Interval Computations
No 2, 1993

Use of a Real-Valued Local Minimum in Parallel Interval Global Optimization

Ole Caprani, Brian Godthaab, and Kaj Madsen

We consider a parallel method for finding the global minimum (and all of the global minimizers) of a continuous non-linear function $f : D \to \mathcal{R}$, where D is an n-dimensional interval. The method combines one of the well known branch-and-bound interval search methods of Skelboe, Moore and Hansen with a real-valued optimization method.

Initially we use a standard real-valued optimization method to find a local minimizer $\mathbf{x}_p$ (or rather: a prediction of a local minimizer). Then the interval Newton method is applied to an interval I_p containing $\mathbf{x}_p$ as its midpoint. I_p is chosen as large as possible under the restriction that the Newton interval method must converge when I_p is used as starting interval. In this way the original problem has been reduced to the problem of searching a domain $D \setminus I_p$ which does not contain the local (and perhaps global) minimizer. The remaining domain is searched by the branch-and-bound interval method, starting by splitting the remaining domain into $2n$ intervals and hence avoiding I_p. This branch-and-bound search then either verifies that the point $\mathbf{x}_p$ is the global minimizer, or the opposite is detected and it finds the global minimum (and the global minimizers) in the usual way.

The combined method parallizes well. On one test case the combined method is faster than the branch-and-bound method itself. However, for another test case we get the opposite result. This is explained.

Использование вещественнозначного локального минимума для параллельной интервальной глобальной оптимизации

О. Капрани, Б. Годтхааб, К. Мадсен

Рассматривается параллельный метод нахождения глобального минимума и всех глобальных минимизаторов непрерывной нелинейной функции $f : D \to \mathcal{R}$, где D — n-мерный интервал. Подход совмещает в себе один из известных методов ветвей и границ (МВГ) для интервального поиска Скелбоу, Мура и Хансена и вещественнозначный метод оптимизации.

Вначале с использованием стандартного вещественнозначного метода оптимизации находится локальный минимизатор $\mathbf{x}_p$ (точнее, его приближение). Затем к интервалу I_p, содержащему $\mathbf{x}_p$ как среднюю точку, применяется интервальный метод Ньютона. Интервал I_p выбирается насколько возможно бóльшим с учетом того, что метод Ньютона, в котором I_p является начальным интервалом, должен сходиться. При этом исходная задача сводится к поиску области $D \setminus I_p$, не содержащей локального (и, возможно, глобального) минимизатора. В оставшейся области производится поиск с применением интервального МВГ, первым шагом которого является ее деление на $2n$ интервалов (таким образом, избегая I_p). Затем МВГ либо подтверждает, что $\mathbf{x}_p$ является глобальным минимизатором, либо устанавливает обратное. В этом случае глобальный минимум и все глобальные минимизаторы отыскиваются как обычно.

Комбинированный метод хорошо поддается распараллеливанию. На одном из тестовых примеров он работает быстрее, чем сам МВГ. Однако в другом примере получен противоположный результат, что находит свое объяснение.

1 Introduction

Consider a continuous non-linear function $f : D \to \mathcal{R}$, where D is an n-dimensional interval. Consider the problem of finding the global minimum of f over D, $f^* = min\{f(\mathbf{x}) \mid \mathbf{x} \in D\}$ (and the global minimizers $G = \{\mathbf{x} \in D \mid f(\mathbf{x}) = f^*\}$). A branch-and-bound search method based on interval tools [2] can be used to solve this problem. This was first suggested by Skelboe, [1]. During the branch-and-bound search we have:

- a finite number of subintervals S_i of D (called *boxes*). This set of boxes is called the *candidate set* and denoted S.
- lower bounds, $LB(S_i)$ on $min\{f(\mathbf{x}) \mid \mathbf{x} \in S_i\}$,
- an upper bound on the global minimum $f_{bound} \geq f^*$.

Any global minimizer is contained in the union of all the boxes from S :

$$G \subseteq \bigcup_S S_i \subseteq D$$

and furthermore we have that:

$$LB(S_i) \leq f_{bound}.$$

A subbox S_i of S is marked as terminated if either the width of S_i is less than a given accuracy ϵ or the width of an interval extension of f on the subbox S_i is less than ϵ. The purpose of the search is to try to reduce the total volume of S until S consists of only terminated subboxes. This is done as follows: At each iteration of the search the box with the lowest lower bound is selected from the non-terminated subboxes of S (best-first), split into two subboxes (bisection), for each subbox a lower bound is obtained, each subbox is either eliminated (because the lower bound is greater than f_{bound}), marked as terminated and inserted into S or put back into the set S. Also f_{bound} may be updated and further elements of the candidate set eliminated. The search stops when S consists of only terminated subboxes. The convergence properties of this method have been extensively studied by Ratschek and Rokne, [3].

This search method has been refined by Moore, [2] and Hansen, [4], among others, with what Ratschek and Rokne [3] has called accelerating devices. These are the monotonicity test, concavity test, and search for minimizers by means of e.g. an interval Newton method. It is, however, a rather expensive method in terms of computation time if we compare the method with a standard local real-valued optimization methods like those described in [6]. Consider the two test cases listed in Appendix 1. A real-valued optimization method from [7] uses less than a second for each test case on e.g. a T800 transputer, [8]. An interval branch-and-bound method with accelerating devices also implemented on the T800 processor, [14], takes 1858 seconds and 390 seconds on the two test cases. The disappointing computation times are not caused by the interval arithmetic. The implementation makes use of an efficient IEEE based interval arithmetic implementation, [10]. Similar disappointing computation times have been reported in e.g. [11].

Attempts to parallize the branch-and-bound method to get better performance have also been reported e.g. in [11]. For a simple centralized master/slave parallel program structure good efficiencies for up to 32 processors on an Intel iPSC/1 Hypercube have been obtained on problems like the two test cases in Appendix 1. Also on Meiko's T800 based Computing Surface, [9], similar efficiencies have been obtained, [12]. However, even with a speed up of 32 for the parallel branch-and-bound methods, the computation times are large compared to the time used by standard real-valued methods.

This suggests that it might be a good idea to try to make use of the solution found by a real-valued method in the branch-and-bound search

method. This has been tried in [12] where an initial phase of the calculation uses a real-valued method to find a good estimate of the global minimum f^*. This estimate is then used as the initial value of the upper bound f_{bound} used in the branch-and-bound search. If initially $f_{bound} = f^*$, a depth-first search strategy can be used instead of the usual best-first strategy (called breath-first in [12]). The depth-first strategy works the boxes off from S one at a time in a last-in-first-out manner, i.e. the candidate set can be organized as a stack. This requires much less storage for the candidate set than in the best-first strategy, and furthermore, the depth-first strategy parallizes more efficiently than the best-first strategy. In the sequential case the number of iterations is the same in the two methods if $f_{bound} = f^*$ initially. The reason is that the two strategies treat the same boxes of the candidate set, only in a different order. However, if f_{bound} is larger than f^* initially this depth-first strategy is slower than best-first. Hence, this usage of a real-valued method often results in worse computation times and at best gives only minor computation time improvements in the sequential case. If the real-valued initialization of f_{bound} is followed by a best-first search nothing is gained as far as computation time is concerned if global minimizers is searched for. This observation has also been reported by Hansen, [5], and Ratschek and Rokne, [3].

In this paper we consider another usage of a real-valued method. We use not only the minimum found but also the minimizer. As explained in Section 2 this gives a different candidate set to be considered in the search. In the third section we describe results from a sequential implementation of this *combined method*. For one of the test cases the improvement is 28% in computation time, for the other the combined method is 27% slower than the branch-and-bound search in itself. This rather disappointing result is explained. Then results from a parallel implementation are presented. Also this combined method parallelizes well and computation times of 179 seconds and 10 seconds are obtained for the two test cases on 32 processors. All the results have been obtained on Meiko's T800 based Computing Surface. It is reported in detail elsewhere, [14].

2 Initial phase of combined method

Initially we use a standard real-valued optimization method to find a local minimizer $\mathbf{x}_p \in D$. Then a symmetric interval I_p with $\mathbf{x}_p$ as its midpoint is

searched for so that the interval Newton method applied to I_p converges to $\mathbf{x}_p$. I_p is found iteratively. In each step the width of I_p is increased with a fixed step size. This continues until $I_p = D$ or the interval Newton method applied to I_p no longer converges. This means that I_p is chosen as large as possible with the given step size under the restriction that the Newton method converges when I_p is used as a starting interval. Only for the first interval I_p the full Newton iteration is carried out. For the other I_p's only one Newton iteration is performed to ensure existence of the solution. Some results for the two test cases are shown in the following table:

test case	n	$w(I_p)$	time (seconds)
1	10	0.80	79.16
2	3	0.40	0.62

A step size of 0.2 has been used. Unfortunately, the times are rather large. This is caused by the time consuming Newton method used where sets of linear interval equations are solved by the sign accord algorithm, [13]. The Newton method has been used in an attempt to get I_p as large as possible. A Krawzcyk method has also been tried. It is faster but the resulting I_p's are smaller in the two test cases. It might turn out that the width of I_p is not important for the overall performance of the combined method, and therefore we will not improve on the time for the initial phase until further experiments have been carried out with the combined method.

When we have found an interval I_p that does contain a local (and perhaps global) minimizer, the original problem has been reduced to the problem of searching $D \setminus I_p$ for intervals that can be eliminated in the light of the knowledge of I_p and $f_{bound} = f(\mathbf{x}_p)$. Of course, the remaining domain can be split in many different ways into intervals to be used as the initial candidate set in the following branch-and-bound method. However, it is important to avoid a split that results in a number of intervals that grows exponentially with the dimension. We have chosen to split into $2n$ intervals as illustrated in Figure 1 for $n = 2$. In general, if $D = X_1 \times X_2 \times \cdots \times X_n$ each X_i is split into three disjoint intervals $X_{i,1}$, $(I_p)_i$, the i's component of I_p, and $X_{i,2}$. Here, $X_{i,1}$ is the interval to the left of $(I_p)_i$ in X_i and $X_{i,2}$ is the interval to the right of $(I_p)_i$. The $2n$ boxes are then obtained as $S_{2k-1} = ((I_p)_1, \ldots, (I_p)_{k-1}, X_{k,1}, X_{k+1}, \ldots, X_n)$ and $S_{2k} = ((I_p)_1, \ldots, (I_p)_{k-1}, X_{k,2}, X_{k+1}, \ldots, X_n)$, $k = 1, 2, \ldots, n$. The computation time for the splitting and calculation of lower bounds are 0.3 seconds

and 0.03 seconds for the two test cases.

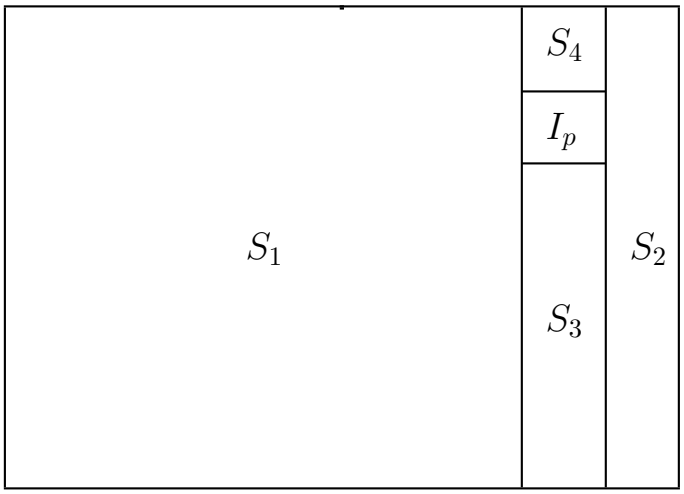

Figure 1: Initial splitting of $D \setminus I_p$ for $n = 2$.

3 Combined sequential method

After the initial phase has initialized the candidate set and the upper bound on the global minimum, the branch-and-bound interval method is applied in a version as described in [12]. In this section we compare the results of the combined method obtained on the two test cases on a single processor with the results obtained when the branch-and-bound is used as in [12]. To see the effect of the initial phase when it succeeds we have initialized $\mathbf{x}_p$ as a global minimizer in the first comparison. For the two test cases the number of iterations used, the computation time and the maximum number of elements in the candidate set during the computation are shown in the tables below.

test case	B&B iterations	time (sec)	combined iterations	time
1	4874	1858	6484	2581
2	6250	390	5545	282

test case	B&B, max candidate set	combined, max candidate set
1	1277	25
2	3647	13

When time is considered the combined method is better in test case 2 and worse in test case 1. Rather disappointing. However, the combined

method is far better when it comes to space requirements. This is caused by the depth-first search strategy. Problems with storage overflow in best-first search have been reported by Hansen, [5], and Ratschek and Rokne, [3]. So depth-first may have its merits.

To investigate the combined method applied to test case 1 a little closer we have traced the $2n = 20$ initial boxes of the candidate set. Half of the boxes are eliminated in one iteration (by the monotonicity test) but one of the boxes (which is almost as large as the original domain D) requires 3267 iterations before it can be eliminated. This is 50% of the iterations. The problem is that $\mathbf{x}_p$ is close to a corner of D. For $n = 2$ the problem can be illustrated by Figure 1. The boxes S_4 and S_2 are easily eliminated. The box S_3 is more difficult, and S_1 is almost as hard to search as D. This phenomena is also illustrated with the results in the table below. This shows the number of iterations required in the branch-and-bound method and in the combined method when different sizes of the initial domain D are used.

domain D	B&B iterations	combined iterations
$[2.001, 9.999]^{10}$	4874	6484
$[3.001, 9.999]^{10}$	3998	6292
$[4.001, 9.999]^{10}$	3955	5039
$[5.001, 9.999]^{10}$	3062	178
$[6.001, 9.999]^{10}$	79	32
$[7.001, 9.999]^{10}$	69	20

In the last line of the table we see that the combined method only requires 20 iterations to eliminate the 20 boxes in S. This is only one iteration for each box in the initial candidate set. This behavior seems to indicate that when the boxes have a width below a certain threshold they are very easily eliminated and above that threshold the exponential nature of the bisection in each dimension results in the large number of iterations.

The splitting strategy has a dramatic influence on the number of iterations needed in test case 1. To carry the experiment a little further we have tried to slice the box that caused 50% of the iterations. It has been sliced into 10 boxes parallel to that side of I_p which is a part of the box. In Figure 1 this corresponds to a vertical slicing of S_1 in 10 tall boxes similar to S_2. The resulting 10 boxes are eliminated immediately by the monotonicity test. If we use the same splitting on the other difficult boxes all of these are also eliminated immediately. The result is that it takes 110 iterations

for the original domain $D = [2.001, 9.999]^{10}$ for the combined method if the slicing is used on all the difficult boxes. Ten iterations are used on the easy boxes and $10 * 10$ on the difficult boxes. This is a dramatic reduction in the number of iterations compared to 6484 and a great improvement over the 4874 of the usual branch-and-bound method. This suggests that it might be a good idea to look into alternative splitting methods in the initial phase after I_p has been found and maybe also in an alternative splitting of a box into subboxes in the branch-and-bound method.

Now, we return to the case when the initial phase fails to find a global minimizer. In test case 1 there is only one minimizer in D and the real-valued method finds this minimizer. But in test case 2 the real-valued method finds a local minimizer $[-0.49, 2.54, 2.54]$ far away from the global minimizer $[-0.49, -0.49, -0.49]$. This results in a larger number of iterations as described in the following table.

$\mathbf{x}_p$	f_{bound}	iterations	time (sec)
$[-0.49, -0.49, -0.49]$	-36.09	5545	282
$[-0.49, 2.54, 2.54]$	-17.35	9391	479

The number of iterations has almost been doubled.

This effect of initial phase failure can be reduced if the real-valued method is also used during the branch-and-bound search to improve f_{bound}. This has been suggested by Ratschek and Rokne, [3].

4 Combined parallel method

The combined method has been parallelized with a simple centralized master/slave program structure as described in [12]. The initial phase is performed by the master before the branch-and-bound search is carried out by the slaves in parallel. The results obtained for the two test cases on 32 processors are shown in the following table:

test case	time (sec)	efficiency
1	$79.2 + 79.5 = 178.7$	0.98
2	$0.6 + 9.8 = 10.4$	0.89

The sum in the second column is the sum of the time it takes to perform the initial phase in the master and the time it takes to do the parallel search in the slaves. The efficiency is a measure of the percentage of time the slaves are busy during the search. In this case the rather fine efficiencies show that the combined method parallelizes well. This was also reported in [12] for a similar depth-first strategy.

The effect of initial phase failure has also been studied for the parallel method. For test case 2 we obtain the following results:

slaves	time (sec)	iterations
1	479	9391
2	159	7215
4	80	7204
8	40	7054
16	26	6985
32	16	8687

The number of iterations used decreases with the number of slaves because the global minimum is found faster when more slaves search in parallel. The increase for 32 slaves is explained by the increasing time it takes when more slaves are present for the master to get a better f_{bound} from the slaves and to broadcast a better f_{bound} to all the slaves.

The effect on the number of iterations of initial phase failure is reduced when the search is done in parallel. This also shows that usage of a real-valued method during the search as suggested in the previous section might decrease the number of iterations even further.

5 Conclusion

We have considered the global optimization problem and solution of this by branch-and-bound interval methods. Our test cases demonstrate that it can be very fast to find a local minimizer by means of a standard real-valued method compared to the time it takes to find the global minimizers by the interval method. If we only use the result of a real-valued method to initialize the bound on the global minimum before we apply an interval branch-and-bound search nothing is gained as far as computation time is concerned.

However, a depth-first search strategy can reduce storage requirements very much.

On the other hand, if we use the local minimizer to guide the splitting of the initial domain into boxes we find that a considerable gain in computation time can be obtained, especially when non-bisection splitting is used. Further test cases need to be investigated to assess the merits of the usage of the local minimizer to guide the splitting during the interval branch-and-bound search.

References

[1] Skelboe, S. *Computation of rational interval functions.* BIT **14** (1974), pp. 87–95.

[2] Moore, R. E. *Methods and applications of interval analysis.* SIAM, Philadelphia, 1979.

[3] Ratschek, H. and Rokne, J. *New computer methods for global optimization.* John Wiley & Sons, 1988.

[4] Hansen, E. *Global optimization using interval analysis — the multidimensional case.* Numerische Mathematik **34** (1980), pp. 247–270.

[5] Hansen, E. *Global optimization using interval analysis.* Marcel Dekker, Inc., 1992.

[6] Dennis Jr., J. E. and Schnabel, R. B. *Numerical methods for unconstrained optimization on nonlinear equations.* Prentice-Hall, 1983.

[7] Madsen, K. and Hegelund, P. *Robust C subroutines for non-linear optimization.* Report 91-03, Institute for Numerical Analysis, The Technical University of Denmark, Lyngby, Denmark, 1991.

[8] Inmos. *Transputer reference manual.* Prentice Hall, 1988.

[9] *Computing Surface technical specifications.* Meiko Ltd, 1987.

[10] Caprani, O. and Madsen, K. *Performance of an Occam/transputer implementation of interval arithmetic.* Lafayette, 1993.

[11] Madsen, K. *Parallel algorithms for global optimization.* Report 91-07, Institute for Numerical Analysis, The Technical University of Denmark, Lyngby, Denmark, 1991.

[12] Henriksen, T. and Madsen, K. *Use of depth-first strategy in parallel global optimization.* Interval Computations 3 (5) (1992), pp. 88–95.

[13] Toft, O. *Sequential and parallel solution of linear interval equations.* Institute for Numerical Analysis, Report 92-04, The Technical University of Denmark, Lyngby, Denmark, 1992.

[14] Godthaab, B. *Parallel interval methods for global optimization.* Institute for Numerical Analysis, The Technical University of Denmark, Lyngby, Denmark, 1993 (in Danish).

O. Caprani
Computer Science Department
Aarhus University
Denmark

B. Godthaab, K. Madsen
Institute for Numerical Analysis
Technical University of Denmark
Denmark

Appendix 1 Test cases

Test case 1:

This is the function of 10 variables:

$$f(\mathbf{x}) = \sum_{i=1}^{n}((\ln(x_i - 2))^2 + (\ln(10 - x_i))^2) - (\prod_{i=1}^{n} x_i)^{0.2}$$

The domain D is $[2.001; 9.999]^{10}$. The accuracy is $\epsilon = 10^{-10}$.

Test case 2:

This is the function of 3 variables:

$$f(\mathbf{x}) = -\sum_{i=1}^{n}\sum_{j=1}^{5} j \sin((j+1)x_i + j)$$

The domain D is $[-5; 5]^3$. The accuracy is $\epsilon = 10^{-10}$.