Journal of Global Optimization 1–11.

Multisection in Interval Branch-and-Bound Methods for Global Optimization II. Numerical Tests*

MIHÁLY CSABA MARKÓT markot@inf.u-szeged.hu
Institute of Informatics, József Attila University, Szeged, Árpád tér 2., Hungary

TIBOR CSENDES csendes@inf.u-szeged.hu
Department of Applied Informatics, József Attila University, Szeged, Árpád tér 2., Hungary

ANDRÁS ERIK CSALLNER csallner@jgytf.u-szeged.hu
Department of Computer Science, Juhász Gyula Teachers Training College, Szeged, Boldogasszony sgt. 4., Hungary

Abstract. We have investigated variants of interval branch-and-bound algorithms for global optimization where the bisection step was substituted by the subdivision of the current, actual interval into many subintervals in a single iteration step. The results are published in two papers, the first one contains the theoretical investigations on the convergence properties. An extensive numerical study indicates that multisection can substantially improve the efficiency of interval global optimization procedures, and multisection seems to be indispensable in solving hard global optimization problems in a reliable way.

Keywords: Branch-and-bound method, Global optimization, Interval arithmetic, Multisection, Accelerating devices

1. Introduction

This paper analyzes the computational efficiency of algorithms solving the unconstrained global optimization problem. In general, we will assume that a nonempty bounded closed n-dimensional interval or box $X \subset \mathbb{R}^n$ containing all global minimizers x^* of the (in most cases continuous) objective function $f : \mathbb{R}^n \to \mathbb{R}$ can always be given. Considering real-life problems this means practically no restrictions on the type of problems considered. Keeping this argumentation in view, the bound constrained global optimization problem has the following form:

$$\min_{x \in X} f(x). \tag{1}$$

* The work has been supported by the Grants AMFK 398/95, FKFP 0739/97, OTKA F 025743, T 016413, T 017241, and MKM 75/96.

The algorithms considered are based on interval arithmetic [7]. We shall denote the inclusion function of the objective function f by $F : \mathbb{I}^n \to \mathbb{I}$, i.e., for $\forall Y \in \mathbb{I}^n$ and $\forall y \in Y \;\; f(y) \in F(Y)$, where $\mathbb{I}$ stands for the set of all bounded closed real intervals. In other words, $f(Y) \subseteq F(Y)$ where $f(Y)$ is the range of f over Y. The lower and upper bounds of an interval $Y \in \mathbb{I}^n$ are denoted by $\mathrm{lb}\,Y$ and $\mathrm{ub}\,Y$, respectively, and the width by $w(Y)$: $w(Y) = \max_i\,(\mathrm{ub}\,Y_i - \mathrm{lb}\,Y_i)$. $\mathbb{I}(X)$ stands for all $Y \in \mathbb{I}^n$ such that $Y \subseteq X$.

Much effort has been made to improve the convergence speed of interval methods for global optimization in the last few decades to enable these reliable methods to solve real-life problems [5, 12, 13, 14]. The main part of this paper develops further an idea [1, 2, 7], subdividing the current subproblem into many ($s > 2$) smaller problems in a single step in contrast to traditional bisection, where two new subintervals are always produced. The general algorithm can be formulated as follows:

Step 1 Let L be an empty list, set the current box $A := X$, and the iteration counter $k := 1$.

Step 2 Subdivide A into a finite number of subsets A_i satisfying $A = \cup A_i$ so that $\mathrm{int}(A_i) \cap \mathrm{int}(A_j) = \emptyset$ for all $i \neq j$ where 'int' denotes the interior of a set.

Step 3 Add the subintervals $\{A_i\}$ to L.

Step 4 Discard certain elements from L that cannot contain a global minimizer.

Step 5 Choose a new $A \in L$ and delete it from the list, $L := L \setminus \{A\}$.

Step 6 While termination criteria do not hold set $k := k + 1$ and go to Step 2.

Step 7 Stop.

Several details of this algorithm have been given in the joint paper [3], here we discuss the interval subdivision direction selection in detail.

1.1. The Subdivision Direction Selection Rule

The selection of subdivision direction is one of the points where the efficiency of the basic branch-and-bound algorithm for unconstrained global optimization can be improved substantially [5, 14].

All the rules select a direction k with a merit function:

$$k := \min\left\{ j \mid j \in \{1, 2, \ldots, n\} \text{ and } D(j) = \max_{i=1}^{n} D(i) \right\}$$

where the function $D(i)$ is determined by the given rule. If many such optimal k indices exist then the algorithm can, e.g., choose the smallest one, or it can select an optimal direction randomly.

The first rule, Rule A, was the interval-width oriented rule. It chooses the coordinate direction with

$$D(i) := w(X_i).$$

Rule B selects the coordinate direction, for which

$$D(i) := w(\nabla_i(X))w(X_i),$$

where ∇_i is the i-th component of the inclusion function of the gradient of $f(x)$. Rule C can be formulated with

$$D(i) := w(\nabla_i(X)(X_i - m(X_i))).$$

The fourth rule, Rule D is derivative-free like Rule A, and reflects the machine representation of the inclusion function $F(X)$. It is again defined by

$$D(i) := \begin{cases} w(X_i) & \text{if } 0 \in X_i, \\ w(X_i)/\operatorname{mig} X_i & \text{otherwise,} \end{cases}$$

where $\operatorname{mig} X$ is the mignitude of the interval X: $\operatorname{mig} X := \min_{x \in X} |x|$.

The theoretical results [5, 14] on the subdivision direction selection rules can be summarized as follows. All the four described rules ensure convergence in the sense that $\lim_{s\to\infty} F(X^s) = f^*$, the set of accumulation points A^* of the current box sequence is not empty, and A^* contains only global minimizer points (where X^s is the current box of the algorithm in the iteration cycle number s). Rules A and D allow convergence also in the sense of $\lim_{s\to\infty} w(X^s) = 0$. For Rules B and C this type of convergence is not always achieved, but for the related instances the respective algorithm variants converge to such a positive width subinterval of the search region X that contains exclusively global minimizer points. The latter type of result set is more valuable for the user than the ones obtained by optimization procedures with Rules A and D.

The conclusions of the numerical tests [5, 14] were essentially the same for the different implementations: the rules B, and C have similar, substantial efficiency improvements against rules A and D, and these improvements were the greater the more difficult the solved problem was. The average performance of Rule D was the worst. Rule C was usually the best, closely followed by Rule B.

2. Numerical Tests

The numerical tests were carried out on various platforms. First we implemented the studied algorithm on a Pentium PC in C-XSC [10] and Borland C with the help of the Numerical Toolbox for Verified Computing [6]. This limited memory environment proved to be not suitable to solve hard problems. Next we have tried an HP 9000-730 workstation of the Institute of Applied Mathematics of the Karlsruhe University, Germany together with the C-XSC – Toolbox environment. This platform allowed satisfying results, yet the third computational environment was substantially quicker. The results to be reported were obtained on a Pentium PC (133 Mhz., 64 Mbyte RAM) equipped with Linux operation system, and the PROFIL/BIAS routines [8] to provide the interval extensions. The standard time unit (1000 evaluations of the non-interval Shekel-5 function) was 0.0125 second. Other programming environments are also available as Fortran-90 [9] and PASCAL-XSC [11].

Several numerical tests were completed to find the most efficient algorithm on which the effects of multisection were investigated. These tests lead us to the interval selection rule that chooses the subinterval with the minimal lower bound of the inclusion functions for further subdivision. The inclusion functions were calculated both by natural interval extension and by central forms. The intersection of these inclusion function ensured good quality lower and upper bounds. Although the first and second derivatives were obtained by automatic differentiation, the inclusion functions were calculated componentwise and in this way some of the components could be skipped if the monotonicity test showed that the objective function was strictly monotonic in a variable.

From the point of view of the numerical efficiency, it is crucial under which condition the interval Newton steps are started. Newton steps are useful only when the boundary of the given subinterval does not contain a global minimizer. If the interval Newton step is carried out unconditionally in each iteration cycle, then it requires for many problems a large amount of additional computation, that will not be justified by the achieved sharper result intervals. This is the reason why some algorithms (e.g. those in [1, 13]) start an interval Newton step only if $w(F(Y))$ and $w(Y)$ were sufficiently small for the current subinterval Y. In our numerical experiences the interval Newton step was started only in those iterations, when one single subinterval remained after the other tests. Although one may expect that if the less expensive acceleration devices were so effective that all but one subintervals were deleted, then there is no reason to start the interval Newton procedure, this starting condition proved to characterize very well those cases when the interval Newton step could improve the efficiency of the whole algorithm.

The working list L was implemented as a dynamical list, and the list operations were made by pointers. It was ordered with increasing lower bounds on the inclusion function. When for a current subinterval selected for further subdivision the width of the inclusion function was smaller than a preset parameter ε, it was moved to a final list containing result intervals. These subintervals were processed further by a few interval Newton steps. This additional algorithm step could improve the width of the result intervals typically by many orders of magnitude. With a few exceptions, for all test problems result intervals were achieved that contain a single global minimizer, a stationary point inside the subinterval (proven by the interval Newton step). The number of final boxes was limited to 100, since for some problems the number of global minimizer points was very high, and thus the processing of the final list could take large portions of the total CPU time necessary — although only the first few result intervals were really useful.

The algorithm was terminated when no interval remained in the working list L. This stopping criterion is substantially stronger than the earlier one [5] that required only that the mentioned relation holds for one interval. This alteration resulted in a better quality result from the user's point of view. The termination condition parameter ε was set to 0.01 in each test.

We have tested numerically the multisplitting technique which was investigated by theoretical tools in the earlier joint paper [3], yet for $s = 3$ and 4 multisection, where the second subdivision direction is also used proved to be even better. It is the reason why the subsequent numerical study used multisection. For $s = 2$ multisplitting and multisection are obviously the same method.

Summarizing the algorithmic changes, the tested procedure was a sophisticated one, equipped with several new features providing tight inclusions of global minimizer points and global minimum values. The studied multisection algorithm variations improved the efficiency of such state of the art techniques. In this way the presented decrease in the necessary CPU times and in the number of objective function and derivative evaluations are net improvements, fully additional to previous ones.

2.1. Numerical Test Results

The numerical tests involved the set of standard global optimization problems (definitions e.g. in [14], further numerical results in [2, 4, 13]) and the set of test problems studied in Hansen's book (descriptions in [7], additional test results in [2, 13]): Shekel-5, 7, 10; Hartman-3, 6; Goldstein-Prize, Six-Hump-Camel-Back, Branin RCOS, Rosenbrock/2, Rosenbrock/5, Three-Hump-Camel-Back, Levy-3, 5, 8-16, 18; Schwefel-2.1 = Beale, 3.1, 3.1p, 2.5 = Booth, 2.18 = Matyas, 3.2, 3.7/5, 3.7/10; Ex-1, Griewank-5, 7; Ratz-4, 5, 6. It is mostly the same set of test problems that was used in [5], yet some of the old problems were omitted (e.g. since they were not twice differentiable everywhere) and some other hard to solve ones were added according to later experience as in [14].

All the test problems were solved, yet for a few of them the final interval Newton steps could not prove that one of the result intervals contains a single global minimizer point. The sharpness of the result intervals and the related inclusion functions were much better that in an earlier study [5]. Each test problem was solved 12 times, by the different algorithm variants: we have tested the A – D interval subdivision direction selection rules and the multisection parameters $s = 2$, 3 and 4. In the subsequent tables only comprised average or sum values are shown, since all single data of the completed comprehensive numerical study would be too lengthy to present.

Table 1 contains the values of three efficiency indicators: the total CPU time necessary in seconds, the average list length MLL, and the sum of function evaluations NFE solving the 37 test problems. The AoP columns give the average of the percentages of the compared efficiency figures on single test problems. The average of the percentages figures reflect the relative computational burden one can expect on a single problem if the given algorithm variant is used instead of A/2, according to the statistical information provided by the set of test problems. Each of the efficiency indicators is also represented in the columns denoted by /(A/2) as the relative value compared to that obtained by the A/2 algorithm variant. In this way the columns /(A/2) reflect the relative improvements to be achieved if we use a given algorithm variant compared to the A/2 procedure solving a set of problems that is similar to the present test problem set. Thus the /(A/2) figures represent better the efficiency improvements achieved on hard to solve problems.

The overall performance of an algorithm variant is represented by the necessary CPU time. The sum of the single CPU times means a kind of weighting that highlights results obtained for hard to solve problems. This value is in general proportional to the number of objective function, gradient and Hessian evaluations. The exceptions are the cases with high memory-complexity. All multisection variants of the algorithms that use the interval subdivision direction selection rules C and B are better than the others. The figures of the bisection variants are similar to earlier results obtained by simpler algorithms [5], thus the present study confirms the generality of the advantages of rules C and B. According to the indicators Sum and /(A/2) the multisection type $s = 3$ is the best for rules A - C, while for rule D the bisection yielded the shortest execution time. The two shortest CPU times were those for algorithms C/3 and C/4, they cause about 88 percent of improvement compared to the CPU time needed by the procedure A/2. It is worth to note that multisection alone brought 22% additional relative improvement to that achieved by rule C. According to the average of percentages values, the two best algorithm

Table 1. The total CPU time necessary in seconds, the average maximal list length used, and the sum of function evaluations solving the 37 test problems. Each of the efficiency indicators CPU, MLL and NFE is also represented in the columns denoted by /(A/2) as the relative value compared to that obtained by the A/2 algorithm variant. The AoP columns give the average of the percentages of the compared efficiency figures on single test problems.

			CPU			**MLL**			**NFE**	
Rule		AoP	Sum	/(A/2)	AoP	Av.	/(A/2)	AoP	Sum	/(A/2)
	/2		4,180			228			502,360	
A	/3	101%	4,134	99%	125%	276	121%	103%	411,439	82%
	/4	110%	5,613	134%	147%	360	158%	113%	434,574	87%
	/2	82%	675	16%	89%	71	31%	83%	131,822	26%
B	/3	84%	551	13%	104%	78	34%	86%	107,046	21%
	/4	100%	562	13%	130%	92	40%	104%	105,306	21%
	/2	81%	666	16%	88%	71	31%	82%	132,395	26%
C	/3	80%	518	12%	103%	73	32%	84%	102,674	20%
	/4	97%	521	12%	128%	84	37%	101%	98,950	20%
	/2	136%	5,369	128%	142%	299	131%	129%	664,474	132%
D	/3	112%	8,830	211%	141%	409	179%	112%	741,433	146%
	/4	110%	12,871	308%	148%	579	254%	111%	801,293	160%

variants were C/3 and C/2 with 80% and 81% relative performance, respectively. The large differences between the values in the AoP and /(A/2) columns are due to the substantially different efficiency improvements on easy and hard to solve problems. For real life problems the later factor is the more important.

The average MLL values of the maximal list lengths used characterize the memory complexity of the given algorithm variant. These figures must be handled with care, since these average values are not direct physical quantities. On the other hand they indicate the minimal memory configuration necessary to solve all the test problems. Again, some high memory complexity problems influence the pattern of average MLL and /(A/2) values stronger than problems that can be solved with small work arrays. A larger s multisection parameter means (with one exception) always slightly larger memory complexity for all of the AoP, av. and /(A/2) values. This phenomenon is fully explained by the larger number of subintervals generated in each iteration cycle. The MLL values in Table 1 for the bisection cases are about half of the figures obtained for a simpler algorithm [5] due to the more effective acceleration tests. All the memory complexity values are acceptable low, much higher MLL figures would mean large additional computational burden due to the related list handling.

In contrast to the necessary CPU times, the number of objective function, gradient and Hessian evaluations characterize the possible computational burden on practical problems which are similar to the test problems, yet the respective function evaluations are more expensive. The sum of the numbers of objective function evaluations (and also that of the gradients and Hessians) must be interpreted with care because the complexities of the individual test problems are different. The sum figures are completed well in this sense by the average of percentages (AoP) values. The computational cost of a gradient and a Hessian evaluation can be higher than that of an objective function depending on the dimension and on other test problem characteristics.

The numbers of objective function evaluations follow basically the trends of the CPU times, albeit with somewhat different proportions. Again the interval selection rules C and B are the best choice, and according to the sum of function evaluations multisection can decrease the computational cost substantially. In the case of the algorithm variants C/2 and C/4 this improvement is more than 25%. The smallest two values were those for C/4 and C/3 with relative improvements of 80% against the basic algorithm A/2. Thus for problems which have high complexity objective function and are similarly hard to solve as the test set, $s = 4$ multisection could provide about one fourth improvement in the computational costs. For easier problems the multisection variants are not justified according to the average of percentages (AoP) values for the number of function evaluations: multisection means always larger AoP values, and this difference is the larger the greater s is. Since easy to solve problems are less important from the point of view of algorithm development, the conclusions for the hard to solve problems have priority for the users.

Table 2 contains numerical results for the number of gradient (NGE), Hessian (NHE) evaluations and the number of necessary iterations (NIT). The sums of the NFE, NGE and NHE indicators are different since the monotonicity test or the interval Newton step requiring inclusions of the gradient and the Hessian, respectively, is carried out only if earlier, computationally less expensive tests were not effective. The NGE and NHE sum values are remarkably stable proportions of the respective NFE figures: NGE/NFE is between 56% and 65%, while NHE/NFE is 3% – 8%. These ratios confirm the results in Table 1 in [3], and they also support our η-assumption (although the NHE/NFE ratio is influenced by other factors too like interval Newton procedure starting condition). In contrast to the computational test resulting in the data of Table 1 of [3], this time the sampling was not uniform, it followed the path of the optimization in the search tree.

According to the number of gradient evaluations, the interval subdivision direction selection rules C and B are the best. The two smallest NGE values were achieved by the algorithm variants C/3 and C/4. The relative advantage of the

Table 2. The sums of the necessary number of gradient and Hessian evaluations, and iterations. Each of the efficiency indicators NGE, NHE and NIT is also represented in the columns denoted by /(A/2) as the relative value compared to that obtained by the A/2 algorithm variant. The AoP columns give the average of the percentages of the compared efficiency figures on single test problems.

		NGE			**NHE**			**NIT**		
Rule		AoP	Sum	/(A/2)	AoP	Sum	/(A/2)	AoP	Sum	/(A/2)
	/2		289,016			39,243			79,422	
A	/3	108%	254,050	88%	96%	24,809	63%	88%	57,964	73%
	/4	123%	283,938	98%	86%	15,852	40%	88%	58,545	74%
	/2	84%	76,234	26%	82%	10,185	26%	81%	19,151	24%
B	/3	92%	66,296	23%	76%	7,071	18%	71%	13,511	17%
	/4	114%	69,046	24%	75%	5,344	14%	80%	12,501	16%
	/2	84%	76,794	27%	82%	10,500	27%	81%	19,053	24%
C	/3	89%	63,629	22%	74%	6,860	17%	70%	12,863	16%
	/4	110%	64,918	22%	70%	5,123	13%	78%	11,628	15%
	/2	126%	373,084	129%	119%	44,789	114%	137%	112,829	142%
D	/3	118%	440,675	152%	101%	41,943	107%	98%	104,188	131%
	/4	121%	522,237	181%	78%	21,592	55%	87%	114,472	144%

$s = 3$ multisection (C/3) compared to bisection (C/2) was more than 17%. The average of percentages values for NGE shows that for simple to solve problems multisection procedures are not better than bisection, regardless which direction selection rule was applied.

The numbers of Hessian evaluations NHE follow different trends: although in general again interval direction selection rules C and B are the most efficient, yet multisection improves efficiency in all the cases, even in terms of average of percentages representing better the easy to solve test problems. According to the NHE values, the most efficient versions are C/4 and B/4, and the improvement of multisection (C/4) compared to bisection (C/2) is more than 51%. Although the present numerical study investigated only $s = 2$, 3 and 4 multisection variants, for the number of Hessian evaluations larger s values could be even better (c.f. [1, 2]). For large dimensional problems the NHE figure is more important than NFE or NGE, thus in such cases multisection with greater s parameter can be suggested.

The number of iterations has no direct effect on the computational effort, yet on the basis of CPU, NFE, NGE, NHE and NIT we can estimate the overhead costs. Also the number of iterations was the lowest for the interval subdivision direction selection rules C and B. For these rules larger s for multisection meant lower number of iterations. The two algorithm variants with the lowest NIT values were C/4 and B/4, thus iteration related overhead costs can be decreased by proper multisection. The AoP figures indicate the smallest NIT values for $s = 3$ multisection, i.e. easy to solve problems are solved in this respect the best by such multisection.

3. Summary and Conclusions

Summarizing the numerical experiences we can conclude that multisection is advantageous in solving hard to solve problems, and algorithm variants C/3 and C/4 are the most efficient according to the majority of indicators. Multisection means usually slightly larger memory complexity. For simple to solve problems the traditional bisection is the best choice. The present study confirmed the results of earlier numerical tests on the advantages of interval subdivision selection rules C and B. The relative improvements to be expected when hard problems must be solved with these versions are as high as 20-22% compared to the best bisection methods. For critical problems these better efficiency values mean that multisection can enable the validated location of the minimizer points. Thus the numerical results indicate that multisection techniques are indispensable in solving hard global optimization problems in a reliable way.

Acknowledgement

The authors acknowledge the kind permission of the Institute of Applied Mathematics of the Karlsruhe University, Germany that allowed to use its HP 9000-730 computer together with the C-XSC – Toolbox environment. The authors are grateful for the useful suggestions of the two anonymous referees.

References

1. Berner, S. (1995), *Ein paralleles Verfahren zur verifizierten globalen Optimierung*, Dissertation, Universität Wuppertal.
2. Berner, S. (1996), New results on verified global optimization, *Computing* 57, 323–343.
3. Csallner, A.E., T. Csendes, and M.Cs. Markót: Multisection in Interval Branch-and-Bound Methods for Global Optimization I. Theoretical Results. Submitted for publication.
4. Csendes, T. and Pintér, J. (1993), The impact of accelerating tools on the interval subdivision algorithm for global optimization, *European J. of Operational Research* 65, 314–320.
5. Csendes, T. and Ratz, D. (1997), Subdivision direction selection in interval methods for global optimization, *SIAM J. Numerical Analysis* 34, 922–938.
6. Hammer, R., Hocks, M., Kulisch, U., and Ratz, D. (1993), *Numerical Toolbox for Verified Computing I.*, Springer, Berlin.
7. Hansen, E. (1992), *Global optimization using interval analysis*, Marcel Dekker, New York.
8. Knüppel, O. (1993), BIAS — basic interval arithmetic subroutines, Technical Report 93.3, University of Hamburg-Harburg.

9. Kearfott, R. B. (1995), A FORTRAN-90 Environment for Research and Prototyping of Enclosure Algorithms for Constrained and Unconstrained Nonlinear Equations, *ACM Transactions on Mathematical Software* 21, 63–78.
10. Klatte, R., Kulisch, U., Lawo, C., Rauch, M., and Wiethoff, A. (1993), *C-XSC — A C++ Class Library for Extended Scientific Computing*, Springer, Heidelberg.
11. Klatte, R., Kulisch, U., Neaga, M., Ratz, D., and Ullrich, Ch. (1992), *PASCAL–XSC — Language Reference with Examples*, Springer, New York.
12. Ratschek, H. and Rokne, J. (1993) Interval Methods, In: Horst R. and Pardalos P. M. (eds.): *Handbook of Global Optimization*, Kluwer, Dordrecht, 751–828.
13. Ratz, D. (1992), *Automatische Ergebnisverifikation bei globalen Optimierungsproblemen*. Dissertation, Universität Karlsruhe.
14. Ratz, D. and Csendes, T. (1995), On the selection of Subdivision Directions in Interval Branch-and-Bound Methods for Global Optimization, *J. Global Optimization* 7, 183–207.