

ON SPEEDING UP A BASIC GLOBAL OPTIMIZATION ALGORITHM OF INTERVAL ANALYSIS

P. S. V. NATARAJ AND A. K. PRAKASH

*Systems and Control Engineering Group
Department of Electrical Engineering
IIT Bombay 400 076, India.
Email: nataraj@ee.iitb.ernet.in*

ABSTRACT. We propose a modification to the basic global optimization algorithm of interval analysis. The proposed modification consists of processing all boxes present in the list at each algorithmic iteration, instead of processing only one box in the list at each iteration as is currently done. Using INTLAB on a PC/Pentium-III machine, we test and compare the performance of the proposed algorithm on a benchmark suite of thirty five test functions. Our results show that the proposed modification is significantly faster. On an average, it runs about 433 times faster, and somewhat surprisingly, requires less number of function evaluation, flops and list length in about 60 – 80% of the test functions.

Glossary

Bold letters: interval quantities

f - objective function to be optimized

$\bar{f}(\mathbf{X})$ - exact range of f over interval \mathbf{X}

i - index that goes over the subboxes (1 to 2).

j - index that goes over all components (1 to l) present in list L .

k - the maximum width component direction, used for bisection

L - the list at a given iteration

l - the number of components of x , i.e., number of variables to be optimized

l_r - the length of list L at a given iteration

$l_{r'}$ - the (temporary) length of list L at a given iteration

m - the number of components making up the objective function f , i.e., f depends on f_1, \dots, f_m

$m(\mathbf{X})$ - mean of an interval \mathbf{X}

n - index that goes over all items in list L

n_p - the number of test functions

n_s - number of algorithms (solvers)

p - a test function

\mathcal{P} - test set of functions

r - the iteration number

$r_{p,s}$ - performance ratio

$t_{p,s}$ - computing time required to solve a test function p by algorithm s .

v^1, v^2 - minimum value of F over \mathbf{V}^1 and \mathbf{V}^2

$\mathbf{V}^1, \mathbf{V}^2$ - the subboxes obtained by bisection

$w(\mathbf{X})$ - width of an interval \mathbf{X}

x - the variables to be optimized

\mathbf{X} - The search region for optimization

z - the second component of any pair in list L , i.e., minimum value of F over \mathbf{Z} .

\mathbf{Z} - the first component of any pair in list L , a subbox

$\rho_s(\tau)$ - performance profile

1. INTRODUCTION

Let \mathbb{R} be the set of reals, $\mathbf{X} \subseteq \mathbb{R}^l$ be a right parallelepiped parallel to the axes (also called as a box), and $f : \mathbf{X} \rightarrow \mathbb{R}$ be a differentiable function. Let $\bar{f}(\mathbf{X})$ denote the set of all values of f on \mathbf{X} . We seek global optimization algorithms that are able to efficiently determine arbitrarily good lower bounds for the minimum of $\bar{f}(\mathbf{X})$.

Many algorithms based on interval analysis (IA) are available to solve this unconstrained global optimization problem, see for instance, [5], [7], [14]. A model or *basic*¹ branch and bound algorithm of IA consists of the Moore-Skelboe algorithm [9] augmented with the midpoint test of Ichida and Fujii [6] and the monotonicity test detailed in [14]. Although this basic algorithm is *reliable*, it is sometimes found to be slow for ‘difficult’ problems. In this paper, we therefore propose a modification of the same with an aim to improve the speed of the algorithm.

As is well known to interval analysts, in the basic algorithm we choose for processing *only* the first box from the list in each iteration. In the proposed modification, we choose for processing *all* boxes from the list in each iteration. We recently investigated the effect of such a modification in a different context of finding zeros of system of nonlinear equations, see [12], [13]. The modification yielded there algorithmic speed-ups of up to 2 orders of magnitude with INTLAB [16] on a PC/Pentium III. Here, we investigate the effectiveness of the same kind of modification in the context of global optimization. We conduct the investigations on a benchmark suite of thirty five optimization problems given in [10].

2. BASIC ALGORITHM FOR GLOBAL OPTIMIZATION

Let $I(\mathbf{X})$ be the set of all boxes contained in \mathbf{X} . Let the width of \mathbf{X} be defined as $w(\mathbf{X}) = \max \mathbf{X} - \min \mathbf{X}$ if $\mathbf{X} \in I(\mathbb{R})$, and as $w(\mathbf{X}) = \max \{w(\mathbf{X}_1), \dots, w(\mathbf{X}_l)\}$, if $\mathbf{X} \in I(\mathbb{R}^l)$. Let the mean of \mathbf{X} be defined as $m(\mathbf{X}) = (\min \mathbf{X} + \max \mathbf{X}) / 2$ if $\mathbf{X} \in I(\mathbb{R})$, and as $m(\mathbf{X}) = \{m(\mathbf{X}_1), \dots, m(\mathbf{X}_l)\}$, if $\mathbf{X} \in I(\mathbb{R}^l)$. We call a function $F : I(\mathbf{X}) \rightarrow I(\mathbb{R})$ an inclusion function [14] for f , if $\bar{f}(\mathbf{Y}) \subseteq F(\mathbf{Y})$ for all $\mathbf{Y} \in I(\mathbf{X})$.

A Basic Algorithm of IA for Unconstrained Global Optimization [15]

Inputs: The box \mathbf{X} , an inclusion function F (usually the natural interval extension, cf. [8]) for $f : \mathbf{X} \rightarrow \mathbb{R}$, and accuracy parameters ε_F and ε_X .

BEGIN Algorithm

1. Set $\mathbf{Z}_1 = \mathbf{X}$, calculate $F(\mathbf{Z}_1)$, and set $z_1 = \min F(\mathbf{Z}_1)$. Next, initialize list $L = ((\mathbf{Z}_1, z_1))$ and the cut-off value $c = f(m(\mathbf{Z}_1))$.
2. Choose a coordinate direction k parallel to which \mathbf{Z}_1 has an edge of maximum length.
3. Bisect \mathbf{Z}_1 in direction k getting boxes \mathbf{V}^1 and \mathbf{V}^2 such that $\mathbf{Z}_1 = \mathbf{V}^1 \cup \mathbf{V}^2$.
4. Calculate $F(\mathbf{V}^1)$ and $F(\mathbf{V}^2)$, and set $v^1 = \min F(\mathbf{V}^1)$, $v^2 = \min F(\mathbf{V}^2)$.
5. Discard the pair (\mathbf{V}^i, v^i) if $v^i > c$, where $i \in \{1, 2\}$.
6. (Monotonicity test, cf. Remark 2.2) discard the remaining pair (\mathbf{V}^i, v^i) if $0 \notin F'_j(\mathbf{V}^i)$ for any $j \in \{1, 2, \dots, l\}$, and $i = 1, 2$.
7. Add the remaining pair(s) to L . If L is empty, then EXIT. Otherwise, arrange L such that the second members of all pairs of L do not decrease, and denote the pairs as in Remark 2.1. Choose the first item (\mathbf{Z}_1, z_1) and delete it from L .
8. Update the cut-off value as $c = \min \{c, f(m(\mathbf{Z}_1))\}$.
9. (Cut-off test) discard from L all pairs whose second members are greater than the cut-off value c .
10. If the termination criteria hold (cf. Remark 2.3), then print all items in L and EXIT algorithm.
11. Go to Step 2.

¹The basic or model version does not include local search procedures, concavity tests, and Newton-like steps, see [3], [15].

END Algorithm

Remark 2.1. The items in list L at iteration r consists of pairs denoted as $(\mathbf{Z}_1, z_1), \dots, (\mathbf{Z}_{l_r}, z_{l_r})$, where, $z_n = \min F(\mathbf{Z}_n)$, $n = 1, \dots, l_r$, and l_r denotes the current list length.

Remark 2.2. In the monotonicity test, if $0 \notin F'_j(\mathbf{V}^i)$ then the interior of \mathbf{V}^i cannot contain a global minimizer. The edge of \mathbf{V}^i still can contain global minimizer if that part of the edge which has the smallest function values is also part of the edge of \mathbf{X} . Otherwise, no global minimizer lies in \mathbf{V}^i . For details, see [14].

Remark 2.3. We use the termination criteria

$$\max \{w(F(\mathbf{Z}_1)), \dots, w(F(\mathbf{Z}_{l_r}))\} < \varepsilon_F \text{ and } \max \{w(\mathbf{Z}_1), \dots, w(\mathbf{Z}_{l_r})\} < \varepsilon_X$$

3. PROPOSED ALGORITHM FOR GLOBAL OPTIMIZATION

In the basic algorithm described above, in each iteration we choose only the leading box from the list L for processing. On the other hand, in the proposed algorithm given below, we choose *all* boxes from the list for processing. To perform function and gradient evaluations, monotonicity test, midpoint test, width checks, and bisections on *all* boxes in an iteration, we use *vectorized* interval arithmetic operations. We could have also used FOR loops that run over all the boxes to do the same things, but we follow [1] which strongly advocates employment of vectorization instead of FOR loops wherever possible.

In the following example, we illustrate how function evaluation can be done on all boxes from a list with INTLAB [16].

Example 3.1. Consider, for example, the 1- dimensional function

$$F(x_1, x_2) = x_1^3 - x_1x_2^2 + x_1^2 - x_1x_2 - x_2^2$$

and suppose we want to evaluate F over all boxes from a list. Then, using the notation of INTLAB (which is based on MATLAB), we can do this using the single program statement

$$F = \text{power}(x(:,1), 3) - x(:,1) .* \text{sqr}(x(:,2)) + \text{sqr}(x(:,1)) - x(:,1) .* x(:,2) - \text{sqr}(x(:,2))$$

where, $x(:,1)$ denotes x_1 for all boxes and $x(:,2)$ denotes x_2 for all boxes. The function evaluation over all the boxes is done with this statement, because the operations $+$, $-$, $.*$, sqr , and power are performed element-wise between vectors (cf. [1]) and INTLAB overloads these ordinary arithmetic operations with the corresponding interval arithmetic operations.

In a similar way, we can perform gradient evaluations, width checks, bisections, etc., on all boxes from a list. We next present the proposed algorithm.

Proposed Algorithm for Global Optimization

BEGIN Algorithm

1. Set $\mathbf{Z}_1 = \mathbf{X}$, calculate $F(\mathbf{Z}_1)$, and set $z_1 = \min F(\mathbf{Z}_1)$. Next, initialize list $L = ((\mathbf{Z}_1, z_1))$ and the cut-off value $c = f(m(\mathbf{Z}_1))$.
2. Set l_r as the length of L . Then, choose a coordinate direction k_n parallel to which \mathbf{Z}_n has an edge of maximum length, $n = 1, \dots, l_r$.
3. Bisect \mathbf{Z}_n in direction k_n getting boxes \mathbf{V}_n^1 and \mathbf{V}_n^2 such that $\mathbf{Z}_n = \mathbf{V}_n^1 \cup \mathbf{V}_n^2$, $n = 1, \dots, l_r$.
4. Calculate $F(\mathbf{V}_n^1)$ and $F(\mathbf{V}_n^2)$, and set $v_n^i = \min F(\mathbf{V}_n^i)$, for $i = 1, 2$, and $n = 1, \dots, l_r$.
5. Discard the pair (\mathbf{V}_n^i, v_n^i) if $v_n^i > c$, where $i \in \{1, 2\}$, $n \in \{1, \dots, l_r\}$.
6. (Monotonicity test) Discard the remaining pairs (\mathbf{V}_n^i, v_n^i) if $0 \notin F'_j(\mathbf{V}_n^i)$ for any $j \in \{1, 2, \dots, l\}$, and $i = 1, 2, n = 1, \dots, l_r$.
7. Delete all items from L , and enter the remaining pair(s) of above step to L . Set $l_{r'}$ as the (temporary) length of L . If $l_{r'}$ is empty, then EXIT. Otherwise, arrange L such that the second members of all pairs of L do not decrease, and denote the pairs as in Remark 2.1.

8. Update the cut-off value as $c = \min \{c, f(m(\mathbf{Z}_1)), \dots, f(m(\mathbf{Z}_{l_r}))\}$.
9. (Cut-off test) Discard from L all pairs whose second members are greater than the cut-off value c .
10. If the termination criteria hold (cf. Remark 2.3), then print all items in L and EXIT algorithm.
11. Go to Step 2.

Remark 3.1. *Note that the cut-off test in proposed algorithm is based on the function values at midpoints of all the boxes present in the list.*

4. TEST RESULTS

For evaluating the effectiveness of the proposed algorithm, we consider the benchmark suite of thirty five optimization test functions given in [10]. We carry out all computations on a single processor PC/Pentium-III 800 MHz machine with 256 MB RAM using version 3 of INTLAB [16]. In all the problems, given $f_i : \mathbb{R}^l \rightarrow \mathbb{R}$ for $i = 1, \dots, m$, with $m \geq l$, our aim is to find

$$\min \left\{ \sum_{i=1}^m f_i^2(x) : x \in \mathbf{X} \right\}$$

For most test functions, we select the initial domain \mathbf{X} as per Hansen [5, pp.135-136]. We choose the accuracies as $\varepsilon_F = \varepsilon_X = 10^{-2}$ or 10^{-4} . For the rest, since with this initial domain selection both the algorithms fail (due to excessive time requirements), we choose smaller initial domains such that they include all the starting points given in [10].

To compare the performances, we use the following performance metrics

- Number of functional evaluations (fe)
- Number of floating operations (*flops*)
- Computational time, seconds (t)
- Maximum list length (ml)

Table 4 gives the obtained results in terms of these performance metrics for the various test problems² (the notation used for the entries in Table 4 is given in brackets above). At the outset, note that for the considered domains and accuracy, the proposed algorithm is able to solve *all* the test functions, whereas the basic algorithm is able to solve only 77.14% of the test functions.

For each metric, in the last two columns of Table 4 we also give the values of ratio and the percent reduction computed as

$$\begin{aligned} \text{Ratio} &= \frac{\text{Perf. metric with basic algorithm}}{\text{Perf. metric with proposed algorithm}} \\ \text{Percent reduction} &= \frac{\text{Perf. metric with basic algorithm} - \text{Perf. metric with proposed algorithm}}{\text{Perf. metric with basic algorithm}} \times 100 \end{aligned}$$

We compare the performance of the two algorithms using different methods: ranking, statistical measures, average and other measures, and performance profiles.

4.1. Ranking. *Ranking* of the algorithms has been used for performance comparison, for instance, in [2], [11], [17]. Ranking is based on the number of times an algorithm comes in the k^{th} place, here $k = 1, 2$. Table 1 gives the ranking of the algorithms in our studies (A higher rank is assigned to the algorithm with lesser performance metric value).

Table 1 shows that in a majority of the solved test functions, the proposed algorithm is better in terms of various performance metrics used (two test functions require same number of function calls and same maximum list length with either algorithm, and they are assigned both the ranks). Especially, the proposed algorithm is able to achieve the 1st rank in *all* the test functions for the computational time metric.

²A ‘*’ entry in the last two columns of Table 4 indicates that a solution could not be obtained by the basic algorithm for the prescribed accuracy, due to excessive time requirements (greater than 10 hours).

TABLE 1. Rankings

Performance metric	Algorithm with 1 st Rank	Number of problems	Algorithm with 2 nd Rank	Number of problems
Function evaluations	Proposed algorithm	21	Basic algorithm	16
<i>Flops</i>	Proposed algorithm	24	Basic algorithm	11
Computational time	Proposed algorithm	35	Basic algorithm	0
Maximum list length	Proposed algorithm	28	Basic algorithm	09

TABLE 2. Statistical measures

Performance metric	Minimum	First Quartile	Median	Second Quartile	Maximum
Function evaluations	-2432	-442	-2	71208	534138
<i>Flops</i>	-2.74×10^7	-201362	79301	2.483×10^7	1.118×10^{10}
Computational time	0.02	3.6	18.1	1133.2	3.59×10^4
Maximum list length	-10923	-3	4	306	38125

TABLE 3. Minimum, Mean, and Maximum of Ratios and Percent Reductions

Performance Metric	Ratio			Percent Reduction		
	Min.	Mean	Max.	Min.	Mean	Max.
Function Evaluations	0.19	1.85	9.34	-423.66%	-52.38%	89.29%
<i>Flops</i>	0.21	57.05	1235.32	-374.91%	-32.37%	99.92%
Computational time	1.08	433.23	8744	4.66%	81.83%	99.99%
Max. list length	0.43	1.56	8.08	-131.82%	7.72%	87.63%

4.2. Statistical measures. Next, we compare the performance of the algorithms based on the *distribution* of the difference between the performance metrics. Such a comparison has been done, for instance, in [2]. The minimum, first quartile, median, third quartile and maximum of this distribution are reported in Table 2.

A positive value of the median for *flops*, computational time and maximum list length indicates that proposed algorithm requires less *flops*, computational time and maximum list length for more than half of the solved test functions. A negative value of median for the function evaluation shows that proposed algorithm requires more function evaluations for more than half of the test functions. The inter-quartile distance clearly shows the advantage of the proposed algorithm in terms all the performance metrics.

4.3. Minimum, mean, and maximum measures. In Table 3, we give the average (over all test functions) of the ratio and percent reduction defined earlier.

Table 3 shows that with the proposed algorithm, on an average, we obtain an increase in the number of function evaluations and *flops*, but a decrease in maximum list lengths. Moreover, we obtain a good reduction in the computational time with the proposed algorithm for all test functions.

4.4. Performance profiles. Performance profile is proposed as a tool for evaluating and comparing performance of algorithms in [4]. The performance profile for an algorithm is the (cumulative) distribution function for a performance metric. Performance profiles eliminate the influence of a small number of problems on the final evaluation conclusions.

For computational time as the performance metric, performance profiles can be generated as follows. Let \mathcal{P} be the test set of functions, n_s be the number of algorithms and n_p be the number of functions. For each test function p and algorithm s , define

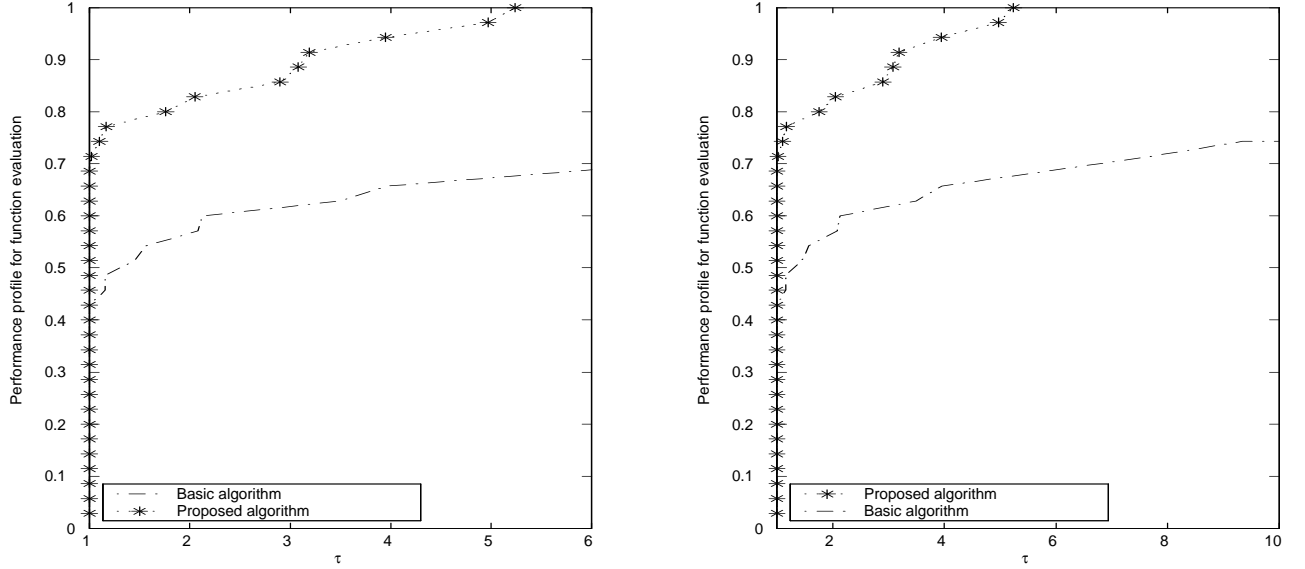


FIGURE 1. Performance profile plots for number of function evaluations

$t_{p,s}$ = computing time required to solve a test function p by algorithm s

The performance ratio for computation time is calculated as

$$r_{p,s} = \frac{t_{p,s}}{\min \{t_{p,s} : 1 \leq s \leq n_s\}}$$

We choose a parameter $r_M \geq r_{p,s}$ for all p, s , such that $r_{p,s} = r_M$ if and only if algorithm does not solve the test function p . Now, the performance profile for computing time can be defined as

$$\rho_s(\tau) = \frac{1}{n_p} \text{size} \{p \in \mathcal{P} : r_{p,s} \leq \tau\}$$

Similarly, performance profiles for other performance metrics can be defined.

The following observations are made from the performance profile plots computed for various performance metrics.

4.4.1. Number of function evaluations. Performance profile plots for the number of function evaluations (cf. Figure 1) show a better performance with the proposed algorithm. The proposed algorithm solves 70% of the (number of) test functions for $\tau = 1$ and remaining 30% for $\tau < 5.5$. The basic algorithm is able to solve only 42.8% of the test functions for $\tau = 1$ and 34.3% of the test functions for $\tau < 10$. This shows that the proposed algorithm is able to solve all the test functions for $\tau < 5.5$, and that it requires less number of function evaluations for 70% of the test functions.

4.4.2. Computational effort (flops). Performance profile plots for computational effort in terms of *flops* are shown in Figure 2. The proposed algorithm solves 71.4% of the test functions for $\tau = 1$, and the remaining 38.6% for $\tau < 4.75$. The basic algorithm could solve only 57.1% of the test functions for $\tau < 1.6$, and 20% of the test functions for $\tau < 1250$. This shows the proposed algorithm is able to solve all the test functions for $\tau < 4.75$, and that 71.4% of the test functions require less computational effort with it.

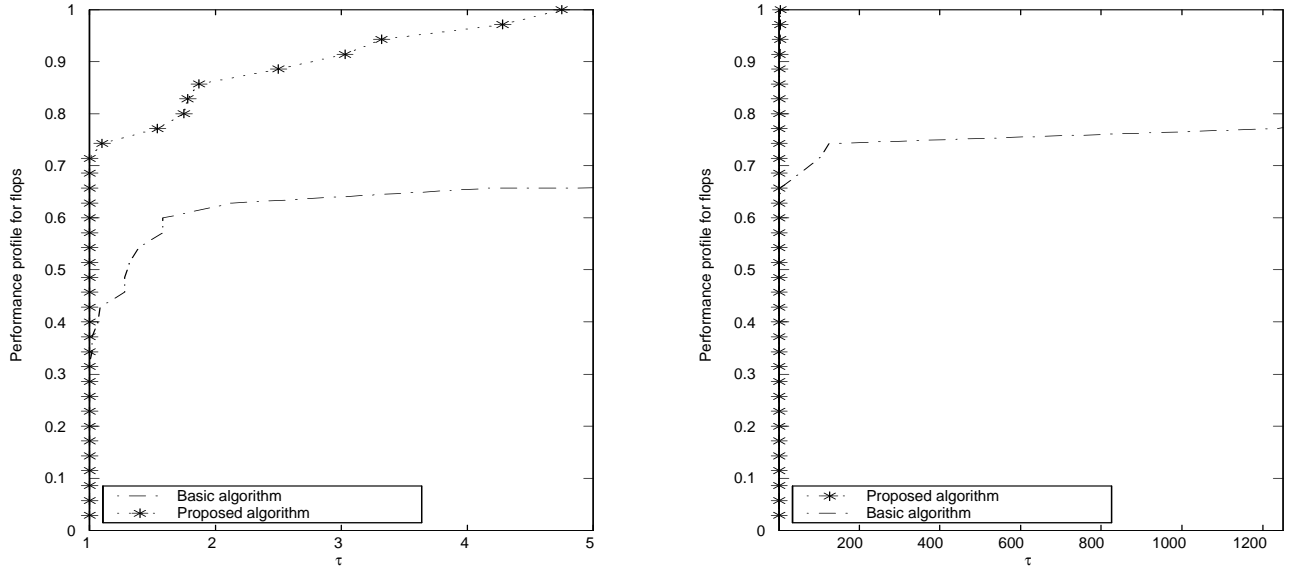
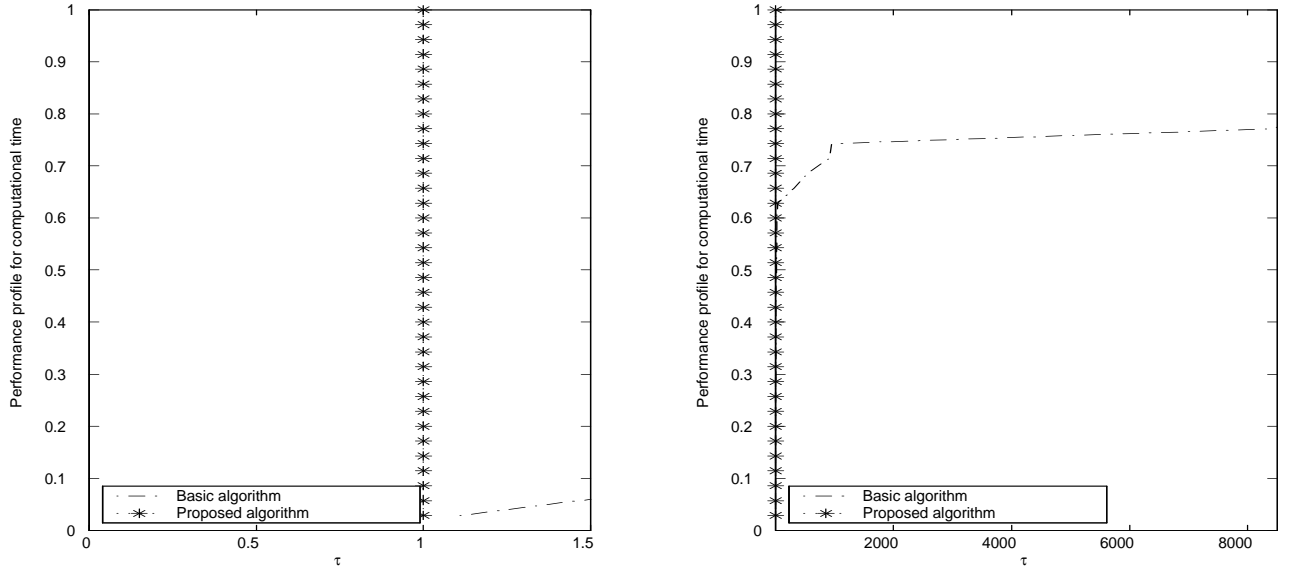
FIGURE 2. Performance profile plots for *flops*.

FIGURE 3. Performance profile plots for computational time.

4.4.3. *Computational time.* Performance profile plots for computational time (cf. Figure 3) show the proposed algorithm is able to solve all the test functions for $\tau = 1$. The basic algorithm is able to solve only 77.1% of the test functions for $\tau < 3000$. In all test functions, the proposed algorithm is considerably faster.

4.4.4. *Maximum list length.* Performance profile plots for maximum list length (cf. Figure 4) show that the proposed algorithm solves 80% of test functions for $\tau = 1$, and remaining 20% for $\tau < 2.5$. The basic algorithm solves just 25.7% for $\tau = 1$, 65.7% of test functions for $\tau < 2$, and the remaining 11.5% for $\tau < 10$. This shows that the proposed algorithm is able to solve all the test function for $\tau < 2.5$, and that it requires less memory for 80% of the test functions.

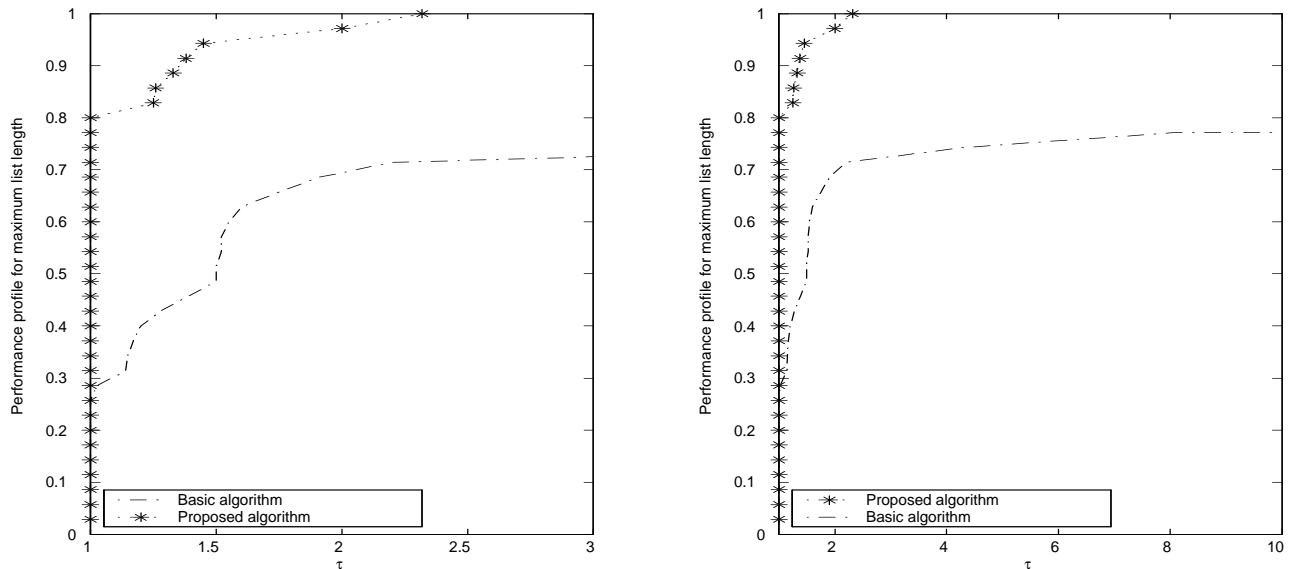


FIGURE 4. Performance profile plots for maximum list length

5. SUMMARY

For the considered domains and accuracy, the proposed algorithm is able to solve *all* the test functions, whereas the basic algorithm is able to solve only about 77% of the test functions.

On an average, the proposed algorithm requires 52% more function evaluations and 32% more computational effort; however, it gives a speed improvement of 82% and requires 8% less list length.

While the improvements in speed are obtained in *all* the test functions, improvements (i.e., reductions) in number of function evaluations, *flops*, and maximum list lengths are obtained in 58%, 69%, and 79% of test functions, respectively.

Lastly, the performance profile plots show a clear superiority of the proposed algorithm over the basic algorithm for all the performance metrics considered.

REFERENCES

- [1] *MATLAB user guide, version 5.3*. The MathWorks Inc., MA, USA, 2000.
- [2] I. Bongartz, A. R. Conn, N. I. M. Gould, and M. A. Saunders. A numerical comparison between the LANCELOT and MINOS packages for large-scale numerical optimization. *Report 97/13 Namur University*, 1997.
- [3] T. Csendes and D. Ratz. Subdivision direction selection in interval methods for global optimization. *SIAM Jl. numerical analysis*, 34:922–938, 1997.
- [4] E. D. Dolan and J. J. More. Benchmarking optimization software with performance profiles. *Mathematical Programming Online*, October 2001.
- [5] E. Hansen. *Global optimization using interval analysis*. Marcel Dekker, 1992.
- [6] K. Ichida and Y. Fujii. An interval arithmetic method for global optimization. *Computing*, 23:85–97, 1979.
- [7] R. B. Kearfott. *Rigorous global search: continuous problems*. Dodrecht: Kluwer Academic Publishers, 1996.
- [8] R. E. Moore. *Methods and applications of interval analysis*. SIAM, Philadelphia, 1979.
- [9] R. E. Moore and H. Ratschek. Inclusion functions and global optimization II. *Mathematical programming*, 41:341–356, 1988.
- [10] J. J. More, B. S. Garbow, and K. E. Hillstom. Testing unconstrained optimization software. *ACM Trans. Mathematical Software*, 7(1):17–41, 1981.
- [11] S. G. Nash and J. Nocedal. A numerical study of the limited memory BFGS method and truncated newton method for large scale optimization. *SIAM J. Optim.*, 1:358–372, 1991.
- [12] P. S. V. Nataraj and A. K. Prakash. A parallelized version of the covering algorithm for solving parameter dependent systems of nonlinear equations. *Reliable Computing*, 8:1–8, 2002.

- [13] A. K. Prakash. On speeding up the interval Newton algorithm. Second Annual PhD progress report, Systems and Control Engg. Group, EE Dept., IIT Bombay, 2001.
- [14] H. Ratschek and J. Rokne. *New computer methods for global optimization*. New York: Wiley, 1988.
- [15] D. Ratz and T. Csendes. On the selection of subdivision directions in interval branch-and-bound methods for global optimization. *Journal of Global Optimization*, 7:183–207, 1995.
- [16] S. M. Rump. INTLAB - interval laboratory. In T. Csendes, editor, *Developments in reliable computing*. Kluwer Academic Publishers, 1999.
- [17] R. J. Vanderbei and D. F. Shanno. An interior point algorithm for nonconvex nonlinear programming. *Comp. Optim. Appl.*, 13:231–252, 1999.

TABLE 4. Performance of the algorithms on test functions

Test function	Dim	Domain	$\varepsilon_F,$ ε_X	Perf. metric	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
1. Rosenbrock	l=2 m=2	$[-1.5, 1.5]^2$	10^{-4}	fc	150	592	0.25	-294.67%
				fl	16,027	53,367	0.30	-232.98%
				t	1.15	0.45	2.55	60.87%
				ml	11	16	0.69	-45.45%
2. Freudenstein and Roth	l=2 m=2	$[-7.5, 7.5]^2$	10^{-4}	fc	1926	2244	0.86	-16.51%
				fl	450,068	348,920	1.28	22.47%
				t	18.90	0.80	23.63	95.77%
				ml	75	66	1.14	12.00%
3. Powell badly scaled	l=2 m=2	$[-10, 10]^2$	10^{-4}	fc	> 244,440	1,122,986	*	*
				fl	> 8.53061×10^9	353,189,839	*	*
				t	> 36000	147.04	*	*
				ml	> 44,184	192,988	*	*
4. Brown badly scaled	l=2 m=3	$[-10^{-5}, 10^5]^2$	10^{-4}	fc	> 2,918,744	104,366	*	*
				fl	> 437,810,978	11,725,776	*	*
				t	> 36000	16.12	*	*
				ml	> 16	16,385	*	*
5. Beale	l=2 m=3	$[-4.5, 4.5]^2$	10^{-4}	fc	756	648	1.17	14.29%
				fl	181,496	130,095	1.39	28.32%
				t	7.50	0.57	13.16	92.40%
				ml	27	23	1.17	14.81%
6. Jenrich and Sampson	l=2 m=10	$[-0.4, 0.4]^2$	10^{-4}	fc	1338	1342	0.99	-0.29%
				fl	3,754,786	3,675,485	1.02	2.10%
				t	13.90	1.20	11.58	91.37%
				ml	70	46	1.52	34.29%
7. Helical valley	l=3 m=3	$[0.001, 1.5]$ $[-1.5, 1.5]^2$	10^{-4}	fc	356	386	0.92	-8.43%
				fl	113,250	114,002	0.99	-0.66%
				t	4.90	1.10	4.45	77.55%
				ml	12	10	1.20	16.67%
8. Bard	l=3 m=15	$[-2.5, 2.5],$ $[0.01, 2.5]^2$	10^{-4}	fc	613,786	79,648	7.71	87.02%
				fl	1.1287×10^{10}	111,450,644	101.27	99.01%
				t	35,983	37.58	957.50	99.89%
				ml	12,714	5781	2.19	54.53%
9. Gaussian	l=3 m=15	$[-1.5, 1.5]^3$	10^{-4}	fc	504	1556	0.32	-208.73%
				fl	2,956,930	8,963,792	0.33	-203.15%
				t	8.22	2.50	3.28	69.59%
				ml	54	72	0.75	-33.33%
10 Meyer	l=3 m=15	$[0.01, 1000],$ $[1000, 4001],$ $[234.7, 260.7]$	10^{-2}	fc	> 534,367	48,290	*	*
				fl	> 1.109×10^{10}	145,900,198	*	*
				t	> 36000	48.22	*	*
				ml	> 4612	9745	*	*

Test function	Dim	Domain	ε_F , ε_X	Perf. metric	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
11. Gulf research and development	l=3 m=16	$[0.1, 5]$, $[0, 5]^2$	10^{-2}	fc	> 416,557	327,062	*	*
				fl	> 9.87128×10^9	523,567,828	*	*
				t	> 36000	170.60	*	*
				ml	> 59,825	44,875	*	*
12. Box three dimensional	l=3 m=3	$[-20, 20]$, $[1, 20]^2$	10^{-4}	fc	290,421	83,179	3.49	71.36%
				fl	67,365,662	42,526,087	1.58	36.87%
				t	3600.81	11.36	316.97	99.68%
				ml	4011	2881	1.39	28.17%
13. Powell singular	l=4 m=4	$[-3, 3]^4$	10^{-4}	fc	5462	5462	1.00	0.00%
				fl	2,470,078	1,556,099	1.58	37.00%
				t	75.20	1.90	39.58	97.47%
				ml	72	72	1.00	0.00%
14. Wood	l=4 m=6	$[-3, 3]^4$	10^{-4}	fc	400	1988	0.20	-397.00%
				fl	169,457	726,677	0.23	-328.83%
				t	6.10	2.00	3.05	67.21%
				ml	22	51	0.43	-131.82%
15. Kowalik and Osborne	l=4 m=11	$[-0.1, 0.2]^4$	10^{-4}	fc	> 193,814	333,996	*	*
				fl	> 1.0133×10^9	760,996,038	*	*
				t	> 36000	307.33	*	*
				ml	> 37,133	57,060	*	*
16. Brown and Dennis	l=4 m=20	$[-25, 25]^4$	10^{-4}	fc	173,221	18,550	9.34	89.29%
				fl	58,184,027	52,903,173	1.09	9.07%
				t	231.91	13.20	17.57	94.31%
				ml	322	207	1.56	35.71%
17. Osborne 1	l=5 m=33	$[0.5, 1.5]$, $[1, 1.5]$, $[-1, 1.5]$, $[0.05, 1.5]$	10^{-2}	fc	39,548	27,359	1.45	30.82%
				fl	299,422,685	279,891,059	1.07	6.81%
				t	386.50	77.19	5.00	80.03%
				ml	1082	936	1.16	13.49%
18. Bigg EXP6	l=6 m=31	$[1, 1.3]$, $[2, 2.3]$, $[1, 1.3]^3$	10^{-2}	fc	79,295	37,993	2.09	52.08%
				fl	485,637,919	228,041,529	2.13	53.04%
				t	1197.90	64.74	18.50	94.60%
				ml	2585	1698	1.52	34.31%
19. Osborne 2	l=6 m=31	See end of Table	10^{-2}	fc	102,810	17,661	5.82	82.82%
				fl	4.579×10^9	1.09795×10^9	4.17	76.02%
				t	3490	322.60	10.82	90.76%
				ml	2343	544	4.31	76.78%
20. Watson	l=6 m=31	$[0, 0.4]^6$	10^{-2}	fc	37,731	38,459	0.98	-1.93%
				fl	271,541,024	298,918,263	0.91	-10.08%
				t	648.50	106.82	6.07	83.53%
				ml	1405	1099	1.28	21.78%

Test function	Dim	Domain	ε_F , ε_X	Perf. metric	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
21. Extended Rosenbrock	$l=4$ $m=4$	$[-1.2, -1.2]^4$	10^{-4}	fc	262	1372	0.19	-423.66%
				fl	71,022	337,294	0.21	-374.91%
				t	2.90	1.30	2.23	55.17%
				ml	11	22	0.50	-100.00%
22. Extended Powell singular	$l=4$ $m=4$	$[-3, 3]^4$	10^{-4}	fc	2672	2674	0.99	-0.08%
				fl	976,677	761,642	1.28	22.02%
				t	33.60	1.50	22.40	95.54%
				ml	33	32	1.03	3.03%
23. Penalty I	$l=4$ $m=5$	$[-4, 4]^4$	10^{-4}	fc	$> 187,224$	184,876	*	*
				fl	$> 1.0245 \times 10^9$	632,126,399	*	*
				t	> 36000	35.53	*	*
				ml	$> 444,742$	27,065	*	*
24. Penalty II	$l=4$ $m=8$	$[-0.5, 0.5]^4$	10^{-4}	fc	$> 192,884$	282,360	*	*
				fl	$> 9.6243 \times 10^9$	451,369,117	*	*
				t	> 36000	156.76	*	*
				ml	$> 40,705$	26,919	*	*
25. Variable dim.	$l=2$ $m=4$	$[-1.5, 1.5]^2$	10^{-4}	fc	42	74	0.57	-76.19%
				fl	8,273	14,370	0.58	-73.69%
				t	0.43	0.41	1.08	4.65%
				ml	2	2	1.00	0.00%
26. Trigonometric	$l=4$ $m=4$	$[-0.25, 0.25]^4$	10^{-4}	fc	882	882	1.00	0.00%
				fl	3,462,401	3,381,955	1.02	2.32%
				t	52.10	5.10	10.22	90.21%
				ml	31	27	1.15	12.90%
27. Brown almost linear	$l=4$ $m=4$	$[-2.5, 2.5]^4$	10^{-4}	fc	266,612	125,362	2.13	52.98%
				fl	2.4498×10^9	46,190,896	53.04	98.11%
				t	10,104	18.50	546.16	99.82%
				ml	5962	3121	1.91	47.65%
28. Discrete boundary value	$l=2$ $m=2$	$[-0.5, 0.5]^2$	10^{-4}	fc	146	126	1.16	13.69%
				fl	30,818	23,414	1.32	24.02%
				t	1.58	0.41	3.85	74.05%
				ml	9	6	1.50	33.33%
29. Discrete integral equation	$l=4$ $m=4$	$[-1, 1]^4$	10^{-4}	fc	132	234	0.56	-77.30%
				fl	141,014	217,454	0.65	-54.21%
				t	4.60	2.60	1.77	43.48%
				ml	8	11	0.73	-37.50%
30. Broyden tridiagonal	$l=4$ $m=4$	$[-1, 1]^4$	10^{-4}	fc	794	2,536	0.31	-219.39%
				fl	348,179	868,626	0.40	-149.47%
				t	10.70	1.60	6.69	85.05%
				ml	85	107	0.79	-25.88%

Test function	Dim	Domain	$\varepsilon_F,$ ε_X	Perf. metric	Basic algorithm	Proposed algorithm	Ratio	Percent reduction
31. Broyden banded	l=4 m=4	$[-1,1]^4$	10^{-4}	fc	396	814	0.49	-105.56%
				fl	231,244	432,606	0.53	-87.07%
				t	7.30	1.80	4.05	75.34%
				ml	56	35	1.60	37.50%
32. Linear full rank	l=4 m=4	$[-1,1]^4$	10^{-4}	fc	1280	3712	0.34	-190.00%
				fl	639,859	1,120,696	0.57	-75.15%
				t	16.10	1.40	11.50	91.30%
				ml	103	59	1.75	42.72%
33. Linear- rank 1	l=4 m=4	$[0,1]^4$	10^{-2}	fc	134888	34,030	3.96	74.77%
				fl	7.961×10^9	6,444,498	1235.32	99.92%
				t	2.6145×10^4	2.99	8744.15	99.99%
				ml	43507	5382	8.08	87.63%
34. Linear rank-1 with zero column and rows	l=2 m=2	$[-1,1]^2$	10^{-4}	fc	196,570	125,362	1.56	36.23%
				fl	6.2032×10^9	49,201,583	126.08	99.21%
				t	2.599×10^4	28.58	909.38	99.89%
				ml	43,690	54,613	0.80	-25.00%
35. Chebyquad	l=3 m=3	$[-1,1]^3$	10^{-4}	fc	> 253,870	62,988	*	*
				fl	> 7.66×10^9	2.3068×10^9	*	*
				t	> 36000	1,617.8	*	*
				ml	> 37,374	5283	*	*

Note: the initial domain for test function number 19, Osborne 2, is

$[1.2, 1.4], [0.6, 0.75], [0.6, 0.75], [0.7, 0.75], [0.5, 0.61], [2.9, 3.1], [4.9, 5.1], [6.9, 7.1], [1.9, 2.1], [4.4, 4.6], [5.4, 5.6]$