Fast interval branch-and-bound methods for unconstrained global optimization with affine arithmetic

Luiz Henrique de Figueiredo
Ronald Van Iwaarden *Jorge Stolfi*

Relatório Técnico IC–97-08

Junho de 1997

Fast interval branch-and-bound methods for unconstrained global optimization with affine arithmetic[*]

Luiz Henrique de Figueiredo[†] Ronald Van Iwaarden[‡] Jorge Stolfi[§]

Abstract

We show that faster solutions to unconstrained global optimization problems can be obtained by combining previous accelerations techniques for interval branch-and-bound methods with affine arithmetic, a recent alternative to interval arithmetic that often provides tighter estimates. We support this claim by solving a few well-known problems.

1 Introduction

In this paper, we consider the *box-constrained global minimization problem*: given a *d-dimensional box* Ω (that is, the Cartesian product of d real intervals), and a continuous *objective function* $f : \Omega \to \mathbf{R}$, find its *global minimum* $f^* = \min\{\, f(x) : x \in \Omega \,\}$, and the set X^* of all *global minimizers* $\Omega^*(f) = \{\, x^* \in \Omega : f(x^*) = f^* \,\}$. Actually, we will consider the approximate numerical version of this problem: instead of finding all minimizers $\Omega^*(f)$, we seek only to identify some subset $\widehat{\Omega}$ of Ω that is guaranteed to contain Ω^*. The goal then becomes to make the measure of $\widehat{\Omega}$ as small as possible, for a given computation budget.

It would seem that finding a global minimum with a computer is a hopeless task. Indeed, this is probably correct, if we are restricted to computing the values of f at a finite set of sample points in Ω, because f may oscillate arbitrarily between these sample points. Nevertheless, there exist numerical techniques for *range analysis* [22] that can provide robust estimates for the complete set of values taken by f in a subregion Δ of Ω. Typically, such methods compute an interval that is *guaranteed* to contain $f(\Delta)$ [18, 20, 22]. Such estimates can be used in branch-and-bound methods to discard large subregions of Ω that cannot contain a global minimum (§2).

[*]Submitted to *SIAM Journal of Optimization* on March 19, 1997.

[†]LNCC–Laboratório Nacional de Computação Científica, Rua Lauro Müller 455, 22290-160 Rio de Janeiro, RJ, Brazil (`lhf@lncc.br`). Supported by research grants from the Brazilian Council for Scientific and Technological Development (CNPq) and Rio de Janeiro Research Foundation (FAPERJ). Research started while visiting the Department of Computer Science, University of Waterloo, Canada, supported by a CNPq post-doctoral fellowship.

[‡]Department of Mathematics, Hope College, Holland, MI 49423 (`rvaniwaa@cs.hope.edu`).

[§]Instituto de Computação, UNICAMP, Caixa Postal 6176, 13083-970 Campinas, SP, Brazil (`stolfi@dcc.unicamp.br`). Partially supported by research grants from the Brazilian Council for Scientific and Technological Development (CNPq) and São Paulo Research Foundation (FAPESP).

Previous global optimization methods based on range analysis use interval arithmetic [10, 23]. In this paper, we show that such methods can be improved by combining previous accelerations techniques, including Back-Boxing [27], with a recently developed alternative to interval arithmetic, called affine arithmetic [2], which often provides much tighter estimates. It has been shown that some methods for numerical problems in computer graphics can be improved, in terms of both speed and accuracy, by replacing interval arithmetic with affine arithmetic [3, 4]. Here, we continue this research, applying affine arithmetic in interval methods for unconstrained global optimization.

In §2, we describe the general branch-and-bound technique, specialized for box-constrained global minimization. In §3, we review some common methods for obtaining robust range estimates, and give a short description of affine arithmetic. In §4, we describe the Back-Boxing [27] acceleration technique for branch-and-bound algorithms. §5 contains an empirical evaluation of Back-Boxing and affine arithmetic for solving some standard global optimization test problems. Finally, §6 contains our conclusions and directions for future work.

2 Branch-and-bound methods for unconstrained global optimization

Branch-and-bound is a general numerical technique for solving global minimization problems. A branch-and-bound algorithm generally alternates between two main steps: *branching*, which is a recursive subdivision of the domain Ω; and *bounding*, which is the computation of lower and upper bounds for the global minimum of f in a subregion of Ω. By keeping track of the current best upper bound for the global minimum of f, one can discard subregions that cannot contain a global minimizer, i.e., subregions where the lower bound for f is greater than the current upper bound for the global minimum f^*. Subregions that cannot be discarded in this way are kept in a list $\mathcal{L}$ to be further processed. Thus, at any time, the set $\widehat{\Omega} = \cup\mathcal{L}$ is a valid solution to the global minimization problem, such as defined in §1. The algorithm stops when the current solution $\widehat{\Omega}$ is adequate for the application (based on the sizes of the boxes in $\mathcal{L}$, on the estimated range for f^*, or on some other criterion).

This algorithm converges provided that: the function f is continuous; the branching step is such that the width of the widest box in $\mathcal{L}$ tends to zero; and the range estimates for $f(\Delta)$ shrink to a single value as the diameter of Δ goes to zero.

The basic branch-and-bound algorithm, outlined above, admits endless variations, depending on how the branching and bounding steps are implemented [11, 14, 15, 24, 25]. The simplest branching method is to bisect the current box orthogonally to its widest direction [17]. Alternatively, one can cyclically bisect along one of the coordinate directions at each step [20]. We now describe some robust bounding methods.

3 Bounding functions

The correctness of general branch-and-bound methods requires range estimates that are guaranteed to contain the values of f in a subregion Δ of Ω. On the other hand, the efficiency of such a method depends on the quality of those estimates. One usually trades quality for speed when computing estimates; however, tight estimates, even if more expensive to compute, sometimes provide overall faster algorithms. We show examples of this phenomenon in §5, when affine arithmetic replaces interval arithmetic in branch-and-bound methods for unconstrained global optimization.

There exist special techniques for range analysis of some classes of functions. For Lipschitz functions f with Lipschitz constant L, a fast estimate for $f(\Delta)$ is given by $[f(x_0) - \delta \mathinner{..} f(x_0) + \delta] \supseteq f(\Delta)$, where x_0 is an arbitrary point of Δ, and $\delta = \max\{\, |x - x_0| : x \in \Delta \,\}$. For polynomial functions, we can represent f in the Bernstein-Bézier basis [5] of the same degree, and then take the interval $[a \mathinner{..} b]$, where a is the minimum value and b is the maximum value of the coefficients of f in the Bernstein-Bézier basis.

In this section, we describe interval arithmetic, a classical range analysis method of wide applicability, and some generalizations, including affine arithmetic [2].

3.1 Interval arithmetic

Interval arithmetic (IA) is the natural technique for computing range estimates [18, 20, 22]. IA was invented by Moore [18] with the explicit goal of improving the reliability of numerical computation. It has since been successfully applied to many numerical problems [20], including global optimization [8, 9, 10, 12, 19, 23, 26].

In IA, a real number x is represented by a pair of floating-point numbers (a, b), corresponding to an interval $[a \mathinner{..} b]$ that is guaranteed to contain x, i.e., such that $a \leq x \leq b$. In this way, IA provides not only an estimate for the value of x, but also bounds on how good this estimate is. The real power of IA is that we can operate with intervals as if they were numbers, and obtain robust estimates for the results of numerical computations.

Simple formulas are easily derived for performing the primitive arithmetic operations on intervals. Interval extensions for a complicated function can be computed by composing these primitive formulas in the same way the primitive operations are composed to compute the function itself [18, 20]. In other words, any algorithm for computing a function using primitive operations can be readily (and automatically) interpreted as an algorithm for computing an interval extension for the same function. (This is specially elegant to implement with programming languages that support operator overloading, such as Ada, C++, Fortran-90, and Pascal-XSC, but can be easily implemented in any programming language, either manually or with the aid of a pre-compiler.) Since it is also relatively easy to provide interval extensions for elementary transcendental functions such as sin, cos, log, and exp, the class of functions for which interval extensions can be easily (and automatically) computed is much larger than the class of rational polynomial functions. These observations are sometimes summarized in the *Fundamental Theorem of Interval Arithmetic*: Every computable function has an interval extension, i.e., for every real function f given by an algorithm, there exists an interval function F such that $F(X) \supseteq f(X) = \{\, f(x) : x \in X \,\}$, for every interval X in the domain of f.

A limitation of IA is that its range estimates tend to be much wider than the exact ranges, sometimes to the point of uselessness. This over-conservatism is mainly due to the implicit assumption that operands in primitive operations are mutually independent. If this assumption is false, then not all combinations of values in the operand intervals will be attained, and the result interval computed by IA may be much wider than the exact range of the result quantity. For an extreme example, consider the computation $y = x - x$, where x has the range $[1 \mathinner{..} 5]$: the IA rules give $[-4 \mathinner{..} +4]$, whereas the exact range of y is of course $[0 \mathinner{..} 0]$. This is sometimes called the *dependency problem* in IA.

The over-conservatism of IA is particularly severe in long computation chains, where one often observes an "error explosion": as the evaluation advances down the chain, the relative accuracy of the computed intervals decreases exponentially, and they soon become too wide to be useful, by many orders of magnitude. Approaches to the dependency problem in IA include *centered forms* [22], Hansen's *generalized interval arithmetic* [7] (§3.2), and, more recently, *affine arithmetic* [2] (§3.3).

3.2 Generalized interval arithmetic

To address the dependency problem in IA, Hansen [7] developed a new computation model, the *generalized interval arithmetic* (GIA). In this model, each input variable x_i is represented by an interval, as in IA, and each intermediate quantity z computed during the evaluation of $f(x_1, \ldots, x_n)$ is represented as an affine combination of the input variables:

$$\tilde{z} = z_0 + z_1 x_1 + \cdots + z_n x_n,$$

where each coefficient z_i is itself an interval. (Only the coefficients z_i are stored for each quantity z; the variables x_i are implicit.) Thus, GIA represents quantities by first-degree polynomials with *interval* coefficients.

As in standard IA, for each elementary operation or transcendental function there is a corresponding GIA procedure that operates on these generalized intervals and returns an analogous representation for the result.

The GIA model keeps track of the contributions of each input variable to the final result, and can exploit this information to produce tighter range estimates for the latter. For example, consider the generalized intervals

$$\begin{aligned} \tilde{y} &= [5 \mathinner{..} 7]x_1 + [3 \mathinner{..} 4]x_2 \\ \tilde{z} &= [4 \mathinner{..} 6]x_1 + [4 \mathinner{..} 5]x_2, \end{aligned}$$

where the variables x_1 and x_2 have ranges $[-10 \mathinner{..} +10]$. From this data, we can conclude that y and z are contained in the interval $[-110 \mathinner{..} +110]$. In the GIA model, the difference $\tilde{w} = \tilde{y} - \tilde{z}$ evaluates to

$$\tilde{w} = [-1 \mathinner{..} +3]x_1 + [-2 \mathinner{..} 0]x_2.$$

This implies that w is in $[-30 \mathinner{..} +30]$, whereas IA gives $[-220 \mathinner{..} +220]$.

3.3 Affine arithmetic

Another model that addresses the dependency problem is *affine arithmetic* (AA) [2]. In AA, each quantity x is represented by an *affine form* $\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots + x_n\varepsilon_n$, where the x_i are known real coefficients (stored as floating-point numbers), and the ε_i are symbolic variables, called *noise symbols*, whose values are unknown but assumed to lie in the interval $[-1 \mathinner{..} +1]$. Thus, AA is superficially similar to GIA in that both represent quantities by a first-degree polynomial on symbolic variables. However, as we shall see below, the number of terms used in AA grows as the computation proceeds, whereas the number of terms used in GIA is fixed, being equal to the number of input variables.

Affine forms implicitly represent partial dependencies between operands: when two affine forms share common noise symbols, the quantities they represent are at least partially dependent on each other. As in GIA, taking such correlations into account allows AA to provide much tighter range estimates than IA, especially in long computation chains. Another major benefit, which is related to this one, is that the intrinsic representation error is quadratic on the size of Ω, instead of linear.

It is simple to use AA for range analysis: First convert all input intervals to affine forms. Then operate on these affine forms with AA to compute the desired function. Finally, convert the result back into an interval.

Conversions between the IA and AA representations are straightforward. Given an interval $\bar{x} = [a \mathinner{..} b]$ representing some quantity x, an equivalent affine form for the same quantity is given by $\hat{x} = x_0 + x_k\varepsilon_k$, where $x_0 = (b+a)/2$ and $x_k = (b-a)/2$. Since input intervals usually represent independent variables, they are assumed to be unrelated, and a new noise symbol ε_k must be used for each input interval. Conversely, the value of a quantity represented by an affine form $\hat{x} = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n$ is guaranteed to be in the interval $[\hat{x}] = [x_0 - \xi \mathinner{..} x_0 + \xi]$, where $\xi = \|\hat{x}\| := \sum_{i=1}^{n} |x_i|$. Note that $[\hat{x}]$ is the smallest interval that contains all possible values of $\hat{x}$, assuming that each ε_i ranges independently over the interval $[-1 \mathinner{..} +1]$.

To evaluate a function f in AA, we must replace each primitive operation ϕ that appears in the expression of f by an equivalent operation $\hat{\phi}$ on affine forms, as done in IA for intervals. For affine operations $z = \phi(x, y)$, the corresponding affine form $\hat{z}$ can be expressed exactly as an affine combination of the noise symbols occurring in the affine forms $\hat{x}$ and $\hat{y}$. More precisely, if $\hat{x} = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n$, $\hat{y} = y_0 + y_1\varepsilon_1 + \cdots + y_n\varepsilon_n$, and $\alpha \in \mathbf{R}$, then:

$$\begin{aligned}
\hat{x} \pm \hat{y} &= (x_0 \pm y_0) + (x_1 \pm y_1)\varepsilon_1 + \cdots + (x_n \pm y_n)\varepsilon_n \\
\alpha\hat{x} &= (\alpha x_0) + (\alpha x_1)\varepsilon_1 + \cdots + (\alpha x_n)\varepsilon_n \\
\hat{x} \pm \alpha &= (x_0 \pm \alpha) + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n.
\end{aligned}$$

Note that the difference $\hat{x} - \hat{x}$ between an affine form and itself is identically zero, a useful property that is not true in IA (or even in GIA). The absence of such trivial cancellations in IA is one source of error explosion.

When the primitive operation ϕ is not affine, the value $\hat{z}$ cannot be expressed exactly as an affine combination of the ε_i. In that case, we pick good affine approximation for ϕ ("good" in the Chebyshev sense of trying to minimize the maximum error), and append an extra term $z_k \varepsilon_k$ to represent the error in this approximation:

$$\hat{z} = z_0 + z_1\varepsilon_1 + \cdots + z_n\varepsilon_n + z_k\varepsilon_k.$$

Here, ε_k is a new noise symbol and z_k is an upper bound for the approximation error. The choice of the affine approximation must take into account the ranges of the operands, as implied by the affine forms $\hat{x}$, $\hat{y}$, and possibly also their correlation. Note that, unlike GIA, the number of terms used in the affine forms grows during the evaluation of a formula. (If necessary, excess terms can be combined and replaced by new noise symbols, at the cost of some loss of precision.)

It is important to note that, for a differentiable operation ϕ, the error bound z_k of a good Chebyshev approximation depends quadratically on the operand ranges $\|\hat{x}\|$ and $\|\hat{y}\|$. Thus, if a function f uses only differentiable operations, then the discrepancy between the actual range of f and the range computed by AA will tend to zero, in *relative* terms, as its domain gets narrowed. In contrast, the internal errors of IA (due to undetected correlations) depend linearly on the size of the domain; therefore, the discrepancy will decrease only in absolute terms.

Adequate approximation algorithms have been developed for the basic arithmetic operations and transcendental functions [2]. For example, in the multiplication of two affine forms $\hat{x}$, $\hat{y}$, we can use the approximation

$$z_0 = x_0 y_0, \qquad z_i = x_0 y_i + y_0 x_i, \qquad z_k = \|\hat{x}\|\,\|\hat{y}\|.$$

The error bound $\|\hat{x}\|\,\|\hat{y}\|$ is not tight, but is at most twice the actual maximum error. (To obtain a tighter bound, we should take into account the correlation of x and y implied by their affine forms, but the gains in precision are hardly worth the cost.)

As described in §3.1 for IA, the formulas for evaluating primitive functions in AA can be automatically combined into formulas for arbitrarily complex functions. This, and the conversion steps, allow AA to transparently replace IA.

As a simple example of how AA handles the dependency problem, consider evaluating $z = x(10 - x)$ with AA, for x in the interval $[4 \,..\, 6]$:

$$\begin{aligned} \hat{x} &= 5 + 1\varepsilon_1 \\ 10 - \hat{x} &= 5 - 1\varepsilon_1 \\ \hat{z} = \hat{x}(10 - \hat{x}) &= 25 + 0\varepsilon_1 + 1\varepsilon_2 \\ [\hat{z}] &= [25 - 1 \,..\, 25 + 1] = [24 \,..\, 26]. \end{aligned}$$

Thus, AA computes an estimate that is very close to $[24 \,..\, 25]$, the exact range of z. On the other hand, the IA estimate is $[16 \,..\, 36]$. If the expression $x(10 - x)$ is expanded out to $10x - x^2$, then the IA estimate becomes significantly worse, $[4 \,..\, 44]$, whereas the AA estimate is actually exact, $[24 \,..\, 25]$.

3.4 Hybrid interval-affine arithmetic

AA's advantage over IA is entirely dependent on its ability to detect and exploit dependencies between operands. When such dependencies do not exist, or happen to agree with IA's conservative assumptions, the AA estimates may be wider than those of IA — even though the former are more accurate than the latter.

The explanation of this paradoxical statement is that IA tries to enclose the graph of each elementary operation with an orthogonal box of minimum vertical extent, whereas AA looks for an oblique box of minimum volume. For example, consider the squaring operation $z = x^2$, for $x \in [1 \,..\, 3]$. IA gives the (tight) range $[1 \,..\, 9]$ for z; whereas AA represents x as $\hat{x} = 2 + 1\varepsilon_1$, and yields $\hat{z} = 4.5 + 4\varepsilon_1 + 0.5\varepsilon_2$, whose range is $[0 \,..\, 9]$. Note that the IA result says only that the pair (x, z) lies in a rectangle of area $2 \times 8 = 16$, whereas the AA result constrains it to a parallelogram of area $2 \times 1 = 2$.

This "undershooting" of AA range estimates unfortunately tends to occur when the functions f is dominated by a square-like term that is being evaluated near its minimum — just where we need precise lower bounds.

This problem can be solved by combining IA with AA. In this hybrid model, each each quantity x is represented by both an affine form $\hat{x}$ and a standard interval $[x]$. Each operation $z = \phi(x, y)$ is carried out in both computing models. The interval part $[z]$ of the result is set to the intersection of the ranges computed by IA and AA; and the AA routine uses the intervals $[x]$ and $[y]$ when choosing the Chebyshev approximation for ϕ. Thus, IA and AA interact synergistically at each step: the explicit intervals given by IA lead to better choices for the best affine approximations and smaller values for the error terms z_k; while the affine forms in AA allow cancellation of correlated errors, and hence lead to narrower explicit intervals. In particular, the hybrid AAIA scheme produces non-negative ranges for $\phi(x) = x^2$, without losing the correlation information provided by AA.

3.5 Rounding error control

When exact arithmetic is used, IA and AA always compute mathematically valid bounds for the range of a sequence of operations. These bounds may overestimate the exact range, but they are always valid. However, IA and AA will be implemented on a digital computer that uses floating-point arithmetic, which is subject to round-off errors. Nevertheless, IA and AA are able to produce mathematically valid bounds on any computer that provides control over rounding. Most modern computers provide rounding error control by implementing the IEEE floating-point standard [1].

A robust implementation of IA should use directed roundings to compute the result of each primitive operation: the lower endpoint of the result interval must be rounded down and the higher endpoint must be rounded up.

Rounding error control in AA is not as easy to implement. One way to implement rounding error control for AA is to have a special noise term that handles only the rounding errors. This term behaves differently from the other error terms in that it is always non-negative, is never decremented, and is operated on similarly to the other error terms, but with absolute values taken everywhere and all subtractions changed to additions. Another implementation of rounding errors in AA can be found in [2].

For IA, the amount of additional work required for directed roundings varies, but is at most twice the amount of work required for not using directed roundings. For AA, the amount of work required is more than doubled for several primitive operations. Fortunately, this does not tend to affect global optimization greatly, except when the stopping tolerance is close to machine accuracy. When this happens, accelerations techniques such as Back-Boxing [27] can be applied to reduce the cost of rounding error control.

4 Accelerating interval branch-and-bound methods

Local search procedures can quickly find the local minimum of a function that is locally convex. Branch-and-bound algorithms, however, tend to spend a great deal of time trying to eliminate regions close to these local minima. Therefore, in practice, it is important to combine pure branch-and-bound methods with additional tests to accelerate convergence.

If the objective function f is sufficiently differentiable, then a simple way to accelerate branch-and-bound methods is to use one of the many excellent fast local optimization methods [6, 16]. Local optimization can be combined with interval tests for the gradient or Hessian: if an interval estimate for the gradient ∇f over a subregion Δ does not contain zero, then Δ cannot contain a local minimizer. If an interval estimate for the Hessian shows that it is positive definite over Δ, then again Δ cannot contain a local minimizer.

Other methods can also be used to discard subregions of Ω [10, 13]. If one obtains an interval estimate on the gradient ∇f over a subregion Δ and finds that this interval does not contain 0, then Δ cannot contain a local minimizer and can be discarded. Similarly, one can test for lack of convexity of f or use a combination of first and second order information to eliminate part or all of Δ [10].

4.1 Back-Boxing

Back-Boxing [27] is an acceleration technique for branch-and-bound algorithms that combines fast local search procedures with result verification techniques to eliminate or reduce much of the work done when one is close to a local minimum. The goal in Back-Boxing is to identify large regions in the domain Ω where the objective function f is guaranteed to have a single local minimum. Once such regions are known, their local minima can be computed efficiently and then tested to see if they are global minima.

Back-Boxing works by first locating a local minimum x^* in a given subregion Δ. It then attempts to find the largest box $B \subseteq \Delta$ such that x^* is the unique local minimum of the box B; the region $\Delta \setminus B$ is then put on the list $\mathcal{L}$ for further processing. Techniques for performing these tasks are described in detail in [27]. Since we now know that the box B has a unique minimum, we no longer need to subdivide B in an attempt to find a smaller function value. We can apply a result verification technique [21] to verify that the unique minimum of the box B occurs in a small region B^* and discard the remainder of the box $B \setminus B^*$.

As with any branch-and-bound acceleration technique, the use of Back-Boxing does not guarantee speedup of a branch-and-bound algorithm. It does, however, attempt to reduce the amount of work spent searching local as well as global minima early in the progress of the algorithm.

The results reported in [27] show that, when a high degree of accuracy is required, Back-Boxing is generally faster than normal branch-and-bound on a given set of test problems with dimension as high as 32. This was not always the case, but it is hoped that, with improved techniques to verify that a box has a unique minimum, the percentage of problems for which Back-Boxing is faster will be increased and the tolerance at which Back-Boxing is faster will also be increased. As shown in §5, Back-Boxing is generally faster with AA as well as with IA.

5 Numerical results

In this section, we compare the performance of IA, AA, and AAIA in solving a number of well-known global optimization problems with three kinds of interval branch-and-bound methods: pure methods, accelerated with local searches; with first-order accelerations (interval gradient test); and with Back-Boxing.

The searches were terminated when no boxes larger in diameter than a specified tolerance remained in the main list $\mathcal{L}$. We solved the problems using increasing tolerances (10^{-3}, 10^{-6} and 10^{-9}), to test the effect of the tolerance in the performance. In the tables below, we give the CPU time in seconds necessary to solve each problem using each of the variants, and also the number of boxes remaining in $\mathcal{L}$ upon completion. A time of ∞ means that the program was unable to complete in 15 minutes of CPU time.

We also give pictures of the domain decompositions using the gradient test and a tolerance of 0.0625% of the domain diameter. In these pictures, boxes shown in white have been eliminated, and boxes shown in grey remain at termination, and are thus guaranteed to contain all global minimizer for f in Ω. Intuitively, a "good" algorithm should generate a picture with few, large white boxes, and few, small grey boxes. This is interpreted as its ability to both quickly discard large subregions of Ω and locate all global minimizers very precisely.

The tests were run on an otherwise idle 166 MHz Pentium machine running Linux, using GNU's g++ compiler. The programs and examples are freely available in the Internet at `http://www.cs.hope.edu/~rvaniwaa/Software.html`.

5.1 Booth

The Booth function is

$$f(x,y) \quad = \quad (x+2y-7)^2 + (2x+y-5)^2.$$

The global minimum of f in the box $\Omega = [-10 \,..\, +10] \times [-10 \,..\, +10]$ is $f^* = 0 = f(1,3)$. Table 1 shows the statistics for solving this problem with all variants. Figure 1 shows the domain decompositions for solving this problem with the gradient test and a relative diameter tolerance of 0.0625%.

This problem was chosen because it highlights the major weaknesses of AA: as discussed in §3.4, AA can perform poorly on functions that are sums of square or when there are pieces that have very little interaction or dependency. This poor performance resulted in longer times for AA and AAIA, but only by a factor of 2–3.

5.2 Exp2

The Exp2 function is

$$f(x,y) = e^{xy}(4x^2 + 2y^2 + 4xy + 2y + 1).$$

The global minimum of f in the box $\Omega = [-10 \mathinner{..} +10] \times [-10 \mathinner{..} +10]$ is $f^* = 0 = f(0.5, -1)$. Table 2 shows the statistics for solving this problem with all variants. Figure 2 shows the domain decompositions for solving this problem with the gradient test and a relative diameter tolerance of 0.0625%.

The quadratic part of this function contains dependencies that allow AA to provide much better estimates. IA provides poor estimates near the global minimum, causing a very large number of boxes in the list for pure branch-and-bound. On the other hand, AA estimates for the exponential part were much worse than the corresponding IA estimates; as consequence, the method could not eliminate many boxes that were distant from the global minimum.

5.3 Goldstein-Price

The Goldstein-Price function is

$$\begin{aligned} f(x,y) &= [1 + (x+y+1)^2(19 - 14x + 3x^2 - 14y - 6xy + 3y^2)] \cdot \\ &\quad [30 + (2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)]. \end{aligned}$$

The global minimum of f in the box $\Omega = [-2 \mathinner{..} +2] \times [-2 \mathinner{..} +2]$ is $f^* = 3 = f(0, -1)$. Table 3 shows the statistics for solving this problem with all variants. Figure 3 shows the domain decompositions for solving this problem with the gradient test and a relative diameter tolerance of 0.0625%.

Goldstein-Price is an easy function for local optimization, but becomes very difficult for a branch-and-bound IA algorithm: The dependency problem generates an unmanageable list of boxes in the pure branch-and-bound case, and causes it to partition the region much more finely with both first and second order tests than is required for AA or AAIA. Since AA and AAIA are able to take advantage of correlations, the run times are 5%–10% that for IA. IA also results in more than 10 times the number of regions with the first order test, and is unable to complete the pure branch-and-bound tests for any of the tested tolerances.

We expect that, in practice, objective functions will have many correlations, as in the Goldstein-Price function. The usual test functions in the literature do not have many correlations, and seem somewhat artificial.

5.4 Levy3

The Levy3 function is

$$\begin{aligned} f(x,y) &= (\cos(2y+1) + 2\cos(3y+2) + 3\cos(4y+3) + 4\cos(5y+4) + 5\cos(6y+5)) \cdot \\ &\quad (\cos(1) + 2\cos(x+2) + 3\cos(2x+3) + 4\cos(3x+4) + 5\cos(4x+5)). \end{aligned}$$

The global minimum of f in the box $\Omega = [-10 \,..\, +10] \times [-10 \,..\, +10]$ is $f^* = -176.542$, attained at one of the following nine points: $(4.97648, 4.85806)$, $(4.97648, -1.42513)$, $(4.97648, -7.70831)$, $(-1.30671, 4.85806)$, $(-1.30671, -1.42513)$, $(-1.30671, -7.70831)$, $(-7.58989, 4.85806)$, $(-7.58989, -1.42513)$, $(-7.58989, -7.70831)$. Table 4 shows the statistics for solving this problem with all variants. Figure 4 shows the domain decompositions for solving this problem with the gradient test and a relative diameter tolerance of 0.0625%.

Levy3 shows that AA is able to take advantage of relationships between sine and cosine functions. Since these functions are just phase shifts of each other, there is a link between their outputs and AA is able to take advantage of that link. For this reason, AA was able to complete the 10^{-6} test for pure branch-and-bound, whereas IA failed. All failed the 10^{-9} test, the AA and AAIA variants due to round-off errors becoming too large. On the other hand, IA was significantly faster for the grad test and Back-Boxing, due to its shorter evaluation time. Also, Back-Boxing provided almost no advantage for these tests because Back-Boxing relies on having a "large" region surrounding the global minima that does not contain any other local minima, and this is not the case with Levy3.

5.5 Discussion

Several features are present in each of the tables and the graphs. First, since AAIA includes the best features of AA and IA, we would expect it to eliminate regions most easily. This is clear in the pictures: the white boxes are largest for AAIA compared to AA and IA.

Second, Back-Boxing results in the smallest number of remaining boxes, indicating its ability to surround the global minimum by a single box rather than spending time with many boxes about the global minimum. Back-Boxing also resulted in almost constant times regardless of tolerance, which duplicates the results found in [27].

Third, AA and AAIA are almost always faster than IA for a pure branch and bound method. This is because AA is able to take advantage of many of the dependencies in the objective function.

Finally, AA and AAIA almost always resulted in the same number or fewer boxes in the final list. This means that the region that has been verified to contain the global minimum is smaller, which gives a better bound on the location of the global minima. Thus, AA and AAIA provide a better solution for the global minimization problem, as defined in §1.

6 Conclusion

Interval branch-and-bound methods are relatively simple to implement, and provide robust and reliable solutions for global optimization problems. Moreover, it is easy to use affine arithmetic instead of interval arithmetic. Although AA is indeed more accurate than IA, it is more complex to implement and more expensive to run. However, as shown by the examples in §5, its higher accuracy is worth the extra cost because it translates into more efficient domain decompositions, even though primitive operations in AA are more expensive than in IA; as a consequence, the final list of boxes is shorter, and sometimes the whole algorithm runs faster.

Sometimes, AA does not provide a tighter estimate than IA. This can be solved by using a hybrid AAIA representation, which is slightly more expensive than pure AA. However, the examples show that the real gain lies in replacing IA by AA, not in improving AA bounds with AAIA. In other words, the gains are primarily a result of AA's ability to handle correlations.

Back-Boxing demonstrated the same performance as seen in [27]. For low tolerances, there is almost no advantage to using Back-Boxing; but for high tolerances, Back-Boxing is almost always faster. Also, when Back-Boxing is combined with AA or AAIA, the algorithm was able to find much larger regions of convexity than with IA alone. This may lead to further improvements in the algorithm.

We plan to continue to investigate other numerical problems that have solutions based on range analysis which would benefit from replacing IA with AA. The obvious next step from the current work is constrained global optimization, but we also plan to study root finding and numerical integration. We expect variants based on AA to be more efficient, but each case requires separate investigation.

References

[1] ANSI/IEEE, *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std 754-1985, IEEE, New York, 1985.

[2] J. L. D. Comba and J. Stolfi, *Affine arithmetic and its applications to computer graphics*, in Proceedings of VI SIBGRAPI (Brazilian Symposium on Computer Graphics and Image Processing), 1993, pp. 9–18. Available at `http://dcc.unicamp.br/~stolfi/`.

[3] L. H. de Figueiredo, *Surface intersection using affine arithmetic*, in Proceedings of Graphics Interface '96, May 1996, pp. 168–175. Available at `ftp://csg.uwaterloo.ca/pub/lhf/gi96.ps.gz`.

[4] L. H. de Figueiredo and J. Stolfi, *Adaptive enumeration of implicit surfaces with affine arithmetic*, Computer Graphics Forum, 15 (1996), pp. 287–296.

[5] R. Farouki and V. Rajan, *Algorithms for polynomials in Bernstein form*, Computer Aided Geometric Design, 5 (1988), pp. 1–26.

[6] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Academic Press, San Diego, CA, 1981.

[7] E. Hansen, *A generalized interval arithmetic*, in Interval Mathematics, K. Nickel, ed., no. 29 in Lecture Notes in Computer Science, Springer Verlag, 1975, pp. 7–18.

[8] ——, *Global optimization using interval analysis — the one-dimensional case*, Journal of Optimization Theory and Applications, 29 (1979), pp. 331–344.

[9] ——, *Global optimization using interval analysis — the multi-dimensional case*, Numerische Mathematik, 34 (1980), pp. 247–270.

[10] ——, *Global Optimization using Interval Analysis*, no. 165 in Monographs and Textbooks in Pure and Applied Mathematics, M. Dekker, New York, 1988.

[11] R. Horst and H. Tuy, *Global Optimization*, Springer-Verlag, Berlin, 1990.

[12] K. Ichida and Y. Fujii, *An interval arithmetic method for global optimization*, Computing, 23 (1979), pp. 85–97.

[13] C. Jansson and O. Knüppel, *A global minimization method: the multi-dimensional case*, Tech. Rep. 92-1, TU Hanburg-Harburg, January 1992.

[14] R. B. Kearfott, *Interval Newton/generalized bisection when there are singularities near roots*, Annals of Operations Research, 25 (1990), pp. 181–196.

[15] R. B. Kearfott, *An interval branch and bound algorithm for bound constrained optimization problems*, Journal of Global Optimization, 2 (1992), pp. 259–280.

[16] D. G. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley, Reading, MA, second ed., 1989.

[17] R. Moore, E. Hansen, and A. Leclerc, *Rigorous methods for global optimization*, in Recent Advances in Global Optimization, C. A. Floudas and P. M. Pardalos, eds., Princeton University Press, Princeton, NJ, 1992, pp. 321–342.

[18] R. E. Moore, *Interval Analysis*, Prentice-Hall, 1966.

[19] ——, *On computing the range of a rational function of n variables over a bounded region*, Computing, 16 (1976), pp. 1–15.

[20] ——, *Methods and Applications of Interval Analysis*, SIAM, Philadelphia, 1979.

[21] A. Neumaier, *Interval Methods for Systems of Equations*, Cambridge University Press, New York, 1990.

[22] H. Ratschek and J. Rokne, *Computer Methods for the Range of Functions*, Ellis Horwood Ltd., 1984.

[23] ——, *New Computer Methods for Global Optimization*, Ellis Horwood Ltd., 1988.

[24] H. Ratschek and R. Voller, *What can interval analysis do for global optimization?*, Journal of Global Optimization, 1 (1991), pp. 111–130.

[25] D. Ratz, *Box-splitting strategies for the interval Gauss-Seidel step in a global optimization method*, Computing, 53 (1994), pp. 1–16.

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

[27] R. van Iwaarden, *An Improved Unconstrained Global Optimization Algorithm*, PhD thesis, Department of Mathematics, University of Colorado at Denver, 1996. Available at `http://www.cs.hope.edu/~rvaniwaa/phd.ps.gz`.

Table 1: Performance statistics for the Booth function.

Branch-and-bound		Pure		with grad test		with Back-Boxing	
Tolerance	IA?	time	boxes	time	boxes	time	boxes
	IA	0.07	4	0.07	4	0.00	1
10^{-3}	AA	0.18	10	0.16	5	0.01	1
	AAIA	0.12	4	0.11	4	0.02	1
	IA	0.12	4	0.12	4	0.00	1
10^{-6}	AA	0.34	10	0.27	5	0.01	1
	AAIA	0.19	4	0.21	4	0.00	1
	IA	0.21	4	0.20	4	0.00	1
10^{-9}	AA	0.52	10	0.43	5	0.00	1
	AAIA	0.29	4	0.32	4	0.00	1

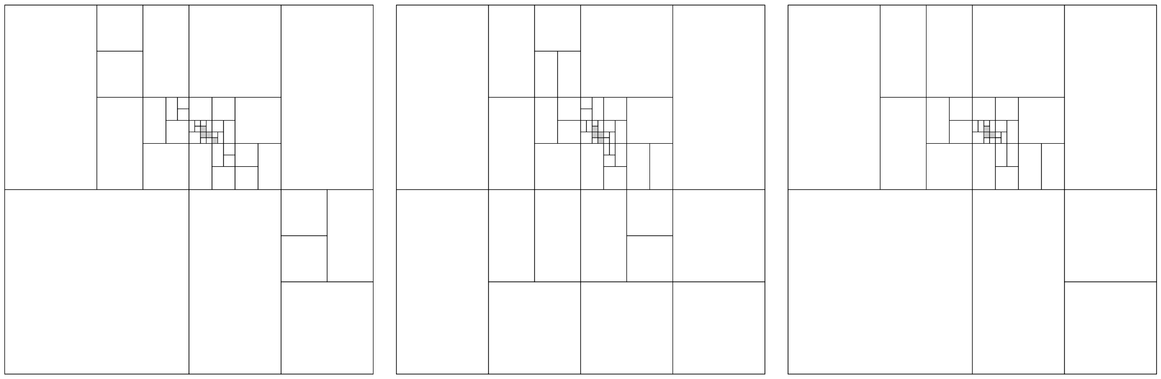

Figure 1: Domain decompositions for minimizing the Booth function with IA (left), AA (center), and AAIA (right).

Table 2: Performance statistics for the Exp2 function.

Branch-and-bound		Pure		with grad test		with Back-Boxing	
Tolerance	IA?	time	boxes	time	boxes	time	boxes
10^{-3}	IA	496.07	20610	0.29	28	0.30	30
	AA	23.50	14	39.34	5	38.69	4
	AAIA	0.41	14	0.48	5	0.59	5
10^{-6}	IA	∞	36727	0.44	28	0.46	28
	AA	23.43	14	39.34	5	38.91	4
	AAIA	0.69	14	0.75	5	0.83	5
10^{-9}	IA	∞	36635	0.59	32	0.61	28
	AA	36.61	4922	40.51	8	39.51	7
	AAIA	12.25	4157	1.12	8	1.18	7

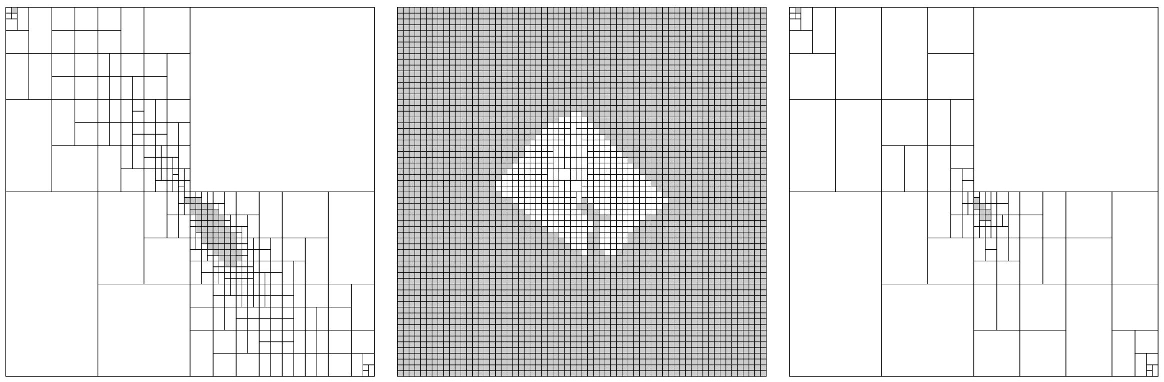

Figure 2: Domain decompositions for minimizing the Exp2 function with IA (left), AA (center), and AAIA (right).

Table 3: Performance statistics for the Goldstein-Price function.

Branch-and-bound		Pure		with grad test		with Back-Boxing	
Tolerance	IA?	time	boxes	time	boxes	time	boxes
10^{-3}	IA	∞	19428	67.44	50	60.56	96
	AA	2.61	8	3.77	4	3.57	1
	AAIA	1.56	8	2.15	4	2.03	1
10^{-6}	IA	∞	19537	63.89	40	65.46	1
	AA	3.18	8	4.33	4	3.59	1
	AAIA	2.23	8	2.81	4	2.06	1
10^{-9}	IA	∞	19921	66.16	40	65.53	1
	AA	10.83	1105	4.90	4	3.59	1
	AAIA	11.47	1103	3.56	4	2.05	1

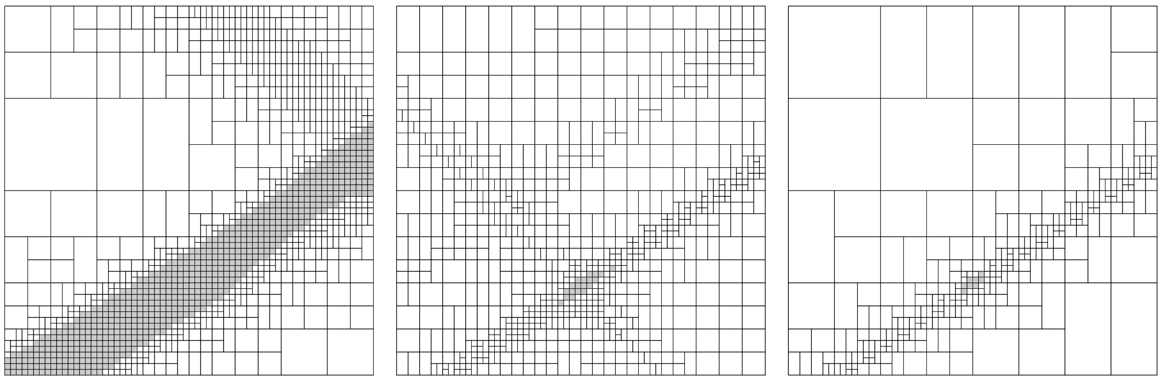
Figure 3: Domain decompositions for minimizing the Goldstein-Price function with IA (left), AA (center), and AAIA (right).

Table 4: Performance statistics for the Levy3 function.

Branch-and-bound		Pure		with grad test		with Back-Boxing	
Tolerance	IA?	time	boxes	time	boxes	time	boxes
10^{-3}	IA	40.20	7692	0.58	9	0.85	9
	AA	10.82	50	8.88	9	9.28	9
	AAIA	9.91	50	6.73	9	5.62	9
10^{-6}	IA	∞	41496	0.80	9	0.88	9
	AA	17.76	48	10.96	9	9.27	9
	AAIA	18.29	48	9.37	9	5.61	9
10^{-9}	IA	∞	42296	1.14	9	0.88	9
	AA	∞	44090	13.47	9	9.32	9
	AAIA	∞	19716	12.92	9	5.66	9

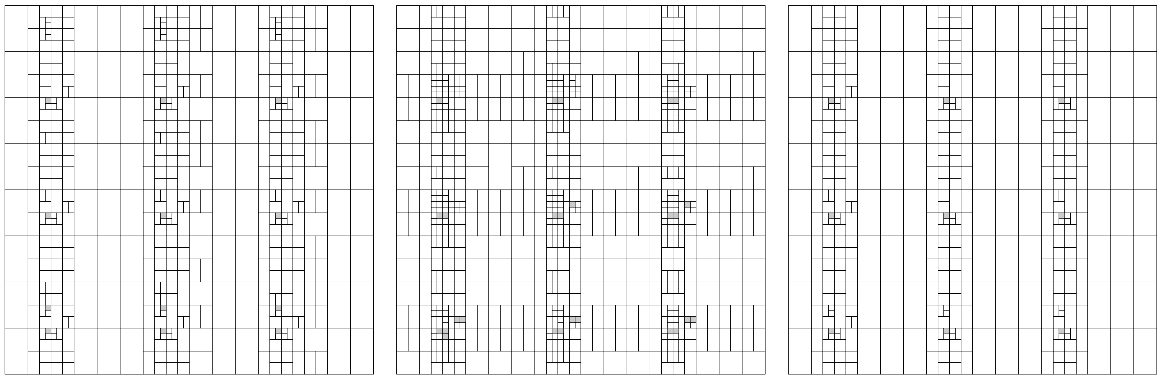
Figure 4: Domain decompositions for minimizing the Exp2 function with IA (left), AA (center), and AAIA (right).

Relatórios Técnicos – 1996

96-01 **Construção de Interfaces Homem-Computador: Uma Proposta Revisada de Disciplina de Graduação,** *Fábio Nogueira Lucena and Hans K.E. Liesenberg*

96Abs **DCC-IMECC-UNICAMP Technical Reports 1992–1996 Abstracts,** *C. L. Lucchesi and P. J. de Rezende and J.Stolfi*

96-02 **Automatic visualization of two-dimensional cellular complexes,** *Rober Marcone Rosi and Jorge Stolfi*

96-03 **Cartas Náuticas Eletrônicas: Operações e Estruturas de Dados,** *Cleomar M. Marques de Oliveira e Neucimar J. Leite*

96-04 **On the edge-colouring of split graphs,** *Celina M. H. de Figueiredo, João Meidanis and Célia Picinin de Mello*

96-05 **Estudo Comparativo de Métodos para Avaliação de Interfaces Homem-Computador,** *S'ılvio Chan e Heloisa Vieira da Rocha*

96-06 **User Interface Issues in Geographic Information Systems,** *Juliano Lopes de Oliveira and Claudia Bauzer Medeiros*

96-07 **Conjunto fonte máximo em grafos de comparabilidade,** *Marcos Fernando Andrielli e Célia Picinin de Mello*

96-08 **96-08 The Effectiveness of Multi-Level Policing Mechanisms in ATM Traffic Control,** *J.A. Silvester, N. L. S. Fonseca, G. S. Mayor e S. P. S. Sobral*

96-09 **Sequential and Parallel Experimental Results with Bipartite Matching Algorithms,** *João Carlos Setubal*

96-10 **96-10 A CPU for Educational Applications Designed with VHDL and FPGA,** *Nelson V. Augusto, Mario L. Côrtes and Paulo C. Centoducatte*

96-11 **Network Design for the Provision of Distributed Home Theatre Services,** *Nelson L. S. Fonseca, Cristiane M. R. Franco, Frank Schaffa*

96-12 **Modelling the Output Process of an ATM Multiplexer with Correlated Priorities,** *Nelson L. S. Fonseca e John A. silvester*

96-13 **Algoritmos de afinamento tridimensional: exemplos de técnicas de otimização,** *F. N. Bezerra and N. J. Leite*

96-14 **Ensino de Estruturas de Dados e seus Algoritmos através de Implementação com Animações,** *Pedro J. de Rezende e Islene C. Garcia*

96-15 **Sinergia em Desenho de Grafos Usando Springs e Pequenas Heurísticas,** *H. A. D. do Nascimento, C. F. X. de Mendonça N., P. S. de Souza*

96-16 **A Temporal Extension to the Parsimonious Covering Theory,** *Jacques Wainer and Alexandre de Melo Rezende*

96-17 **Workcase-centric workflow model,** *Jacques Wainer and Paulo Barthelmess*

96-18 **Integrating heuristics and spatial databases: a case study,** *Cid Carvalho de Souza, Claudia Bauzer Medeiros, Ricardo S. Pereira*

96-19 **Uma Abordagem de Programação Inteira para o Problema da Triangulação de Custo Mínimo,** *A. P. Nunes e C. C. de Souza*

96-20 **Contracting and Moving Agents in Distributed Applications Based on a Service-Oriented Architecture,** *B.Schulze and E.R.M.Madeira*

Relatórios Técnicos – 1997

97-01 **Um Ambiente Distribuído de Visualização com Suporte para Geometria Projetiva Orientada,** *Pedro J. de Rezende e César N. Gon*

97-02 **Approximate Models for the Output Process of an ATM Multiplexer with Markov Modulated Input,** *Nelson L. S. Fonseca and John A. Silvester*

97-03 **Controle de Concorrência no Cm,** *Célio Norbiato Targa, Mauro da Silva Oliveira Filho, Celso Gonçalves Jr, Rogério Drummond*

97-04 **Compilação Condicional em Cm,** *Sheila P. Maceira, Alexandre Prado Teles, Rogério Drummond*

97-05 **Linear: Linearizador de Estruturas Complexas,** *Rogério Drummond, Carlos Hoyos*

97-06 **LegoShell: Linguagem Visual de Programação Distribuída,** *Rogério Drummond, Celso Gonçalves Jr.*

97-07 **Desenvolvendo Aplicações Distribuídas em Cm,** *Celso Gonçalves Jr., Alexandre Prado Teles, Rogério Drummond*

Instituto de Computação
Universidade Estadual de Campinas
13083-970 – Campinas – SP
BRASIL

`reltec@dcc.unicamp.br`