try to decrease the value $\widetilde{f}$. See [5] for the description of such local search procedures. For our studies in this paper, we do not apply any local method. We also do not apply any boundary treating, so we assume that all x^* lie in the interior of $[x]$.

2 Branching Rules

In Algorithm 1, Step 2(a), different branching rules can be applied to determine "optimal" components for subdividing the current box $[y]$ (cf. [2] and [9]). Each of these rules selects directions k_1 and k_2 with $D(k_1) \geq D(k_2) \geq D(i)$ for all $i = 1, \ldots, n$ and $i \notin \{k_1, k_2\}$, where $D(i)$ is fixed by the given rule.

For the current study, we investigate four rules (we leave out Rule D from [2] and [9]):

Rule A: $D(i) := d([y]_i)$

Rule B: $D(i) := d(g_i([y])) \cdot d([y]_i)$ (cf. [5])

Rule C: $D(i) := d\Big(g_i([y]) \cdot ([y]_i - c_i)\Big)$ (cf. [7])

Rule E: $D(i) := d\Big(([y]_i - c_i) \cdot \Big(g_i(c) + \dfrac{1}{2}\sum_{j=1}^{n}(H_{ij}([y]) \cdot ([y]_i - c_i))\Big)\Big).$

Here, $g = \nabla f$, $H = \nabla^2 f$, and $c = m([y])$, where $m([y])$ and $d([y])$ denote the midpoint and the diameter (width) of $[y]$, respectively.

Similar to Rule C (cf. [7]), the underlying idea of the new Rule E is to minimize

$$\begin{aligned} d(f([y])) &= d(f([y]) - f(c)) \\ &\approx d\Big(([y]-c)^T \cdot \Big(\nabla f(c) + \frac{1}{2}\nabla^2 f([y]) \cdot ([y]-c)\Big)\Big) \\ &= d\sum_{i=1}^{n}\Big(([y]_i - c_i) \cdot \Big(\frac{\partial f}{\partial x_i}([y]) + \frac{1}{2}\sum_{j=1}^{n}\frac{\partial^2 f([y])}{\partial x_i \partial x_j} \cdot ([y]_j - c_j)\Big)\Big). \end{aligned}$$

3 Interval Newton Gauss-Seidel Step

In our global optimization method, we apply one step of the extended interval Newton Gauss-Seidel method (cf. [1]) to the nonlinear system $\nabla f(y) = 0$ with $y \in [y]$. The subbox $[y]$ is a candidate box for enclosing a minimizer x^*, which we have assumed must satisfy $\nabla f(x^*) = 0$. One step of the extended interval Newton Gauss-Seidel method shall improve the enclosure $[y]$ by formally solving the system $g = [H] \cdot (c - y)$, where $c = m([y])$, $g = \nabla f(c)$, and $[H] = \nabla^2 f([y])$. This method works better if we first apply a *preconditioning*, by using a special matrix $R \in I\!R^{n\times n}$ for computing $b := R \cdot g$ and $[A] := R \cdot [H]$, and consider then the system $b = [A] \cdot (c - y)$. Then, we compute $N'_{\mathrm{GS}}([y])$ according to

$$\begin{aligned} [z] &:= [y] \\ [z]_i &:= \Big(c_i - \Big(b_i + \sum_{\substack{j=1 \\ j\neq i}}^{n}[A]_{ij} \cdot ([z]_j - c_j)\Big) \Big/ [A]_{ii}\Big) \cap [z]_i, \qquad i = 1, \ldots, n \\ N'_{\mathrm{GS}}([y]) &:= [z] \end{aligned}$$

If $0 \in [A]_{ii}$ for some i, extended interval arithmetic (see [3] for details) is applied. In this case, a gap can be produced in the corresponding components $[z]_i$ of $[z]$. Therefore, the interval Gauss-Seidel step may result in the union of several boxes $N'_{\mathrm{GS}}([y]) = [V]_1 \cup \ldots \cup [V]_p$, where $[V]_i \in I\!\!R^n$, $i = 1, \ldots, p$, that is $[V] \in I\!\!R^{p \times n}$.

In a practical realization of the interval Newton Gauss-Seidel method, it is not necessary to compute the $[y]_i$ in fixed order $i = 1, \ldots, n$. A well-known strategy is the Hansen/Greenberg realization [4], which first performs the single component steps of the Gauss-Seidel step for all i with $0 \notin [A]_{ii}$ and then for the remaining indices with $0 \in [A]_{ii}$ by using extended interval arithmetic.

If $0 \in [A]_{ii}$ for several components i, then the extended interval divisions in the interval Newton Gauss-Seidel method possibly produces several gaps in the actual box $[y]$. So we must split the result $N'_{\mathrm{GS}}([y])$ into two or more boxes. In this case, different splitting techniques may be applied resulting in different values for $[V]$ and p.

We give two examples, which are used in the current study:

$p \leq 2$ — *Compute all possible gaps in* $[y]$, *and finally use only the largest gap to split* $[y]$. This technique is known from Hansen/Greenberg [4], and the Newton step results in at most 2 boxes, thus $N'_{\mathrm{GS}}([x]) = [V]_1 \cup [V]_2$.

$p \leq n + 1$ — *Compute every gap, and use it* immediately *to split* $[y]$ *in a special way.* For this special splitting technique introduced in [7] the Newton step results in at most $n + 1$ boxes, thus $N'_{\mathrm{GS}}([x]) = [V]_1 \cup \ldots \cup [V]_{n+1}$.

In our special technique with $p \leq n+1$, we use each gap to store one part of the actual box $[y]$ by using one part of the component $[y]_i$ and the other part of $[y]_i$ to update $[y]$ before continuing with the next component step of the interval Gauss-Seidel method. That is, we perform one component step according to the scheme:

1. Compute $[y]_i = [v]_i \cup [w]_i$.
2. If $[v]_i = [w]_i = \emptyset$, then stop {no solution in $[y]$}.
3. If $[w]_i \neq \emptyset$, then set $[y]_i := [w]_i$ and store $[y]$.
4. Set $[y]_i := [v]_i$ and continue with next i.

In many cases, we get $\Big(b_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} [A]_{ij} \cdot ([y]_j - c_j)\Big) \Big/ [A]_{ii} = (-\infty, \infty)$ and no gap occurs, so $[y]_i := [y]_i \cap (-\infty, \infty)$ remains unchanged. In these cases, we introduce "gaps" of width zero by splitting $[y]_i = [v]_i \cup [w]_i$ with $[v]_i := [\underline{y}_i, m([y]_i)]$ and $[w]_i := [m([y]_i), \overline{y}_i]$, that is we do a bisection.

4 Sorted Interval Newton Gauss-Seidel Step

We investigate the branching rules applied in the main Algorithm 1 also in connection with the interval Gauss-Seidel step. We use these rules to compute a sorting vector $s = (s_1, s_2, \ldots, s_n)$ with $s_i \in \{1, \ldots, n\}$ and $s_i \neq s_j$ for $i \neq j$, which satisfies

$D(s_i) \geq D(s_{i+1})$, $i = 1, \ldots, n-1$ for the corresponding direction selection rule $D(\ldots)$. Then, we perform the *sorted* interval Newton Gauss-Seidel step according to

$$[z] := [y]$$

$$[z]_{s_i} := \left(c_{s_i} - \left(b_{s_i} + \sum_{\substack{j=1 \\ j \neq s_i}}^{n} [A]_{s_i j} \cdot ([z]_j - c_j)\right) \Big/ [A]_{s_i s_i}\right) \cap [z]_{s_i}, \qquad i = 1, \ldots, n$$

$$N'_{GS}([y]) := [z]$$

incorporating the Hansen/Greenberg realization and different splitting techniques.

5 Numerical Experiences

For testing we used the two groups of standard test functions and some new functions (see [5] and [9] for details on the functions). We carried out the numerical tests on an HP 9000/730 equipped with PASCAL–XSC [6] using the basic toolbox modules for automatic differentiation and extended interval arithmetic [3].

In our test suite we compared the methods with branching rules A, B, C, and E combined with the usual splitting technique ($p \leq 2$) and with the special splitting technique ($p \leq n+1$, "0-width-gaps"). The complete results for all test functions together with the source code of our test program can be obtained by anonymous ftp from `iamk4515.mathematik.uni-karlsruhe.de (129.13.129.15)` in directory `pub/documents/ratz/scan.95`.

We list the results for four test functions in the following. Important columns are the runtime (in STUs), the storage space or maximum list length (LL) and the Eeff_1 and Eeff_2 values. The latter combine the three values for the number of function (FE), gradient (GE), and Hessian (HE) evaluation to single values approximating the total evaluation effort in terms of objective function evaluations by

$$\text{Eeff}_1 = \text{FE} + n \cdot \text{GE} + \frac{n \cdot (n+1)}{2} \cdot \text{HE} \quad \text{and} \quad \text{Eeff}_2 = \text{FE} + \min\{4, n\} \cdot \text{GE} + n \cdot \text{HE}$$

(with respect to forward (Eeff_1) and backward (Eeff_2) mode of automatic differentiation, see [8] for details).

Shekel10 ($n = 4$)

$p \leq$	Rule	STUs	FE	GE	HE	Eeff_1	Eeff_2	LL
2	A	2.09	132	106	42	976	724	17
	B	2.12	133	108	43	995	737	17
	C	1.68	112	86	32	776	584	15
	E	1.68	112	86	32	776	584	15
$n+1$	A	1.45	144	62	22	612	480	33
	B	1.71	169	70	26	709	553	39
	C	1.31	129	56	19	543	429	31
	E	1.33	129	56	19	543	429	31

Hartman6 ($n = 6$)

$p \leq$	Rule	STUs	FE	GE	HE	Eeff_1	Eeff_2	LL
2	A	40.11	1762	959	366	15202	7794	115
	B	26.61	1141	668	239	10168	5247	78
	C	22.32	963	574	195	8502	4429	62
	E	24.14	1014	611	212	9132	4730	70
$n+1$	A	37.94	2357	697	205	10844	6375	360
	B	24.93	1542	510	143	7605	4440	143
	C	24.25	1496	491	129	7151	4234	235
	E	21.90	1377	439	118	6489	3841	139

Levy12 ($n = 10$)								
$p \le$	Rule	STUs	FE	GE	HE	Eeff$_1$	Eeff$_2$	LL
2	A	24.71	246	205	76	6476	1826	43
	B	23.90	239	200	74	6309	1779	39
	C	23.90	239	200	74	6309	1779	39
	E	23.90	239	200	74	6309	1779	40
$n+1$	A	19.19	401	106	36	3441	1185	238
	B	17.39	376	97	33	3161	1094	231
	C	18.19	391	101	34	3271	1135	231
	E	16.76	367	89	32	3017	1043	228

Griewank7 ($n = 7$)								
$p \le$	Rule	STUs	FE	GE	HE	Eeff$_1$	Eeff$_2$	LL
2	A	15.57	304	255	114	5281	2122	60
	B	15.12	301	249	111	5152	2074	61
	C	15.23	302	251	112	5195	2090	61
	E	15.36	301	249	111	5152	2074	61
$n+1$	A	11.04	483	125	52	2814	1347	216
	B	11.24	493	129	54	2908	1387	212
	C	10.96	477	125	52	2808	1341	211
	E	10.94	472	123	51	2761	1321	202

Finally, we give an overview on the results for the complete test set by listing the necessary resources for the different variants of our method. The values given are relative values with respect to the method with Rule A and usual splitting.

$p \le$	Rule	Values	STUs	Eeff$_1$	Eeff$_2$	LL
2	B	best	60.3%	62.3%	61.8%	66.7%
		average	95.0%	94.8%	94.8%	102.4%
		worst	131.3%	129.9%	129.4%	178.3%
	C	best	53.2%	54.5%	54.1%	53.9%
		average	93.9%	95.0%	95.0%	98.5%
		worst	137.1%	136.1%	136.8%	169.6%
	E	best	54.8%	55.9%	55.4%	55.6%
		average	95.3%	95.1%	95.1%	99.1%
		worst	141.7%	134.9%	134.5%	157.9%
$n+1$	A	best	54.5%	51.7%	52.2%	100.0%
		average	82.0%	75.0%	78.6%	221.2%
		worst	113.6%	111.2%	111.5%	553.5%
	B	best	45.5%	40.1%	40.6%	66.7%
		average	76.5%	69.6%	72.9%	206.4%
		worst	108.3%	104.7%	105.3%	537.2%
	C	best	45.5%	40.1%	40.6%	66.7%
		average	73.1%	67.5%	70.7%	207.4%
		worst	101.5%	104.0%	104.7%	537.2%
	E	best	45.5%	40.1%	40.6%	71.4%
		average	75.8%	67.8%	71.1%	202.7%
		worst	100.0%	100.0%	100.0%	530.2%

6 Conclusion

Studying the numerical results for the four branching rules combined with different splitting techniques, we recognize that there are test problems for which Rule B, Rule C, and Rule E are much more efficient than Rule A. On the other hand, there are also some problems where the new rules are worse. On average, the branching rules alone lead to an improvement of about 10%.

The special splitting technique improves the performance of the global optimization method significantly, by drastically decreasing the evaluation effort. The price to pay for this improvement is an increasing storage space. Further improvement is due to the branching rules B, C, and E, used as sorting rules in the interval Newton Gauss-Seidel step. This holds for the best cases, the average, and for the worst cases.

Summarizing the consequences of the numerical tests, we can conclude that for Rules B, C, and E combined with the special splitting technique we can expect an average improvement of about 25% in the efficiency of the method, keeping in mind that on average there is approximately a doubling in the necessary storage space.

References

[1] Alefeld, G., Herzberger, J.: *Introduction to Interval Computations*. Academic Press, New York, 1983.

[2] Csendes, T., Ratz, D.: *Subdivision Direction Selection in Interval Methods for Global Optimization.* SIAM Journal of Numerical Analysis, accepted for publication, 1995.

[3] Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: *Numerical Toolbox for Verified Computing I – Basic Numerical Problems*. Springer-Verlag, Heidelberg, New York, 1993.

[4] Hansen, E., Greenberg, R.: *An Interval Newton Method*. Applied Mathematics and Computations **12**, 89–98, 1983.

[5] Hansen, E.: *Global Optimization Using Interval Analysis*. Marcel Dekker, New York, 1992.

[6] Klatte, R., Kulisch, U., Neaga, M., Ratz, D., Ullrich, Ch.: *PASCAL–XSC – Language Reference with Examples*. Springer-Verlag, New York, 1992.

[7] Ratz, D.: *Automatische Ergebnisverifikation bei globalen Optimierungsproblemen*. Dissertation, Karlsruhe, 1992.

[8] Ratz, D.: *Box-Splitting Strategies for the Interval Gauss-Seidel Step in a Global Optimization Method.* Computing **53**, 337–353, Springer-Verlag, Wien, 1994.

[9] Ratz, D., Csendes, T.: *On the Selection of Subdivision Directions in Interval Branch-and-Bound Methods for Global Optimization.* Journal of Global Optimization, **7**, 183–207, 1995.

Address:

D. Ratz, Universität Karlsruhe (TH), Institut für Angewandte Mathematik, D-76128 Karlsruhe, Germany, e-mail: `Dietmar.Ratz@math.uni-karlsruhe.de`.