RWTH Aachen

Department of Computer Science
Technical Report

The Complexity of Derivative Computation

Uwe Naumann

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2005-15

RWTH Aachen · Department of Computer Science · August 2005

The publications of the Department of Computer Science of *RWTH Aachen University* are in general accessible through the World Wide Web.

http://aib.informatik.rwth-aachen.de/

The Complexity of Derivative Computation

Uwe Naumann

LuFG Software and Tools for Computational Engineering, Department of Computer Science
RWTH Aachen University, 52056 Aachen, Germany
WWW: `http://www.stce.rwth-aachen.de`, Email: `naumann@stce.rwth-aachen.de`

Abstract. We show that the problem of accumulating Jacobian matrices by using a minimal number of floating-point operations is NP-complete by reduction from ENSEMBLE COMPUTATION. The proof makes use of the fact that, deviating from the state-of-the-art assumption, algebraic dependences can exist between the local partial derivatives. It follows immediately that the same problem for directional derivatives, adjoints, and higher derivatives is NP-complete, too.

1 Context

We consider the automatic differentiation (AD) [10] of an implementation of a non-linear vector function

$$\mathbf{y} = F(\mathbf{x}, \mathbf{a}), \quad F : I\!R^{n+\tilde{n}} \supseteq D \to I\!R^m \quad , \tag{1}$$

as a computer program.[1] With the Jacobian matrix F' of F defined in Equation (6) *tangent-linear*

$$\dot{\mathbf{y}} = \dot{F}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{a}) \equiv F'(\mathbf{x}, \mathbf{a}) * \dot{\mathbf{x}}, \quad \dot{\mathbf{x}} \in I\!R^n \quad , \tag{2}$$

and *adjoint*

$$\bar{\mathbf{x}} = \bar{F}(\mathbf{x}, \bar{\mathbf{y}}, \mathbf{a}) \equiv \left(F'(\mathbf{x}, \mathbf{a})\right)^T * \bar{\mathbf{y}}, \quad \bar{\mathbf{y}} \in I\!R^m \quad , \tag{3}$$

versions of numerical simulation programs with potentially complicated intra- and interprocedural flow of control can be generated automatically by AD tools [9, 11, 15, 25]. This technique has been proved extremely useful in the context of numerous applications of computational science and engineering requiring numerical methods that are based on derivative information [3, 5–7]. For the purpose of this paper we may assume trivial flow of control in the form of a straight-line program. Similarly, one may consider the evaluation of an arbitrary function at a given point to fix the flow of control.
Our interest lies in the computation of the Jacobian of the *active* outputs (or *dependent* variables) $\mathbf{y} = (y_j)_{j=1,\ldots,m}$ with respect to the *active* inputs (or *independent* variables) $\mathbf{x} = (x_i)_{i=1,\ldots,n}$. The $\tilde{n}-$vector $\mathbf{a}$ contains all *passive* inputs. Conceptually, AD decomposes the program into a sequence of scalar assignments

$$v_j = \varphi_j(v_i)_{i \prec j} \tag{4}$$

for $j = 1, \ldots, p+m$. where we follow the notation in [10]. We refer to Equation (4) as the *code list* of F, and we set $x_i = v_{i-n}$ for $i = 1, \ldots, n$ and $y_j = v_{p+j}$ for $j = 1, \ldots, m$. The v_j, $j = 1, \ldots, p$, are referred to as *intermediate* variables. The

[1] F is used to refer to the given implementation.

notation $i \prec j$ marks a direct dependence of v_j on v_i meaning that v_i is an argument of the *elemental function*[2] φ_j. The code list induces a directed acyclic graph $G = (V, E)$ such that $V = \{v_{1-n}, \ldots, v_{p+m}\}$ and $(i, j) \in E \Leftrightarrow i \prec j$. Assuming that all elemental functions are continuously differentiable at their respective arguments all local partial derivatives can be computed by a single evaluation of the *linearized* code list

$$\left.\begin{aligned} c_{j,i} &= \frac{\partial \varphi_j}{\partial v_i}(v_k)_{k \prec j} \quad \forall i \prec j \\ v_j &= \varphi_j(v_i)_{i \prec j} \end{aligned}\right. \quad \text{for } j = 1, \ldots, p+m \tag{5}$$

for given values of $\mathbf{x}$ and $\mathbf{a}$. The corresponding linearized computational graph is obtained by attaching the $c_{j,i}$ to the corresponding edges (i, j). An example is shown in Figure 1.

It has been well-known for some time [2] that the entries of the Jacobian

$$F'(\mathbf{x}, \mathbf{a}) \equiv (f_{j,i}) = \left(\frac{\partial y_j}{\partial x_i}\right) = \left(\frac{\partial v_{p+j}}{\partial v_{i-n}}\right), \quad i = 1, \ldots, n, \; j = 1, \ldots, m, \tag{6}$$

can be computed as

$$f_{j,i} = \sum_{[i \to j]} \prod_{(k,l) \in [i \to j]} c_{l,k} \quad , \tag{7}$$

where $[i \to j]$ denotes a path from i to j in G, that is, a sequence of edges $((k_\nu, l_\nu), \nu = 1, \ldots, |P_{i,j}|)$ such that $k_1 = i$, $l_{|P_{i,j}|} = j$, and $l_\nu = k_{\nu+1}$ for $\nu = 1, \ldots, |P_{i,j}| - 1$. We set $P_{i,j} \equiv \{(k, l) \in [i \to j]\}$. Throughout this paper we use the notation $|S|$ for the cardinality of a set S. Moreover, the theoretical investigation of the potential decrease of the number of floating-point operations required for the evaluation of Equation (7) has been the subject of a number of papers [13, 14, 18, 20]. In all cases the authors consider elimination problems in versions of the computational graph. The $c_{j,i}$ are assumed to be algebraically independent. Linear parts of the program may represent an exception as the values of the local partial derivatives are invariant with respect to the inputs of F. Hence they can potentially be eliminated statically (at compile-time) as described in [21].[3]

[2] Elemental functions are the arithmetic operators and intrinsic functions provided by the programming language.

[3] In AD one is primarily interested in a low dynamic (run-time) cost of the automatically generated derivative code. Similar to classical compiler techniques the cost of compile-time manipulations is not taken into account. This approach assumes that the cost of the compilation is still within reasonable bounds, that is, it must be much lower than the cumulative computational cost of all executions of the generated code. This requirement is satisfied by the static techniques proposed in [21].

Example 1 *Consider the following linearized code list of a vector function* $F : I\!R^4 \to I\!R^4$:

$$
\begin{array}{lll}
c_{1,-3} = v_{-2}; & c_{1,-2} = v_{-3}; & v_1 = v_{-3} * v_{-2} \\
c_{2,-1} = v_0; & c_{2,0} = v_{-1}; & v_2 = v_{-1} * v_0 \\
c_{3,1} = 1/v_2; & c_{3,2} = -v_1/v_2^2; & v_3 = v_1/v_2 \\
c_{4,1} = v_3; & c_{4,3} = v_1; & v_4 = v_1 * v_3 \\
c_{5,3} = \cos(v_3); & & v_5 = \sin(v_3) \\
c_{6,3} = 3.14; & & v_6 = 3.14 * v_3 \\
c_{7,2} = v_3; & c_{7,3} = v_2; & v_7 = v_2 * v_3
\end{array}
$$

The computation of the values of the code list variables $v_1, \ldots, v_7$ *is preceded by the computation of the corresponding local partial derivatives. The linearized computational graph is shown in Figure 1. The fact that* $c_{4,1} = c_{7,2} = v_3$, *contradicts the assumption about the algebraic independence of the partial derivatives. Nevertheless, to our knowledge such dependences have been ignored in previous work on the subject.*

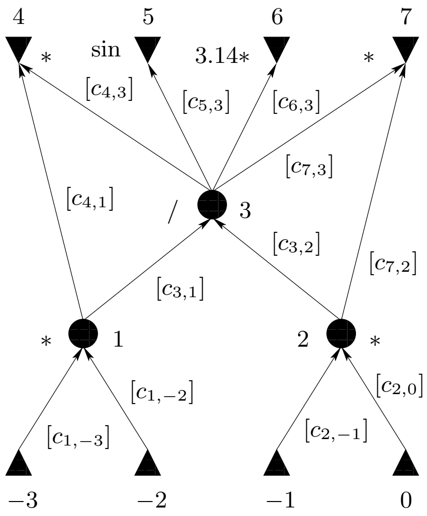

Fig. 1. *Linearized Computational Graph G:* Independent and dependent variables are represented by black triangles pointing upward $(-3, \ldots, 0)$ and downward $(4, \ldots, 7)$, respectively. Intermediate and maximal vertices are marked with their respective elemental functions. The corresponding local partial derivatives that are associated with all edges in G are enclosed within square brackets.

The remainder of the paper is organized as follows: In Section 2 we state the OPTIMAL JACOBIAN ACCUMULATION problem, and we prove its NP-completeness by reduction from ENSEMBLE COMPUTATION taking into account possible algebraic dependences between the local partial derivatives. Various special cases are considered as well as the generalization for higher derivative tensors. In Section 3 we discuss the results in the context of Jacobian accumulation techniques that exploit structural properties of the computational graph.

2 Result

ENSEMBLE COMPUTATION [8] is defined as follows: Given a collection $C = \{C_\nu \subseteq A : \nu = 1, \ldots, |C|\}$ of subsets $C_\nu = \{c_i^\nu : i = 1, \ldots, |C_\nu|\}$ of a finite set A and a positive integer Ω is there a sequence $u_i = s_i \cup t_i$ for $i = 1, \ldots, \omega$ of $\omega \leq \Omega$ union operations, where each s_i and t_i is either $\{a\}$ for some $a \in A$ or u_j for some $j < i$, such that s_i and t_i are disjoint for $i = 1, \ldots, \omega$ and such that for every subset $C_\nu \in C$, $\nu = 1, \ldots, |C|$, there is some u_i, $1 \leq i \leq \omega$, that is identical to C_ν?

Lemma 1. ENSEMBLE COMPUTATION *is NP-complete.*

Proof. The proof is by reduction from VERTEX COVER as shown in [8]. □

Example 2 *Let an instance of* ENSEMBLE COMPUTATION *be given by*

$$A = \{a_1, a_2, a_3, a_4\}$$
$$C = \{\{a_1, a_2\}, \{a_2, a_3, a_4\}, \{a_1, a_3, a_4\}\}$$

and $\Omega = 4$. The answer to the decision problem is positive with a corresponding instance given by

$$\begin{aligned} C_1 = u_1 &= \{a_1\} \cup \{a_2\} \\ u_2 &= \{a_3\} \cup \{a_4\} \\ C_2 = u_3 &= \{a_2\} \cup u_2 \\ C_3 = u_4 &= \{a_1\} \cup u_2 \quad . \end{aligned}$$

W.l.o.g. we assume $A \subsetneq I\!R$ in the following. The number of floating-point operations (scalar multiplications "$*$" and additions "$+$") can potentially be decreased compared to the straight-forward application of Equation (7) by exploiting the algebraic laws of the field $(I\!R, +, *)$, that is associativity, commutativity, and distributivity. The corresponding computer programs are referred to as *Jacobian accumulation codes.* There is an exponential (in the size of G) number of them.

Example 3 *The derivation of the Jacobian accumulation code for Example 1 which corresponds to Equation (7) is straight-forward. In*

$$\begin{aligned} f_{1,1} &= c_{4,3} * c_{3,1} * c_{1,-3} + c_{4,1} * c_{1,-3} \\ f_{1,2} &= c_{4,3} * c_{3,1} * c_{1,-2} + c_{4,1} * c_{1,-2} \\ f_{1,3} &= c_{4,3} * c_{3,2} * c_{2,-1} \\ f_{1,4} &= c_{4,3} * c_{3,2} * c_{2,0} \\ f_{2,1} &= c_{5,3} * c_{3,1} * c_{1,-3} \\ &\vdots \\ f_{7,4} &= c_{7,3} * c_{3,2} * c_{2,0} + c_{7,2} * c_{2,0} \end{aligned}$$

36 multiplications and 4 additions are performed. It is less obvious that the Jacobian can also be obtained by the Jacobian accumulation code

$c_{4,1}{+}{=}c_{4,3} * c_{3,1};\ ; f_{1,1} = c_{4,1} * c_{1,-3};\quad f_{1,2} = c_{4,1} * c_{1,-2}$
$c_{7,2}{+}{=}c_{7,3} * c_{3,2};\quad f_{4,3} = c_{7,2} * c_{2,-1};\quad f_{4,4} = c_{7,2} * c_{2,0}$
$c_{3,-3} = c_{3,1} * c_{1,-3};\quad c_{3,-2} = c_{3,1} * c_{1,-2};\quad c_{3,-1} = c_{3,2} * c_{2,-1};\quad c_{3,0} = c_{3,2} * c_{2,0}$
$f_{1,3} = c_{4,3} * c_{3,-1};\quad f_{1,4} = c_{4,3} * c_{3,0}$
$f_{2,1} = c_{5,3} * c_{3,-3};\quad f_{2,2} = c_{5,3} * c_{3,-2};\quad f_{2,3} = c_{5,3} * c_{3,-1};\quad f_{2,4} = c_{5,3} * c_{3,0}$
$f_{3,1} = c_{6,3} * c_{3,-3};\quad f_{3,2} = c_{6,3} * c_{3,-2};\quad f_{3,3} = c_{6,3} * c_{3,-1};\quad f_{3,4} = c_{6,3} * c_{3,0}$
$f_{4,1} = c_{7,3} * c_{3,-3};\quad f_{4,2} = c_{7,3} * c_{3,-2}$

at the cost of 22 multiplications and 2 additions.[4] *Assuming structural independence of the local partial derivatives the above is the best solution for the Bat graph*[5] *known today. See [20] for details.*

We refer to the problem of minimizing the number of scalar multiplications and additions performed by a Jacobian accumulation code as the OPTIMAL JACOBIAN ACCUMULATION (OJA) problem. The corresponding decision version is defined as follows:

Given a linearized computational graph G of a vector function F as defined in Equation (1) and a positive integer Ω is there a sequence of scalar assignments $u_k = s_k \circ t_k$, $\circ \in \{+, *\}$, $k = 1, \ldots, \omega$, where each s_k and t_k is either $c_{j,i}$ for some $(i, j) \in E$ or $u_{k'}$ for some $k' < k$ such that $\omega \leq \Omega$ and for every Jacobian entry there is some identical u_k, $k \leq \omega$?

Theorem 1. OPTIMAL JACOBIAN ACCUMULATION *is NP-complete.*

Proof. We reduce from ENSEMBLE COMPUTATION. A given solution is verified in polynomial time by counting the number of operations.

Given an arbitrary instance of ENSEMBLE COMPUTATION we define the corresponding OPTIMAL JACOBIAN ACCUMULATION problem as follows:

Consider $\mathbf{y} = F(\mathbf{x}, \mathbf{a})$ where $\mathbf{x} \in I\!R^{|C|}$, $\mathbf{a} \equiv (a_j)_{j=1,\ldots,|A|} \in I\!R^{|A|}$ is a vector containing all elements of A, and $F : I\!R^{|C|+|A|} \to I\!R^{|C|}$ defined as

$$y_\nu = x_\nu * \prod_{j=1}^{|C_\nu|} c_j^\nu \tag{8}$$

for $\nu = 1, \ldots, |C|$ and where c_j^ν is equal to some a_i, $i = 1, \ldots, |A|$, for all ν and j. The elements of A are set to be random numbers. We assume that

$$\prod_{j=1}^{|C_\nu|} c_j^\nu = c_1^\nu * c_2^\nu * \ldots * c_{|C_\nu|}^\nu$$

[4] We use the C-style notation $a{+}{=}b$ to denote the incrementation of a by b.
[5] Turn the graph in Figure 1 upside down and use a little imagination to verify the appropriateness of the naming.

and that, w.l.o.g.,

$$x_\nu * \prod_{j=1}^{|C_\nu|} c_j^\nu = x_\nu * c_1^\nu * c_2^\nu * \ldots * c_{|C_\nu|}^\nu = (\ldots((x_\nu * c_1^\nu) * c_2^\nu)\ldots) * c_{|C_\nu|}^\nu$$

is evaluated from left to right. This transformation is linear with respect to the original instance of ENSEMBLE COMPUTATION in both space and time. The Jacobian $F'(\mathbf{x}, \mathbf{a})$ is a diagonal matrix with nonzero entries

$$f_{\nu,\nu} = \prod_{j=1}^{|C_\nu|} c_j^\nu$$

for $\nu = 1, \ldots, |C|$. Is there a Jacobian accumulation code for $F'(\mathbf{x}, \mathbf{a})$ of length less than Ω? We claim that the answer is positive if and only if there is a solution of the corresponding ENSEMBLE COMPUTATION problem.

"$\Leftarrow$" In a given solution of the ENSEMBLE COMPUTATION problem we simply substitute $*$ for $\cup$. The length of the resulting Jacobian accumulation code is less than Ω. The correctness of the code follows immediately from the definition of both ENSEMBLE COMPUTATION and OPTIMAL JACOBIAN ACCUMULATION.

"$\Rightarrow$" No additions are performed by any Jacobian accumulation code as a result of the definition of $F(\mathbf{x}, \mathbf{a})$. Let

$$u_k = s_k * t_k, \quad k = 1, \ldots, \omega \quad , \tag{9}$$

be a solution of the OPTIMAL JACOBIAN ACCUMULATION problem. We claim that a solution of ENSEMBLE COMPUTATION is obtained by substitution of $\cup$ for $*$ in Equation (9). According to the definition of OPTIMAL JACOBIAN ACCUMULATION each s_k and t_k is either a_i for some $i \in \{1, \ldots, |A|\}$ or $u_{k'}$ for some $k' < k$. Similarly, there is some u_j, $j \leq \omega$, for every subset $C_\nu \in C$, $\nu = 1, \ldots, |C|$, that is identical to C_ν since all Jacobian entries are computed by Equation (9). It remains to be shown that s_i and t_i are disjoint for $1 \leq i \leq \omega$.

W.l.o.g. suppose that there is some $i \leq \omega$ such that $s_i \cap t_i = \{b\}$. Hence the computation of u_i in the Jacobian accumulation code involves a factor $b * b$. Note that such a factor is not part of any Jacobian entry which implies that the computation of u_i is obsolete and therefore cannot be part of an optimal Jacobian accumulation code.

While $\{a, b\} = \{c, d\}$ if and only if $a = c$ and $b = d$ (or $a = d$ and $b = c$) we may well have $a * b = c * d$. Pick, for example, $a = 2$, $b = 6$, $c = 3$, and $d = 4$. However as A consists of random numbers the similar treatment of $(A, \cup)$ and $(A, *)$ is feasible. □

Example 4 *The equivalent instance of* OPTIMAL JACOBIAN ACCUMULATION *for Example 2 comes as a vector function $F : \mathbb{R}^{3+4} \to \mathbb{R}^3$ defined by the following system of equations:*

$$\begin{aligned} y_1 &= x_1 * a_1 * a_2 \\ y_2 &= x_2 * a_2 * a_3 * a_4 \\ y_3 &= x_3 * a_1 * a_3 * a_4 \quad . \end{aligned}$$

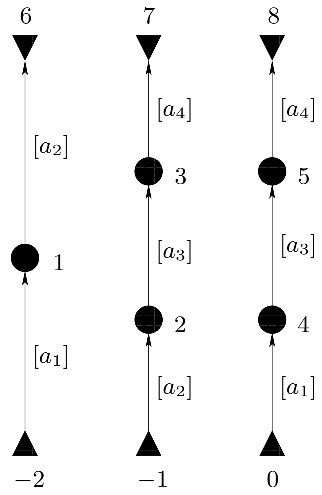

Fig. 2. Linearized Computational Graph for Example 4.

The Jacobian accumulation code according to Equation (7) is

$$\begin{aligned} f_{1,1} &= a_1 * a_2 \\ f_{2,2} &= a_2 * a_3 * a_4 \\ f_{3,3} &= a_1 * a_3 * a_4 \quad . \end{aligned}$$

A Jacobian accumulation code that solves the Optimal Jacobian Accumulation *problem with $\Omega = 4$ is given as*

$$\begin{aligned} t &= a_3 * a_4 \\ f_{1,1} &= a_1 * a_2 \\ f_{2,2} &= a_2 * t \\ f_{3,3} &= a_1 * t \quad . \end{aligned}$$

While the straight-forward reduction from Ensemble Computation in the proof of Theorem 1 is sufficient to show the NP-completeness of Optimal Jacobian Accumulation it does not illustrate how differentiation makes life difficult. A similar combinatorial optimization problem arises already for the function evaluation itself. To circumvent this inconvenience one may introduce local nonlinearities as follows.

Consider $\mathbf{y} = F(\mathbf{x}, \mathbf{a})$ where $\mathbf{x} \in I\!R^{|C|}$, $\mathbf{a} \equiv (a_j)_{j=1,\dots,|A|} \in I\!R^{|A|}$ is a vector containing all elements of A, and $F : I\!R^{|C|+|A|} \rightarrow I\!R^{|C|}$ defined as

$$y_\nu = \varphi^\nu_{|C_\nu|-1}(\dots \varphi^\nu_2(\varphi^\nu_1(x_\nu * c^\nu_1) * c^\nu_2) \dots \; c^\nu_{|C_\nu|-1}) * c^\nu_{|C_\nu|} \tag{10}$$

for $\nu = 1, \dots, |C|$ and where c^ν_j is equal to some random number a_i, $i = 1, \dots, |A|$, for all ν and j. The unary nonlinear functions are chosen such that $\varphi^\nu_j(v) \neq \varphi^\mu_k(w)$ if $v \neq w$. The function needs to be evaluated forward. The algebraic properties of scalar multiplication cannot be exploited anymore.

Differentiation of Equation (10) with respect to $\mathbf{x}$ yields again a diagonal matrix with nonzero entries

$$f_{\nu,\nu} = c^{\nu}_{|C_\nu|} * \prod_{j=1}^{|C_\nu|-1} \frac{\partial \varphi^{\nu}_j(v)}{\partial v}(v^{\nu}_j) * c^{\nu}_j$$

for $\nu = 1, \ldots, |C|$ and where

$$v^{\nu}_j = \begin{cases} x_\nu * c^{\nu}_j & \text{if } j = 1 \\ \varphi^{\nu}_j(\varphi^{\nu}_1(x_\nu * c^{\nu}_1) * c^{\nu}_j) & \text{if } j = 2 \\ \varphi^{\nu}_{j-1}(\ldots \varphi^{\nu}_1(x_\nu * c^{\nu}_1) \ldots * c^{\nu}_{j-1}) * c^{\nu}_j & \text{if } j = 3, \ldots, |C_\nu| - 1 \end{cases} .$$

The products of the c^{ν}_j in the derivative accumulation yield an Ensemble Computation problem with $\Omega := \Omega + \sum_{\nu=1}^{|C|} |C_\nu| - |C|$.

Example 5 *Consider*

$$\begin{aligned} y_1 &= \varphi_1(x_1 * a_1) * a_2 \\ y_2 &= \varphi_3(\varphi_2(x_2 * a_2) * a_3) * a_4 \\ y_3 &= \varphi_5(\varphi_4(x_3 * a_1) * a_3) * a_4 \quad . \end{aligned}$$

The Jacobian accumulation code according to Equation (7) is

$$\begin{aligned} f_{1,1} &= a_1 * \frac{\partial \varphi_1(v)}{\partial v}(x_1 * a_1) * a_2 \\ f_{2,2} &= a_2 * \frac{\partial \varphi_2(v)}{\partial v}(x_2 * a_2) * a_3 * \frac{\partial \varphi_3(v)}{\partial v}(\varphi_2(x_2 * a_2) * a_3) * a_4 \\ f_{3,3} &= a_1 * \frac{\partial \varphi_4(v)}{\partial v}(x_3 * a_1) * a_3 * \frac{\partial \varphi_5(v)}{\partial v}(\varphi_4(x_3 * a_1) * a_3) * a_4 \quad . \end{aligned}$$

A Jacobian accumulation code that solves the Optimal Jacobian Accumulation *problem with $\Omega = 4 + \sum_{\nu=1}^{|C|} |C_\nu| - |C| = 9$ is given as*

$$\begin{aligned} t &= a_3 * a_4 \\ f_{1,1} &= a_1 * \frac{\partial \varphi_1(v)}{\partial v}(x_1 * a_1) * a_2 \\ f_{2,2} &= a_2 * \frac{\partial \varphi_2(v)}{\partial v}(x_2 * a_2) * \frac{\partial \varphi_3(v)}{\partial v}(\varphi_2(x_2 * a_2) * a_3) * t \\ f_{3,3} &= a_1 * \frac{\partial \varphi_4(v)}{\partial v}(x_3 * a_1) * \frac{\partial \varphi_5(v)}{\partial v}(\varphi_4(x_3 * a_1) * a_3) * t \quad . \end{aligned}$$

Note that only multiplications of local partial derivatives are counted. The values $\frac{\partial \varphi^{\nu}_j(v)}{\partial v}(v^{\nu}_j)$ are assumed to be available after linearization of the computational graph as described in Section 1.

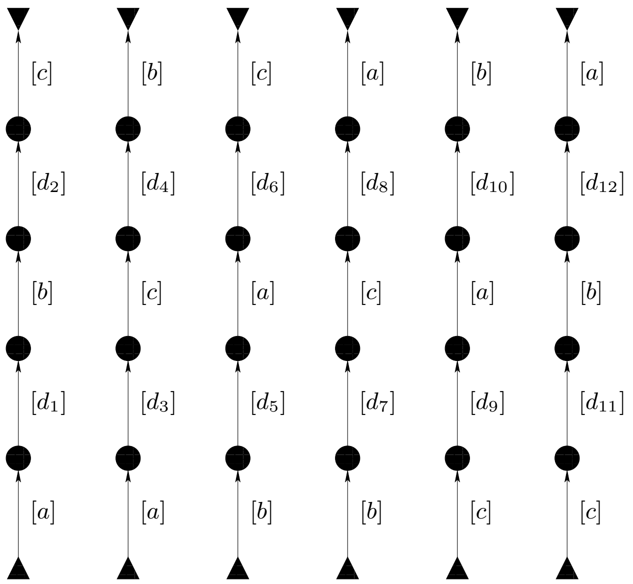

Fig. 3. Cheap Jacobians – An Extreme Case

We may ask ourselves how large the savings in Jacobian accumulation can become compared to the computational cost of evaluating the underlying function. For the type of function used in the proof of Theorem 1 savings are negligible as outlined before. The proposed extension with local nonlinear functions yields maximum savings of a factor of roughly two. Consider therefore $\mathbf{y} = F(\mathbf{x}, \mathbf{a})$ as before where $\mathbf{x} \in I\!R^{|A|!}$, and $F : I\!R^{|A|!+|A|} \rightarrow I\!R^{|A|!}$ defined as

$$y_\nu = \varphi^\nu_{|A|-1}(\ldots \varphi^\nu_1(x_\nu * c^\nu_1) * \ldots c^\nu_{|A|-1}) * c^\nu_{|A|} \tag{11}$$

for $\nu = 1, \ldots, |A|!$ such that the $c^\nu_1, \ldots, c^\nu_{|A|}$ range over all permutations of the elements in A. An example computational graph is shown in Figure 3 for $|A| = 3$, where $A = \{a, b, c\}$ and the partial derivatives of the φ^ν_j are denoted by d_j for $j = 1, \ldots, (|A| - 1)|A|!$. Assuming again that the $\frac{\partial \varphi^\nu_j(v)}{\partial v}(v^\nu_j)$ are algebraically independent we observe that the function evaluation takes $(2|A|-1)|A|!$ operations whereas the Jacobian can be accumulated at a cost of $2+(|A|-1)|A|!$ operations. For the example in Figure 3 we get $(2|A|-1)|A|! = 30$ and $2+(|A|-1)|A|! = 14$.

The idea underlying the proof of Theorem 1 can be applied to a wide variety of practically relevant derivative computations as shown below. A major strength of AD is the ability to compute Jacobian-vector (for example, used in matrix-free Newton-type methods for solving systems of nonlinear equations [17]) and transposed-Jacobian-vector products (for example, used in nonlinear least squares optimization [16]) in its forward and reverse modes, respectively, without building the full Jacobian. The latter can be computed by the tangent-linear (resp. adjoint) code that implements Equation (2) (resp. Equation (3)) at a complexity that is proportional to n (resp. m) if $\dot{\mathbf{x}}$ (resp. $\bar{\mathbf{y}}$) ranges over the Cartesian basis vectors in $I\!R^n$ (resp. $I\!R^m$). Gradients ($m = 1$), in particular, can be evaluated as adjoints in reverse mode at a computational cost that is a small constant multiple of the cost of evaluating F itself [22].

The accumulation of the Jacobian for a function with the Bat graph $G = (V, E)$ displayed in Figure 1 takes $n * |E| = 4 * 12 = 48$ scalar multiplications in forward mode and, as $n = m$, the same number in reverse mode. The best Jacobian accumulation code for the Bat graph know so far takes less than half this number. We believe that this improvement is rather impressive keeping in mind that the function at hand is extremely simple compared to numerical simulation programs for real-world applications. See [10] for further details on AD. In any case, the minimization of the corresponding computational effort must play a central role in all attempts to speed up numerical methods that rely on some kind of derivative information. In the following we derive NP-completeness results for various related problems.

We define the Optimal Gradient Accumulation problem similar to Optimal Jacobian Accumulation with a scalar dependent variable in Equation (1).

Theorem 2. Optimal Gradient Accumulation *is NP-complete.*

Proof. The proof follows immediately by modifying Equation (8) in the proof of Theorem 1 as follows

$$y = \sum_{\nu=1}^{|C|} y_\nu = \sum_{\nu=1}^{|C|} \left(x_\nu * \prod_{j=1}^{|C_\nu|} c_j^\nu \right) \quad .$$

The additional sum introduces a linear section that can be eliminated statically as described in [21]. The nonzeros on the diagonal of the Jacobian become the gradient entries. □

Gradients are single rows in the Jacobian. Similarly, we can show that the computation of single columns, that is, for a scalar independent variable in Equation (1), is NP-complete by considering the Jacobian of

$$y_\nu = x * \prod_{j=1}^{|C_\nu|} c_j^\nu \quad \nu = 1, \ldots, |C| \quad .$$

Example 6 *The linearized computational graphs for*

$$y = x_1 * a_1 * a_2 + x_2 * a_2 * a_3 * a_4 + x_3 * a_1 * a_3 * a_4 \tag{12}$$

and

$$\begin{aligned} y_1 &= x * a_1 * a_2 \\ y_2 &= x * a_2 * a_3 * a_4 \\ y_3 &= x * a_1 * a_3 * a_4 \quad . \end{aligned} \tag{13}$$

are shown in Figure 4(a) and Figure 5, respectively. The graph in Figure 4(b) is obtained by applying static elimination techniques for edges that carry label 1 *to the graph in Figure 4(a) as described in [21].*

We define the Optimal Tangent Computation and Optimal Adjoint Computation problems as the problems of computing the product of the Jacobian with an n-vector $\dot{\mathbf{x}}$ and the product of the transposed Jacobian with an m-vector $\bar{\mathbf{y}}$, respectively.

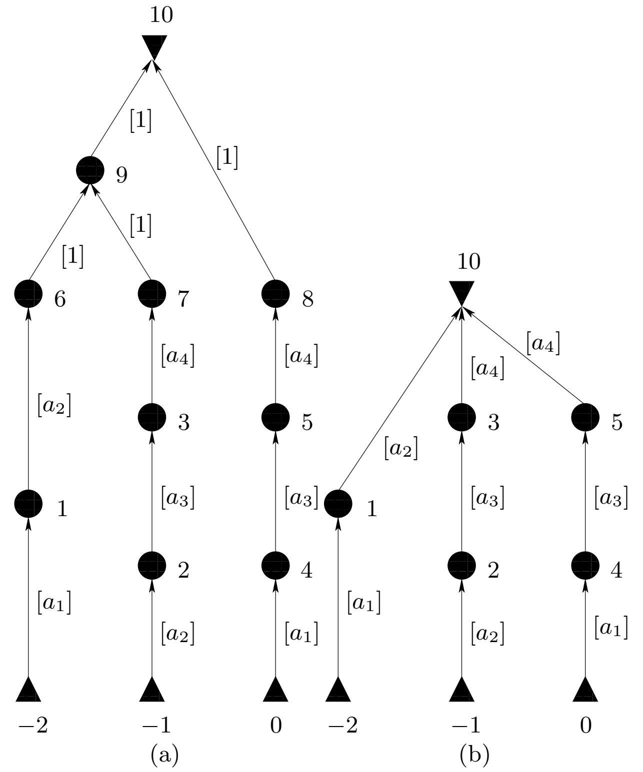

Fig. 4. Linearized computational graph for Equation (12) before (a) and after (b) folding of constant local partial derivatives.

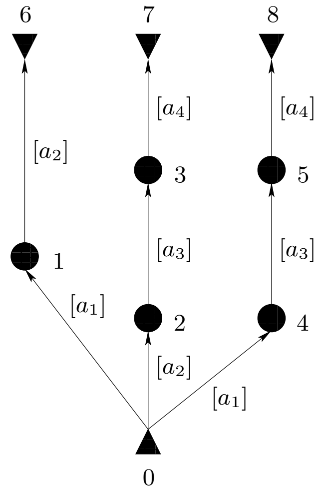

Fig. 5. Linearized computational graph for Equation (13).

Theorem 3. Optimal Tangent Computation *and* Optimal Adjoint Computation *are NP-complete.*

Proof. Setting $\dot{x}_\nu = c_1^\nu$ for $\nu = 1, \ldots, |C|$ the computation of $F' * \dot{\mathbf{x}}$ becomes equivalent to the computation of the $|C| \times 1$ Jacobian of $\mathbf{y}$ with respect to x for

$$\mathbf{y} = \dot{F}(x, \dot{\mathbf{x}}, \mathbf{a}), \quad \dot{F} : I\!R^{1+|C|+|A|} \supseteq D \rightarrow I\!R^{|C|}$$

defined as

$$y_\nu = x * \dot{x}_\nu * \prod_{j=2}^{|C_\nu|} c_j^\nu \quad .$$

The proof is similar to the proof of Theorem 1.

The result for $(F')^T * \bar{\mathbf{y}}$ follows by symmetry. Simply set $\bar{y}_\nu = c_{|C_\nu|}^\nu$ for $\nu = 1, \ldots, |C|$ and consider

$$y = \bar{F}(\mathbf{x}, \mathbf{a}, \bar{\mathbf{y}}), \quad F : I\!R^{|C|+|A|+|C|} \supseteq D \rightarrow I\!R$$

defined as

$$y = \sum_{\nu=1}^{|C|} \left(x_\nu * c_1^\nu * \ldots * c_{|C_\nu|-1}^\nu * \bar{y}_\nu \right) = \sum_{\nu=1}^{|C|} \left(x_\nu * \prod_{j=1}^{|C_\nu|-1} c_j^\nu \right) * \bar{y}_\nu) \quad . \quad \square$$

Example 7 *With* $\bar{\mathbf{y}} = (a_1, a_2, a_1)$ *and* $\dot{\mathbf{x}}^T = (a_2, a_4, a_4)$ *the linearized computational graphs are similar to Figure 4 (a) and Figure 5, respectively.*

The results can be generalized to derivatives of arbitrary order $q > 0$ by considering

$$y_\nu = \frac{x_\nu^q}{q!} \prod_{j=1}^{|C_\nu|} c_j^\nu \quad .$$

for $\nu = 1, \ldots, |C|$ instead of Equation (8) and deriving the corresponding special cases. The division by $q!$ is not essential. All it does is ensure that the qth derivative of y_ν with respect to x_ν is equal to $\prod_{j=1}^{|C_\nu|} c_j^\nu$, thus making the proof of Theorem 1 applicable without any modification.

3 Discussion

Previous work on the subject has always made the assumption that the local partial derivatives are algebraically independent [4, 12, 13, 19]. The potential impact of dependences between the $c_{j,i}$ on the complexity of Jacobian accumulation has been acknowledged only recently during personal communication with A. Griewank and T. Steihaug at Humboldt University Berlin [1].

The simplicity of the result of reducing Ensemble Computation to Optimal Jacobian Accumulation ensures that all elimination techniques known so far are covered. Vertex [13], edge [18], and face elimination [20] as well as rerouting and normalization [14] exploit different structural properties of the computational graph. Neither rerouting nor normalization are applicable if all dependent variables are mutually independent and all intermediate vertices have both a single predecessor and a single successor. Both face and edge elimination are equivalent to vertex elimination in this case. Hence we have shown the NP-completeness of a special case that implies the NP-completeness of all elimination problems on versions of the computational graph known so far. Note that we have not shown the NP-completeness of Optimal Jacobian Computation for mutually algebraically independent local partial derivatives. We conjecture that a similar result can be derived in this case. However, the proof appears to be less straight-forward than that developed in this paper. Moreover, the relevance of this special case becomes questionable in the light of these new results.

Optimal Adjoint Computation is equivalent to the problem of computing adjoints with a minimal number of operations if the entire code list can be stored. Hence, it represents a special case of the Optimal Checkpointing problem that aims to balance the use of storage and recomputation such that the overall number of operations is minimized [23, 24]. Theoretically, the NP-completeness of this special case could be taken as proof for the NP-completeness of the Optimal Checkpointing problem. However, one must acknowledge that there is no Optimal Checkpointing problem if the entire execution of the function can be stored. Hence we envision further work to be necessary in order to handle this highly relevant problem theoretically in a consistent way. There is no doubt that practical algorithmic work is crucial to speed up derivative-based numerical methods.

All existing algorithms for minimizing the operations count of Jacobian accumulation codes aim to exploit structural properties of the computational graph. The main conclusion from this paper is that a mind shift is required to tackle the Optimal Jacobian Accumulation problem adequately. Tools for AD need to take potential algebraic dependences between the local partial derivatives into account. First steps in this direction are currently made at Humboldt University Berlin and RWTH Aachen University as part of a collaborative research project. The practical implementation of new (pre)accumulation strategies in existing AD tools (see, for example, [9, 11, 15, 25] and `www.autodiff.org`) remains a major technical challenge.

References

1. Personal communication with T. Steihaug, Bergen University, Norway, and A. Griewank at Humboldt University Berlin, March 2005.
2. W. Baur and V. Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22:317–330, 1983.
3. M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors. *Computational Differentiation: Techniques, Applications, and Tools*, Proceedings Series. SIAM, 1996.
4. C. Bischof and M. Haghighat. Hierarchical approaches to automatic differentiation. In *[3]*, pages 82–94.
5. M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, editors. *Automatic Differentiation: Applications, Theory, and Tools*, number 50 in Lecture Notes in Computational Science and Engineering, Berlin, 2005. Springer.
6. G. Corliss, C. Faure, A. Griewank, L. Hascoët, and U. Naumann, editors. *Automatic Differentiation of Algorithms – From Simulation to Optimization*, New York, 2002. Springer.
7. G. Corliss and A. Griewank, editors. *Automatic Differentiation: Theory, Implementation, and Application*, Proceedings Series. SIAM, 1991.
8. M. Garey and D. Johnson. *Computers and Intractability - A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, 1979.

9. R. Giering and T. Kaminski. Applying TAF to generate efficient derivative code of Fortran 77-95 programs. In *Proceedings of GAMM 2002, Augsburg, Germany*, 2002.
10. A. Griewank. *Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Applied Mathematics. SIAM, Philadelphia, 2000.
11. A. Griewank, D. Juedes, and J. Utke. ADOL–C, a package for the automatic differentiation of algorithms written in C/C++. *ACM Trans. Math. Software*, 22(2):131–167, 1996.
12. A. Griewank and U. Naumann. Accumulating Jacobians as chained sparse matrix products. *Math. Prog.*, 3(95):555–571, 2003.
13. A. Griewank and S. Reese. On the calculation of Jacobian matrices by the Markovitz rule. In *[7]*, pages 126–135.
14. A. Griewank and O. Vogel. Analysis and exploitation of Jacobian scarcity. In *Proceedings of HPSC Hanoi*. Springer, 2003.
15. L. Hascoët and V. Pascual. Tapenade 2.1 user's guide. Technical report 300, INRIA, 2004.
16. M. Heath. *Scientific Computing. An Introductory Survey*. McGraw-Hill, New York, 1998.
17. C. Kelley. *Solving Nonlinear Equations with Newton's Method*. SIAM, 2003.
18. U. Naumann. Elimination techniques for cheap Jacobians. In *[6]*, chapter 29, pages 247–253. 2001.
19. U. Naumann. Cheaper Jacobians by Simulated Annealing. *SIAM J. Opt.*, 13(3):660–674, March 2002.
20. U. Naumann. Optimal accumulation of Jacobian matrices by elimination methods on the dual computational graph. *Math. Prog.*, 3(99):399–421, 2004.
21. U. Naumann and J. Utke. Optimality-preserving elimination of linearities in Jacobian accumulation. *Electronic Transactions on Numerical Analysis (ETNA)*, 2005. To appear in special issue on combinatorial scientific computing dedicated to Professor Alan George.
22. B. Speelpenning. *Compiling Fast Partial Derivatives of Functions Given by Algorithms*. PhD thesis, University of Chicago, Urbana-Champaign, IL, Jan. 1980.
23. A. Walther. *Program Reversal Schedules for Single- and Multi-processor Machines*. PhD thesis, Institute of Scientific Computing, Technical University Dresden, Germany, 1999.
24. A. Walther and A. Griewank. New results on program reversals. In *[6]*, chapter 28, pages 237–243. Springer, 2001.
25. C. Wunsch, C. Hill, P. Heimbach, U. Naumann, J. Utke, M. Fagan, and N. Tallent. OpenAD. Preprint ANL/MCS-P1230-0205, Argonne National Laboratory, Argonne, IL, February 2005.