Optimization Methods and Software
Vol. 00, No. 00, Month 200x, 1–16

RESEARCH ARTICLE

Algorithmic Differentiation Techniques for Global Optimization in the COCONUT Environment

Hermann Schichl* and Mihály Csaba Markót.

Universität Wien, Nordbergstraße 15, A-1090 Wien, Austria

(*Received 00 Month 200x; in final form 00 Month 200x*)

We describe algorithmic differentiation as it can be used in algorithms for global optimization. We focus on the algorithmic differentiation methods implemented in the COCONUT Environment for global nonlinear optimization.

The COCONUT Environment represents each factorable optimization problem as a directed acyclic graph (DAG). Various inference modules implemented in this software environment can serve as building blocks for solution algorithms. Many of them use techniques based on various forms of algorithmic differentiation for computing approximations or enclosures of functions or their derivatives.

The algorithmic differentiation in the COCONUT Environment does not only provide point evaluations but also range enclosures of derivatives up to order 3, as well as slopes up to second order. Care is taken to ensure that rounding errors are treated correctly. The ranges of the enclosures can be tightened by combining the evaluation routines with constraint propagation. Advantages and pitfalls of this method are also outlined.

Keywords: global optimization, directed acyclic graphs, automatic differentiation, slope, interval analysis

AMS Subject Classification: 65G40, 90C26, 90C30

1. Introduction

The COCONUT Environment [50, 59] is a public domain software platform for implementing algorithms for solving factorable global optimization problems [16, 43]. It is best tailored to implementing deterministic algorithms which usually use branch-and-bound like schemes [2, 22, 30, 31, 35, 48, 56]. The success of such methods heavily relies on the quality of the range estimates computed for the functions involved, and since local optimization is usually indispensable for a successful algorithm, also the quality and speed of function evaluation and evaluation of derivatives is important. Derivatives of second order can significantly speed up

*Corresponding author. Email: hermann.schichl@univie.ac.at
This research was supported by the Austrian Science Foundation FWF Grant nr. P22239-N13 and by the Hungarian National Development Agency (NFÜ) Grant TÁMOP-4.2.2/08/1/2008-0008

ISSN: 1055-6788 print/ISSN 1029-4937 online

DOI: 10.1080/03081080xxxxxxxxx
http://www.informaworld.com

local solvers [42], and are used in interval Newton-methods. Third derivatives can be, e.g., used to avoid the cluster effect close to the global minima [36, 52].

The algorithmic differentiation (AD) in the COCONUT Environment has the advantage as compared to, e.g. ADIFOR [8, 9], ADOL-C [26], that it can not only compute point evaluations for higher derivatives but also range enclosures, where roundoff errors are treated rigorously. This rigor is provided to ensure mathematical correctness of the enclosures in a floating point environment. As compared to other interval based libraries, like Profil/BIAS [37] and the C++ Toolbox for Verified Computing [29] for C-XSC, it has the advantage that it provides higher order enclosures. In addition it provides the only implementation for computing slopes of first and second order, which have a clear advantage over interval gradients and Hessians for improving interval range enclosures. As a platform for global optimization the AD tools enable the COCONUT Environment to solve problems with general nonlinear factorable functions and to interface general nonlinear local solvers [42].

In Section 2 we briefly recall the **directed acyclic graphs** (DAGs) used to represent optimization problems. They have traditionally been used in automatic differentiation (AD) [27, 28] and have since proved very useful for global optimization, too. We will shortly focus on the advantages DAGs provide in global optimization and talk about the difference to computational trees and DAGs which are provided by parsers of high-level programming language compilers like `FORTRAN 90`, `C++`, or the parsers of modeling languages like `AMPL` [23] or `GAMS` [13].

Section 3 explains the basic evaluation algorithms used in the COCONUT Environment for computing function values, ranges, first to third derivatives, and first and second order slopes.

In [53] and [58] it was outlined that one of the strengths of the DAG concept is that it is a suitable representation both for efficient evaluation and for performing efficient constraint propagation (CP). The results of constraint propagation, especially the ranges of the inner nodes, can be used to improve the ranges of the standard evaluation methods for interval derivatives, and slopes of all orders. The principles and pitfalls are outlined in Section 4.

Our notation follows the notation suggested in [45]. In particular, inequalities between vectors are interpreted component-wise, I denotes the identity matrix, e_i the ith unit vector, intervals and boxes are written in bold face, and $\operatorname{rad} \boldsymbol{x} = \frac{1}{2}(\overline{x} - \underline{x})$ denotes the radius of a box $\boldsymbol{x} = [\underline{x}, \overline{x}] \in \mathbb{IR}^n$, here $\mathbb{IR}$ is the set of real, possibly unbounded, intervals.

2. Directed acyclic graphs

This section is devoted to a short review of the definition of the DAGs used to represent the global optimization problems as used in the [53] and in the COCONUT Environment.

DEFINITION 2.1 *A **directed multigraph** $\Gamma = (V, E, f)$ consists of a finite set of vertices (nodes) V, a finite set of edges E, and a mapping $f : E \to V \times V$. For every edge $e \in E$ we define the **source** of e as $s(e) := \mathrm{Pr}_1 \circ f(e)$ and the **target** of e as $t(e) := \mathrm{Pr}_2 \circ f(e)$, where Pr_i denotes the projection onto the ith component in*

a Cartesian product. An edge e with $s(e) = t(e)$ is called a ***loop****. Edges $e, e' \in E$ are called* ***multiple*** *if $f(e) = f(e')$.*

For every vertex $v \in V$ we define the set of ***in-edges***

$$E_i(v) := \{e \in E \mid t(e) = v\}$$

as the set of all edges which have v as their target, and the set of ***out-edges*** *analogously as the set*

$$E_o(v) := \{e \in E \mid s(e) = v\}$$

of all edges with source v. The ***indegree*** *of a vertex $v \in V$ is defined as the number of in-edges $\text{indeg}(v) = |E_i(v)|$, and the* ***outdegree*** *of v as the number of out-edges $\text{outdeg}(v) = |E_o(v)|$.*

A vertex $v \in V$ with $\text{indeg}(v) = 0$ is called a ***(local) source*** *or* ***leaf*** *of the graph, and a vertex $v \in V$ with $\text{outdeg}(v) = 0$ is called a* ***(local) sink*** *or* ***root*** *of the graph.*

Let $\Gamma = (V, E, f)$ be a directed multigraph. A ***directed path*** *from $v \in V$ to $v' \in V$ is a sequence $\{e_1, \ldots, e_n\}$ of edges with $t(e_i) = s(e_{i+1})$ for $i = 1, \ldots, n-1$, $v = s(e_1)$, and $v' = t(e_n)$. A directed path is called a* ***closed path*** *or a* ***cycle****, if $v = v'$. The multigraph Γ is called* ***acyclic*** *if it does not contain a cycle.*

A ***directed multigraph with ordered edges (DMGoe)*** *$\Gamma = (V, E, f, \leq)$ is a quadruple such that (V, E, f) is a directed multigraph and $(E, \leq)$ is a linearly ordered set. As subsets of E, the in-edges $E_i(v)$ and out-edges $E_o(v)$ for every vertex become linearly ordered as well.*

Definition 2.2 *A DMGoe together with a set* OP *of elementary operations and two maps* $\text{op} : V \to \text{OP}$ *and* $\text{mult} : E \to \mathbb{R}$ *is called a* ***(computational) DAG****.*

Every factorable global optimization problem can represented as a DAG together with a subset $\mathcal{O} \subseteq V$ of objective nodes and a map $b : V \to \mathbb{IR}$ of bounds. Semantically, the direction of the edges represents the computational flow, and for an edge e the scalar $\text{mult}(e)$ specifies a weight with which data passing through e is multiplied.

As a simple example consider the *factorable* optimization problem

$$\begin{aligned} \min \;& (4x_1 - x_2x_3)(x_1x_2 + x_3), \\ \text{s.t.} \;& x_1^2 + x_2^2 + x_1x_2 + x_2x_3 + x_2 = 0, \\ & \exp(x_1x_2 + x_2x_3 + x_2 + \sqrt{x_3}) \in [-1, 1], \\ & x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \in [-1, 8]. \end{aligned} \tag{1}$$

This defines the DAG depicted in Figure 1.

In some sense, this DAG is optimally small, because it contains every subexpression of the objective and constraint functions only once. Such DAGs are called **reduced**. The COCONUT Environment represents all factorable global optimization problems internally as reduced DAGs. This is in contrast to the DAGs provided by most

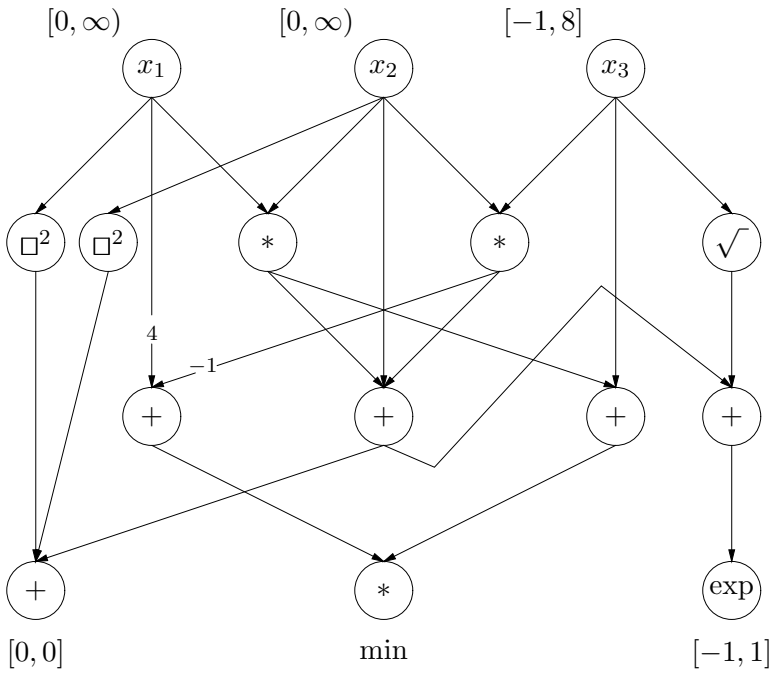

Figure 1. DAG representation of Problem (1)

parsers of high level languages like `FORTRAN 90`, `C++`, `AMPL`, or `GAMS` which usually do not detect and merge common subexpressions except for the variable nodes.

3. Evaluation

There are several types of information which the COCONUT Environment provides for functions represented as DAGs. All of them are based on a combination of forward and backward evaluation schemes as described in [6, 27, 28].

The following evaluators are implemented for the COCONUT Environment:

- symbolic first derivatives,
- function values, first to third derivatives at points,
- function ranges over boxes,
- interval gradients, Hessians, and third derivatives over boxes,
- derivatives of arbitrary order for univariate functions,
- first and second order slopes over boxes with fixed center(s),
- linear and quadratic enclosures.

Derivatives and interval derivatives are commonly used in local and global optimization. But since slopes are not so well known, we give a short definition here. For a Lipschitz continuous function $f : \mathbb{R}^n \to \mathbb{R}$ we can always write

$$f(x) - f(z) = f[z, x](x - z)$$

for any two points x and z with a suitable vector $f[z, x] \in \mathbb{R}^n$, called a **slope vector** for f. While $f[z, x]$ is not uniquely determined, except for univariate functions, we

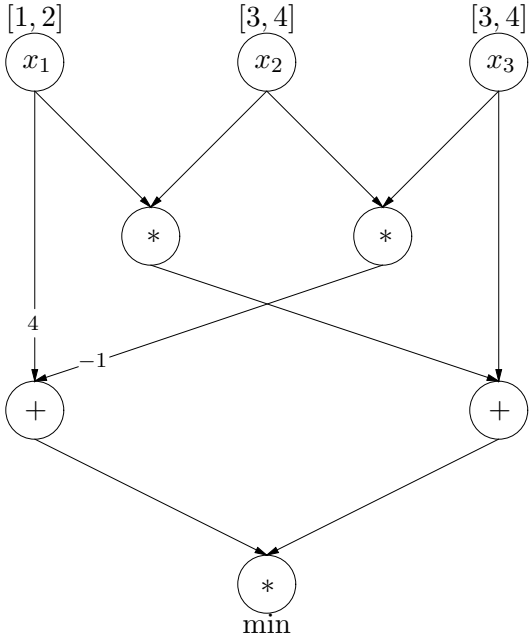

Figure 2. Directed Acyclic Graph representation of the objective function of Problem (1).

always have

$$f[z, z] = \nabla f(z). \tag{2}$$

Slopes obey a similar chain rule as derivatives, so recursive procedures to calculate $f[z, x]$ given x and z can be developed, see [38, 40, 47]. If the slope vector $f[z, x]$ is itself Lipschitz continuous we can further write for arbitrary $x, w, z \in \mathbb{R}^n$

$$f[z, x] = f[z, w] + (x - w)^T f[z, w, x] \tag{3}$$

with a **second order (bicentered) slope matrix** $f[z, w, x] \in \mathbb{R}^{n\times n}$. If $z = w$ the formula above somewhat simplifies, because of (2), to

$$f[z, x] = \nabla f(z) + (x - z)^T f[z, z, x].$$

Slopes are very useful for calculating range estimates via centered forms, e.g.

$$f(x) \in f(z) + f[z, \boldsymbol{x}](\boldsymbol{x} - z),$$
$$f(x) \in f(z) + (f[z, w] + (\boldsymbol{x} - w)^T f[z, w, \boldsymbol{x}])(\boldsymbol{x} - z)$$

for all $x \in \boldsymbol{x}$. These estimates are in general tighter than analogous ones computed by interval derivatives. In addition, slopes can be used for calculating large exclusion boxes for solving systems of equations [54]. By defining a slope of order k analogously to (3) as a slope of slopes of order $k-1$, we can get slopes of arbitrary order.

To illustrate the techniques, we will, for simplicity, focus on evaluating the objective function of Problem (1), whose DAG representation is depicted in Figure 2.

3.1. *Forward Evaluation Scheme*

The forward mode in the COCONUT Environment works by propagating data forward along the arrows in the DAG, starting from the variable nodes. The most basic forward evaluator performs a point evaluation: function values are propagated, and at every node the elementary operation is evaluated with those arguments provided by the in-edges. The result is propagated further on the out-edges.

In the COCONUT Environment, for performance reasons only scalar functions are propagated through the DAG in forward mode. All vector valued expressions are computed in backward mode. There is only one exception to that rule: there is a forward operator implemented for interval enclosures of gradients over a box.

Clearly, the effort for computing an interval gradient of an n-variate function in forward mode is worst-case roughly n times higher than computing it in backward mode. However, the backward mode depends on the chain rule

$$\frac{\partial}{\partial x_i}(f \circ g)(x) = \sum_k \frac{\partial f}{\partial g_k}(g(x)) \cdot \frac{\partial g_k}{\partial x_i}(x)$$

for $f : \mathbb{R}^m \to \mathbb{R}$, $g : \mathbb{R}^l \to \mathbb{R}^m$ and the distributive law:

$$\begin{aligned}\frac{\partial}{\partial x_i}(f \circ g \circ h)(x) &= \sum_k \frac{\partial f}{\partial g_k}(g(h(x))) \cdot \frac{\partial (g_k \circ h)}{\partial x_i}(x) \\ &= \sum_k \frac{\partial f}{\partial g_k}(g(h(x))) \cdot \sum_j \frac{\partial g_k}{\partial h_j}(h(x)) \cdot \frac{\partial h_j}{\partial x_i}(x) \\ &= \sum_{k,j} \frac{\partial f}{\partial g_k}(g(h(x))) \cdot \frac{\partial g_k}{\partial h_j}(h(x)) \cdot \frac{\partial h_j}{\partial x_i}(x).\end{aligned}$$

for $h : \mathbb{R}^n \to \mathbb{R}^l$. However, in the interval case the distributive law does not hold, but only the weaker subdistributive law

$$\boldsymbol{x}(\boldsymbol{y} + \boldsymbol{z}) \subseteq \boldsymbol{x}\boldsymbol{y} + \boldsymbol{x}\boldsymbol{z},$$

and thus for interval gradients we get two different "chain rules" for the forward and backward modes, respectively

$$\begin{aligned}\frac{\partial}{\partial x_i}(f \circ g \circ h)(\boldsymbol{x}) &= \sum_k \frac{\partial f}{\partial g_k}(g(h(\boldsymbol{x}))) \cdot \frac{\partial (g_k \circ h)}{\partial x_i}(\boldsymbol{x}) \\ &= \sum_k \frac{\partial f}{\partial g_k}(g(h(\boldsymbol{x}))) \cdot \sum_j \frac{\partial g_k}{\partial h_j}(h(\boldsymbol{x})) \cdot \frac{\partial h_j}{\partial x_i}(\boldsymbol{x}) \qquad (4) \\ &\subseteq \sum_{k,j} \frac{\partial f}{\partial g_k}(g(h(\boldsymbol{x}))) \cdot \frac{\partial g_k}{\partial h_j}(h(\boldsymbol{x})) \cdot \frac{\partial h_j}{\partial x_i}(\boldsymbol{x}). \qquad (5)\end{aligned}$$

Hence, interval gradients computed in forward mode, which makes use of the chain rule (4), usually provide tighter enclosures than those computed in backward mode, that depends on the chain rule (5). So in certain cases the higher effort needed for computing them in forward mode is justified.

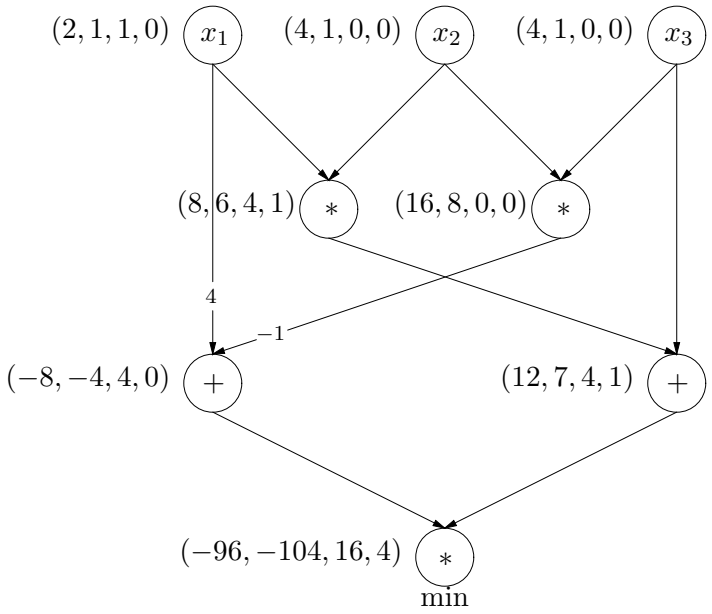

Figure 3. Hessian-two vector product evaluation for the objective in (1)

Apart from the function evaluation and the interval range evaluation by forward propagation using interval arithmetic the following scalar valued forward evaluators are provided:

(1) Evaluation of directional derivatives $\nabla f(x).v$ for arbitrary vectors v,
(2) Range enclosures of directional derivatives $\nabla f(\boldsymbol{x}).v$ for arbitrary vectors v,
(3) First order directional slopes $f[\boldsymbol{z}, \boldsymbol{x}].v$ for arbitrary vectors v,
(4) Second order directional derivatives $v^T.\nabla^2 f(x).w$ for arbitrary vectors v and w,
(5) Range enclosures of second order directional derivatives $v^T.\nabla^2 f(\boldsymbol{x}).w$ for arbitrary vectors v and w,
(6) Second order directional (bicentered) slopes $v^T.f[\boldsymbol{z}, \boldsymbol{y}, \boldsymbol{x}].w$ for arbitrary vectors v and w.

All of those evaluators compute their result with an effort which is a small multiple of one function evaluation.

As an example, we compute the expression $(1, 1, 1)^T.\nabla^2 f(2, 4, 4).e_1$ as depicted in Figure 3. The evaluator simultaneously propagates four expressions, $(f(x), \nabla f(x).v, \nabla f(x).w, v^T.\nabla^2 f(x).w)$, through the DAG. In Figure 3, we have written the values of these expressions for all nodes to the left of the circle representing them. What we find is that $f(2, 4, 4) = -96$, $\nabla f(2, 4, 4)^T(1, 1, 1) = -104$, $\nabla f(2, 4, 4)^T e_1 = 16$, and $(1, 1, 1)^T \nabla^2 f(2, 4, 4) e_1 = 4$. Note that all these results can be computed using ordinary (univariate) arithmetic on higher order differential numbers, which are implemented in the COCONUT Environment for arbitrarily high order.

In an analogous way we can compute range estimates by replacing input values of the variables with input intervals of their ranges and replacing ordinary arithmetic with interval arithmetic.

Slope evaluation runs similarly. However, it is very cumbersome and difficult to calculate slopes of optimal quality for orders higher than 2, therefore in the COCONUT Environment slopes are limited to second order.

3.2. *Backward Evaluation Scheme*

Calculating derivatives or slopes of any order could be done by the forward mode as well, but then we would need to propagate vectors, matrices or higher order tensors through the graph, and at every node we would have to perform at least one "higher dimensional" addition, so the effort to calculate a derivative or slope of order p would be n^p times the effort of calculating a function value, if the function is n-variate.

However, it is well known from automatic differentiation [27] that the number of operations can be reduced by one factor of n by reversing the direction of evaluation.

The COCONUT Environment provides the following operators in backward mode. They depend on data generated during the forward pass of some of the evaluators described in Section 3.1.

(1) Evaluation of gradients $\nabla f(x)$,
(2) Range enclosures of gradients $\nabla f(\boldsymbol{x})$,
(3) First order slopes $f[\boldsymbol{z}, \boldsymbol{x}]$,
(4) Hessian–vector products $\nabla^2 f(x).v$ for arbitrary vectors v,
(5) Range enclosures of Hessian–vector products $\nabla^2 f(\boldsymbol{x}).v$ for arbitrary vectors v,
(6) Second order slope–vector products $f[\boldsymbol{z}, \boldsymbol{y}, \boldsymbol{x}].v$ for arbitrary vectors v,
(7) Third order derivative–vector products of the form $\sum_{i,j} \nabla^3_{ijk} f(x) v_i w_j$ for arbitrary vectors v and w,
(8) Range enclosures of third order derivative–vector products of the form $\sum_{i,j} \nabla^3_{ijk} f(\boldsymbol{x}) v_i w_j$ for arbitrary vectors v and w.

All of those evaluators compute their results with an effort which is a small multiple of one function evaluation.

As an example, we will evaluate the third derivative of f at $(2, 4, 4)$ multiplied by the vectors $(1, 1, 1)$ and e_1. For that during the forward path we perform the calculation of $v^T.\nabla^2 f(x).w$ as described in Section 3.1. At the in-edge of f with index k we store a four dimensional vector ℓ_k which contains the information

$$\ell_k = \begin{pmatrix} \displaystyle\sum_{l,m} \frac{\partial^3 f}{\partial g_l \partial g_m \partial g_k}(g)\ \nabla g_l(x).v\ \nabla g_m(x).w + \sum_l \frac{\partial^2 f}{\partial g_l \partial g_k}(g)\ v^T.\nabla^2 g_l(x).w \\ \displaystyle\sum_l \frac{\partial^2 f}{\partial g_l \partial g_k}(g)\ \nabla g_l(x).w \\ \displaystyle\sum_l \frac{\partial^2 f}{\partial g_l \partial g_k}(g)\ \nabla g_l(x).v \\ \displaystyle\frac{\partial f}{\partial g_k}(g) \end{pmatrix},$$

where the vector g denotes all arguments on that f depends. Note that the f–term

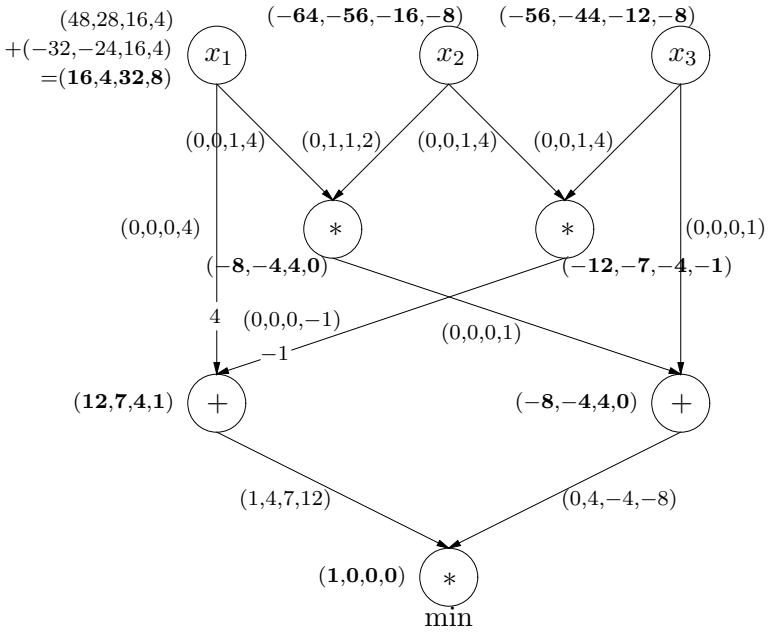

Figure 4. Third derivative evaluation of the objective in (1)

in every formula is local to the node, since it is only a partial derivative with respect to its input variables. The ∇g_i–terms are scalar and computed during the forward pass. During the backward process we accumulate another four dimensional vector φ calculating $(\nabla f, \nabla^2 f.v, \nabla^2 f.w, v^T.\nabla^3 f.w)$.

The update formula for backward propagation along edge k is then $\varphi' = (\ell_{k,4}\varphi_1, \ell_{k,3}\varphi_1 + \ell_{k,4}\varphi_3, \ell_{k,2}\varphi_1 + \ell_{k,4}\varphi_2, \ell_k^T\varphi)$, where φ' denotes the updated vector after proceeding backward along the edge. As usual in backward evaluation schemes, on a node N all the results along all out-edges of N are summed.

The resulting vectors ℓ (at the edges) and φ (in bold face at the nodes) for the example can be found in Figure 4. At the end, we can read off the result from the variable nodes and get $\nabla f(2,4,4) = (16,-64,-56)^T$, $\nabla^2 f(2,4,4).(1,1,1)^T = (4,-56,-44)^T$, $\nabla^2 f(2,4,4).e_1 = (32,-16,-12)^T$, and $(1,1,1)^T.\nabla^3 f(2,4,4).e_1 = (8,-8,-8)^T$. The effort of the backward process is about nine backward gradient evaluations.

There is hardly any difference in computing interval derivatives or slopes. The advantage of this approach for computing higher derivatives over interpolation approaches [10] is that they can be carried over to interval enclosures without additional wrapping. For second and third derivatives graph-coloring like techniques [24] can be used to effectively compute the whole Hessian and tensor of third derivatives in the sparse case.

4. Constraint Propagation on DAGs

As already mentioned, one strength of the DAG concept for global optimization is that knowledge of feasible points and the constraints can be used to narrow the ranges of the variables, cf. [4, 49, 57].

The COCONUT Environment implements various methods for interval constraint

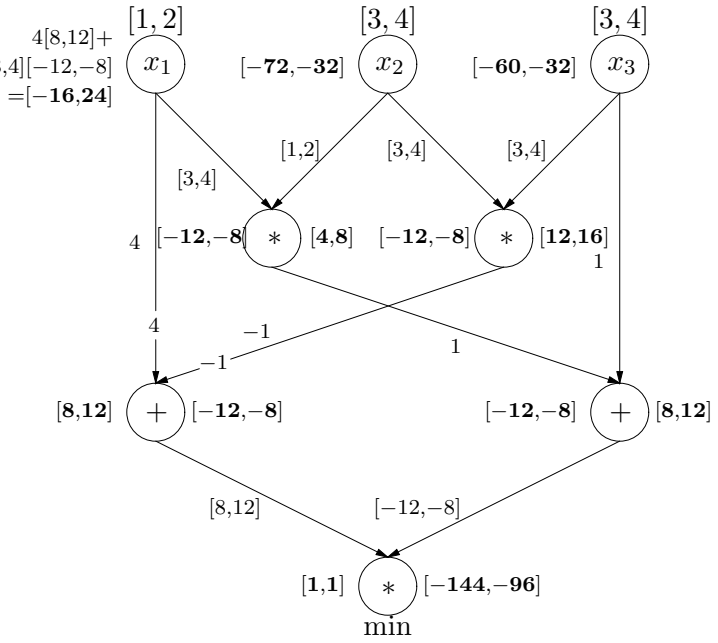

Figure 5. Interval gradient evaluation for the objective in (1) after constraint propagation.

propagation, most prominently HC4 [3, 25] and FBPD [58]. A specialty of FBPD is that it also provides bounds on the intermediate nodes, and those can be used to further tighten the range estimates for derivatives during back propagation, as explained in [53].

For the discussion, we will review the improvement of the interval gradient of the example function f on the box $\boldsymbol{x} = [1,2] \times [3,4] \times [3,4]$ after constraint propagation of the constraint $f(x) \leq \tilde{f} = -144$ (see Figure 5). Without using the intermediate node information of constraint propagation, the result is $f'(\boldsymbol{x}) \subseteq ([-24,45];[-72,-19];[-60,-19])^T$. This enclosure significantly reduces to $f'(\boldsymbol{x}) \subseteq ([-16,24];[-72,-32];[-60,-32])^T$ if the intermediate node ranges of the constraint propagator (displayed in bold face on the right of each intermediate node) are used.

If these tightened enclosures are used during, e.g., a global optimization algorithm, care must be taken to avoid various pitfalls.

Generally speaking, the interval gradient computed in Figure 5 is an enclosure $\boldsymbol{g}$ of the set

$$M := \{\nabla f(x) \mid x \in \boldsymbol{x} \wedge f(x) \leq \tilde{f}\}.$$

This improved interval gradient can safely be used, e.g., for the monotonicity test. This test checks for the objective function f of a bound constrained optimization problem and a subbox $\boldsymbol{x}$ of the feasible domain whether $0 \notin \nabla_i f(\boldsymbol{x})$ for some i, to conclude that $\boldsymbol{x}$ does not contain the global optimum (except possibly on the border of the ith coordinate), because of the first order necessary optimality conditions. If the enclosure $\boldsymbol{g}$ is used instead of $\nabla f(\boldsymbol{x})$, this remains true, because a global optimum x^*, unless it is on the border, must satisfy $f(x^*) \leq \tilde{f}$ and $\nabla f(x^*) = 0$. Hence, for the monotonicity test we can safely conclude that f does not have a global minimum in the interior of $\boldsymbol{x}$, provided $0 \notin \boldsymbol{g}_i$ for some i.

On the other hand, interval gradients are also used for range estimation via centered forms:

$$f(\boldsymbol{x}) \subseteq f(z) + f'(\boldsymbol{x})(\boldsymbol{x} - z), \tag{6}$$

where $z \in \boldsymbol{x}$ is usually chosen as the center of the box $\boldsymbol{x}$. A centered form can be also used for enclosing f over an arbitrary set S, but it is in general valid only if S is star-shaped with center z. Unfortunately, M is not always star shaped with center z, even if $z \in \boldsymbol{x}$. That can, e.g., be derived from Figure 5 for the point $(1, 3, 3)$. On the lower right + operator this point evaluates to 6, which is clearly outside the range $[8, 12]$. Hence, $(1, 3, 3) \notin M$, which cannot easily be seen without propagating through the DAG. So if CP-improved interval gradients are intended to be used in centered forms, care must be taken that the forward propagation of the center through the DAG on every node gives a result which is inside the CP-improved node range.

For slopes this problem does not appear since the slope form

$$f(\boldsymbol{x}) \subseteq f(z) + f[z, \boldsymbol{x}](\boldsymbol{x} - z) \tag{7}$$

remains true, even if $z \notin M$.

Of course, the higher order derivatives and slopes can be used to generate higher order centered forms for function estimation, and the COCONUT Environment provides a module for those:

$$\begin{aligned}
f(\boldsymbol{x}) &\subseteq f(z) + (f'(z) + \tfrac{1}{2}(\boldsymbol{x} - z)^T \nabla^2 f(\boldsymbol{x}))(\boldsymbol{x} - z) \\
f(\boldsymbol{x}) &\subseteq f(z) + (f[z, w] + (\boldsymbol{x} - w)^T f[z, w, \boldsymbol{x}])(\boldsymbol{x} - z) \\
f(\boldsymbol{x}) &\subseteq f(z) + \big(f'(z) + \tfrac{1}{2}(\boldsymbol{x} - z)^T \big(\nabla^2 f(z) + \tfrac{1}{3}\nabla^3 f(\boldsymbol{x})(\boldsymbol{x} - z)\big)\big)(\boldsymbol{x} - z)
\end{aligned}$$

They have approximation properties of order 3 or 4, respectively, and are therefore useful for reducing the cluster effect close to the optimizers [34]. Of course, for all centered forms involving an interval derivative, the same discussion as in the paragraph above applies to the center.

5. Linear and quadratic enclosures

As another application of higher order interval derivatives and slopes we consider the general nonlinear global optimization problem

$$\begin{aligned}
\min \;& f(x) \\
\text{s.t.} \;& F(x) \in \boldsymbol{F} \\
& x \in \boldsymbol{x},
\end{aligned} \tag{8}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $F : \mathbb{R}^n \to \mathbb{R}^m$.

The linear approximations (6) and (7) of a function f provided by interval gradients or slopes can be used to construct an enclosure of f by linear functions. This in turn can be used to construct a linear relaxation of (8).

We cite the following proposition from [53].

PROPOSITION 5.1 *Let $\boldsymbol{s}$ be a first order interval slope $f[z, \boldsymbol{x}]$ or an interval gradient $\nabla f(\boldsymbol{x})$ of the function $f : \mathbb{R}^n \to \mathbb{R}$ over the box $\boldsymbol{x}$. If $z \in \boldsymbol{x}$ then the function*

$$\underline{f}(x) = \underline{f} + \sum_{i=1}^{n} \overline{s}_i(\underline{x}_i - z_i) + \frac{\underline{s}_i(\overline{x}_i - z_i) - \overline{s}_i(\underline{x}_i - z_i)}{\overline{x}_i - \underline{x}_i}(x_i - \underline{x}_i)$$

is a linear function which underestimates f on $\boldsymbol{x}$, i.e.,

$$\underline{f}(x) \leq f(x) \quad \text{for all } x \in \boldsymbol{x},$$

and the function

$$\overline{f}(x) = \overline{f} + \sum_{i=1}^{n} \underline{s}_i(\underline{x}_i - z_i) + \frac{\overline{s}_i(\overline{x}_i - z_i) - \underline{s}_i(\underline{x}_i - z_i)}{\overline{x}_i - \underline{x}_i}(x_i - \underline{x}_i)$$

is a linear overestimating function for f over $\boldsymbol{x}$, where $f(z) \in [\underline{f}, \overline{f}]$.

Using this result, by enclosing all functions in pairs of linear over- and underestimators, a linear relaxation can be computed for the optimization problem. Note that this linear relaxation is different from reformulation linearization as, e.g., computed in BARON [48], since it is of the same dimension as the original problem.

Similarly, as described in [51], quadratic underestimation functions $\underline{q}$ and quadratic overestimation functions $\overline{q}$ can be constructed if enclosures of $f(z)$ and $f'(z)$ and either an interval Hessian $\nabla^2 f(\boldsymbol{x})$ or a second order slope of the form $f[z, z, \boldsymbol{x}]$ are available.

PROPOSITION 5.2 *Let $\boldsymbol{H} := f[z, w, \boldsymbol{x}]$ be a second order slope of the function*

$f : \mathbb{R}^n \to \mathbb{R}$ or an interval Hessian. If $z, w \in \boldsymbol{x}$ then the function

$$\begin{aligned}
\underline{q}(x) = \underline{f} + \sum_{i=1}^{n}\sum_{j=1}^{n} - \Bigg(& 2\frac{(\overline{H}_{ij} - \underline{H}_{ij})(\overline{x}_j - w_j)(w_j - \underline{x}_j)}{\overline{x}_j - \underline{x}_j} + (\overline{g}_j - \underline{g}_j)\Bigg) \\
& \cdot \frac{(\overline{x}_i - z_i)(z_i - \underline{x}_i)}{\overline{x}_i - \underline{x}_i} \\
& + \Bigg(\frac{(\overline{H}_{ij} - \underline{H}_{ij})(\overline{x}_j - w_j)(w_j - \underline{x}_j)(2z_i - \overline{x}_i - \underline{x}_i)}{\overline{x}_j - \underline{x}_j} \\
& \quad + \underline{g}_j(\overline{x}_i - z_i) + \overline{g}_j(z_i - \underline{x}_i)\Bigg) \frac{x_i - z_i}{\overline{x}_i - \underline{x}_i} \\
& + \frac{(\overline{H}_{ij} - \underline{H}_{ij})(\overline{x}_j + \underline{x}_j - 2w_j)(\overline{x}_i - z_i)(z_i - \underline{x}_i)}{(\overline{x}_i - \underline{x}_i)(\overline{x}_j - \underline{x}_j)}(x_j - w_j) \\
& + \Bigg(\frac{(\overline{H}_{ij} - \underline{H}_{ij})(w_j(\underline{x}_i + \overline{x}_i) + z_i(\underline{x}_j + \overline{x}_j) - 2z_i w_j)}{(\overline{x}_i - \underline{x}_i)(\overline{x}_j - \underline{x}_j)} \\
& \quad + \frac{-\overline{H}_{ij}(\overline{x}_i\underline{x}_j + \underline{x}_i\overline{x}_j) + \underline{H}_{ij}(\underline{x}_i\underline{x}_j + \overline{x}_i\overline{x}_j)}{(\overline{x}_i - \underline{x}_i)(\overline{x}_j - \underline{x}_j)}\Bigg)(x_i - z_i)(x_j - w_j)
\end{aligned}$$

is a quadratic function which underestimates f on $\boldsymbol{x}$, i.e.,

$$\underline{q}(x) \le f(x) \quad \textit{for all } x \in \boldsymbol{x},$$

and the function

$$\begin{aligned}
\overline{q}(x) = \overline{f} + \sum_{i=1}^{n}\sum_{j=1}^{n} \Bigg(& 2\frac{(\overline{H}_{ij} - \underline{H}_{ij})(\overline{x}_j - w_j)(w_j - \underline{x}_j)}{\overline{x}_j - \underline{x}_j} + (\overline{g}_j - \underline{g}_j)\Bigg) \\
& \cdot \frac{(\overline{x}_i - z_i)(z_i - \underline{x}_i)}{\overline{x}_i - \underline{x}_i} \\
& - \Bigg(\frac{(\overline{H}_{ij} - \underline{H}_{ij})(\overline{x}_j - w_j)(w_j - \underline{x}_j)(2z_i - \overline{x}_i - \underline{x}_i)}{\overline{x}_j - \underline{x}_j} \\
& \quad - \overline{g}_j(\overline{x}_i - z_i) - \underline{g}_j(z_i - \underline{x}_i)\Bigg) \frac{x_i - z_i}{\overline{x}_i - \underline{x}_i} \\
& + \frac{(\overline{H}_{ij} - \underline{H}_{ij})(\overline{x}_j + \underline{x}_j - 2w_j)(\overline{x}_i - z_i)(z_i - \underline{x}_i)}{(\overline{x}_i - \underline{x}_i)(\overline{x}_j - \underline{x}_j)}(x_j - w_j) \\
& + \Bigg(\frac{(\overline{H}_{ij} - \underline{H}_{ij})(w_j(\underline{x}_i + \overline{x}_i) + z_i(\underline{x}_j + \overline{x}_j) - 2z_i w_j)}{(\overline{x}_i - \underline{x}_i)(\overline{x}_j - \underline{x}_j)} \\
& \quad + \frac{\overline{H}_{ij}(\overline{x}_i\overline{x}_j + \underline{x}_i\underline{x}_j) - \underline{H}_{ij}(\overline{x}_i\underline{x}_j + \underline{x}_i\overline{x}_j)}{(\overline{x}_i - \underline{x}_i)(\overline{x}_j - \underline{x}_j)}\Bigg)(x_i - z_i)(x_j - w_j)
\end{aligned}$$

is a quadratic overestimating function for f over $\boldsymbol{x}$, where $f(z) \in [\underline{f}, \overline{f}]$ and $f[z, w] \in [\underline{g}, \overline{g}]$.

If we then consider the constraints for problem (8) componentwise, for every component $F_j(x) \in \boldsymbol{F}_j$ the constraints $\underline{Q}_j(x) \le \overline{F}_j$ and $\overline{Q}_j(x) \ge \underline{F}_j$ are valid quadratic constraints; here $\underline{Q}_j(x)$ and $\overline{Q}_j(x)$ are the quadratic under- and overestimating functions for $F_j(x)$. Together with the quadratic underestimating function $\underline{q}(x)$ of the objective function f, we get the QCQP (quadratically constrained quadratic program) relaxation

$$\begin{aligned} \min\ & \underline{q}(x) \\ \text{s.t.}\ & \underline{Q}(x) \le \overline{F} \\ & \overline{Q}(x) \ge \underline{F} \\ & x \in \boldsymbol{x} \end{aligned}$$

of (8), where $\underline{Q}(x)$ and $\overline{Q}(x)$ denote the vector of all underestimating and overestimating functions $\underline{Q}_j$ and $\overline{Q}_j$, respectively, for all components Q_j. The QCQP is different from quadratic relaxations computed in αBB [2], because no explicit decomposition into a difference of convex functions is used before computing the quadratic underestimators. The QCQPs are also not convex in general. However, there are specialized methods for nonconvex QCQPs, see [17–20].

If no QCQP solver is available, alternatively the constraint functions F_j can be linearly relaxed by $\underline{F}_j$ and $\overline{F}_j$ leading to the QP relaxation

$$\begin{aligned} \min\ & \underline{q}(x) \\ \text{s.t.}\ & \underline{F}(x) \le \overline{F} \\ & \overline{F}(x) \ge \underline{F} \\ & x \in \boldsymbol{x}, \end{aligned}$$

which captures the behaviour of the problem close to a local minimum z better than the purely linear relaxation constructed by Proposition 5.1, if the local minimum is chosen as the center. This is because the constant term of the linear relaxation is of order $\operatorname{rad} \boldsymbol{x}$ whereas the constant term of the quadratic relaxation is of order $(\operatorname{rad} \boldsymbol{x})^2$, so at z the overestimation is smaller for small boxes. If z is in a corner of $\boldsymbol{x}$ the constant term vanishes completely. The overall deviation $|q(x) - f(x)|$ is also $O(\operatorname{rad} \boldsymbol{x})$ for linear and $O((\operatorname{rad} \boldsymbol{x})^2)$ for the quadratic relaxation, so for small boxes $\boldsymbol{x}$ the quadratic relaxation indeed captures the behaviour of the problem better than the linear relaxation.

6. Conclusion

The COCONUT Environment for global optimization provides many inference modules, which can be used for building solution strategies for global optimization solvers, e.g. `cocos` [21] and `coco_gop_ex` [41]. Many of these modules heavily rely on automatic differentiation techniques, that would not have been possible without the major contributions to this field by Andreas Griewank.

References

[1] *ILOG Solver 5.1*, 2001.
[2] I.P. Androulakis, C.D. Maranas, and C.A. Floudas. αBB: a global optimization method for general constrained nonconvex problems. *J. Global Optim.*, 7:337–363, 1995.
[3] F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. Revising Hull and Box Consistency. In: *Proceedings of the International Conference on Logic Programming (ICLP'99)*, 230–244, 1999.
[4] F. Benhamou and W. Older. Applying interval arithmetic to real, integer, and boolean constraints. *Journal of Logic Programming*, 1997.
[5] M. Berz. COSY INFINITY version 8 reference manual. Technical report, National Superconducting Cyclotron Lab., Michigan State University, East Lansing, Mich., 1997. MSUCL–1008.
[6] M. Berz, C. Bischof, G. Corliss, and A. Griewank. Computational Differentiation. SIAM Publications, Philadelphia, 1996.
[7] M. Berz and K. Makino. Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. *Reliable Computing*, 4:361–369, 1998.
[8] C. Bischof, A. Carle, G. Corliss, A. Griewank, and P. Hovland. ADIFOR–generating derivative codes from Fortran programs. *Scientific Programming*, 1(1):11–29, 1992.
[9] C. Bischof, P. Khademi, A. Mauer, and A. Carle. ADIFOR 2.0: Automatic differentiation of Fortran 77 programs. *Computational Science & Engineering, IEEE*, 3(3):18–32, 2002.
[10] C. Bischof, G. Corliss, and A. Griewank. Structured second-and higher-order derivatives through univariate Taylor series. *Optimization Methods and Software*, 2(3&4):211–232, 1993.
[11] C. Bliek. *Computer methods for design automation*. PhD thesis, Dept. of Ocean Engineering, Massachusetts Institute of Technology, 1992.
[12] C. Bliek, P. Spellucci, L.N. Vicente, A. Neumaier, L. Granvilliers, E. Monfroy, F. Benhamouand, E. Huens, P. Van Hentenryck, D. Sam-Haroud, and B. Faltings. Algorithms for Solving Nonlinear Constrained and Optimization Problems: The State of the Art. Report of the European Community funded project COCONUT, Mathematisches Institut der Universität Wien, http://www.mat.univie.ac.at/~neum/glopt/coconut/StArt.html, 2001.
[13] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS — A User's Guide (Release 2.25)*. Boyd & Fraser Publishing Company, Danvers, Massachusetts, 1992.
[14] C. Chekuri, R. Johnson, R. Motwani, B. Natarajan, B. Ramakrishna Rau, and M. S. Schlansker. Profile-driven instruction level parallel scheduling with application to super blocks. In *International Symposium on Microarchitecture*, pages 58–67, 1996.
[15] S. Dallwig, A. Neumaier, and H. Schichl. GLOPT — A Program for Constrained Global Optimization. In I. M. Bomze, T. Csendes, R. Horst, and P.M. Pardalos, editors, *Developments in Global Optimization*, pages 19–36. Kluwer, Dordrecht, 1997.
[16] L.C.W. Dixon and G.P. Szegö. *Towards global optimization*. Elsevier, New York, 1975.
[17] F. Domes. Rigorous techniques for continuous constraint satisfaction problems. PhD Thesis, Universität Wien, 2010
[18] F. Domes and A. Neumaier. Quadratic constraint propagation, *Constraints*, 15(3):404-429, 2010.
[19] F. Domes and A. Neumaier. Rigorous enclosures of ellipsoids and Directed Cholesky factorizations. Manuscript, 2010. `http://www.mat.univie.ac.at/~dferi/research/Cholesky.pdf`.
[20] F. Domes and A. Neumaier. Linear methods for quadratic constraint satisfaction problems. Manuscript, 2010. `http://www.mat.univie.ac.at/~dferi/research/Linear.pdf`
[21] F. Domes, M. Fuchs, and H. Schichl. The Optimization Test Environment. To appear in Optimization Methods and Software, 2010.
[22] C.A. Floudas, Deterministic Global Optimization: Theory, Algorithms and Applications, Kluwer, Dordrecht 1999.
[23] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL — A Mathematical Programming Language*. Thomson, second edition, 2003.
[24] A.H. Gebremedhin, F. Manne, and A. Pothen. What Color Is Your Jacobian? Graph Coloring For Computing Derivatives, *SIAM Rev.*, 47:629–705, 2005.
[25] L. Granvilliers, F. Goualard, and F. Benhamou. Box Consistency through Weak Box Consistency. In: *Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'99)* 373–380, 1999.
[26] A. Griewank, D. Juedes, and J. Utke. Algorithm 755: ADOL-C: a package for the automatic differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software (TOMS)*, 22(2):131–167, 1996.
[27] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. 2nd Edition. SIAM Publications, Philadelphia, 2008.
[28] A. Griewank and G.F. Corliss. *Automatic Differentiation of Algorithms*. SIAM Publications, Philadelphia, 1991.
[29] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz. *C++ Toolbox for Verified Computing I, Basic Numerical Problems: Theory, Algorithms, and Programs*, Springer, 1995.
[30] R. Horst. A General Class of Branch-and-Bound Methods in Global Optimization with Some New Approaches for Concave Minimization. *Journal of Optimization Theory and Applications*, 51:271–291, 1986.
[31] R. Horst. Deterministic Global Optimization with Partition Sets Whose Feasibility Is Not Known: Application to Concave Minimization, Reverse Convex Constraints, DC-Programming, and Lipschitzian Optimization. *Journal of Optimization Theory and Applications*, 58:11–37, 1988.
[32] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.
[33] R.B. Kearfott. Decomposition of Arithmetic Expressions to Improve the Behavior of Interval Iteration for Nonlinear Systems, *Computing*, 47:169–191, 1991.
[34] R.B. Kearfott and K. Du. The cluster problem in multivariate global optimization. *J. Global Opti-*

mization, 5:253–265, 1994.
[35] R.B. Kearfott. *Rigorous Global Search: Continuous Problems*. Kluwer, Dordrecht, 1996.
[36] M. Kieffer, M.C. Markót, H. Schichl, and E. Walter, Verified global optimization for estimating the parameters of nonlinear models. Submitted. `http://www.mat.univie.ac.at/~herman/papers/Guar_Optim_2010_v4.pdf`
[37] O. Knüppel PROFIL/BIASa fast interval library. *Computing*, 53(3):277–287, 1994.
[38] L.V. Kolev. Use of interval slopes for the irrational part of factorable functions. *Reliable Computing*, 3:83–93, 1997.
[39] L.V. Kolev An improved interval linearization for solving non-linear problems, Manuscript (2002)
[40] R. Krawczyk and A. Neumaier, Interval slopes for rational functions and associated centered forms, *SIAM J. Numer. Anal.* 22:604–616, 1985.
[41] M.C. Markót and H. Schichl. Bound constrained optimization in the COCONUT Environment. Manuscript, 2010.
[42] M.C. Markót and H. Schichl. Comparison and automated selection of local optimization solvers for interval global optimization methods. Submitted, 2010. `http://www.mat.univie.ac.at/~markot/loptcomp.pdf`
[43] G.P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems *Math. Programming*, 10:147–175, 1976.
[44] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge Univ. Press, Cambridge, 1990.
[45] A. Neumaier. *Introduction to Numerical Analysis*. Cambridge Univ. Press, Cambridge, 2001.
[46] A. Neumaier. Taylor forms — use and limits. *Reliable Computing*, 9:43–79, 2002.
[47] S.M. Rump. Expansion and estimation of the range of nonlinear functions, *Math. Comp.* 65:1503–1512, 1996.
[48] N.V. Sahinidis. BARON: A general purpose global optimization software package. *J. Global Optim.*, 8:201–205, 1996.
[49] D. Sam-Haroud and B. Faltings. Consistency techniques for continuous constraints. *Constraints*, 1(1&2):85–118, 1996.
[50] H. Schichl. Global Optimization in the COCONUT project. In *Proceedings of the Dagstuhl Seminar "Numerical Software with Result Verification"*, Springer Lecture Notes in Computer Science, page 9, 2003.
[51] H. Schichl and M.C. Markót. Interval analysis on directed acyclic graphs for global optimization. Higher order methods. Manuscript, 2010.
[52] H. Schichl and M.C. Markót. Exclusion regions for optimization problems. Manuscript, 2010.
[53] H. Schichl and A. Neumaier. Interval analysis on directed acyclic graphs for global optimization. *Journal of Global Optimization*, 33(4):541–562, 2006.
[54] H. Schichl and A. Neumaier. Exclusion regions for systems of equations. *SIAM J. Numer. Anal.*, 42:383–408, 2004.
[55] Z. Shen and A. Neumaier. The Krawczyk operator and Kantorovich's theorem. *J. Math. Anal. Appl.*, 149:437–443, 1990.
[56] M. Tawarmalani and N.V. Sahinidis, Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications, Kluwer, Dordrecht 2002.
[57] P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica — A Modeling Language for Global Optimization*. MIT Press, Cambridge, MA, 1997.
[58] X.-H. Vu, H. Schichl and D. Sam-Haroud. Using Directed Acyclic Graphs to Coordinate Propagation and Search for Numerical Constraint Satisfaction Problems. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*. Florida, USA, 2004.
[59] The COCONUT Environment. `www.mat.univie.ac.at/coconut-environment`