Math. Program., Ser. A (2008) 112:427–441
DOI 10.1007/s10107-006-0042-z

FULL LENGTH PAPER

Optimal Jacobian accumulation is NP-complete

Uwe Naumann

Received: 17 January 2006 / Accepted: 22 August 2006 / Published online: 21 October 2006

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.

Keywords Automatic differentiation · Complexity · NP-completeness

Mathematics Subject Classification (2000) 26B10 · 68Q17

1 Context

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

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

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

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

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

U. Naumann (✉)
Software and Tools for Computational Engineering, Department of Computer Science,
RWTH Aachen University, 52056 Aachen, Germany
e-mail: naumann@stce.rwth-aachen.de
URL: http://www.stce.rwth-aachen.de

and *adjoint*

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

versions of numerical simulation programs with potentially complicated intra- and interprocedural flow of control can be generated automatically by AD tools [9,11,15,22]. 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 Eq. (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 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 $V = \{1-n, \ldots, p+m\}$ 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

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

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 Fig. 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

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

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

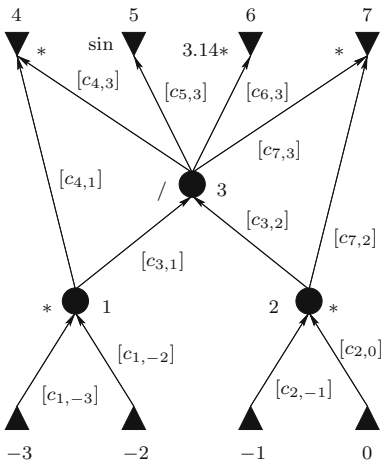

Fig. 1 *Linearized Computational Graph G:* Independent and dependent variables are represented by *black triangles* pointing upward $(-3,\ldots,0)$ and downward $(4,\ldots,7)$, respectively. Intermediate and maximal vertices are marked with their respective elemental functions. The corresponding local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

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 Eq. (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]

Example 1 Consider the following linearized code list of a vector function $F: \mathbb{R}^4 \to \mathbb{R}^4$:

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

[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].

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 Fig. 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.

The remainder of the paper is organized as follows: In Sect. 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 Sect. 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

$$\begin{aligned} A &= \{a_1, a_2, a_3, a_4\} \\ C &= \{\{a_1, a_2\}, \{a_2, a_3, a_4\}, \{a_1, a_3, a_4\}\} \end{aligned}$$

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

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

W.l.o.g. we assume $A \subsetneq \mathbb{R}$ in the following. The number of floating-point operations (scalar multiplications "$*$" and additions "$+$") can potentially be decreased compared to the straight-forward application of Eq. (7) by exploiting the algebraic laws of the field $(\mathbb{R}, +, *)$, 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 Eq. (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

$$\begin{array}{llll}
c_{4,1} += c_{4,3} * c_{3,1}; & f_{1,1} = c_{4,1} * c_{1,-3}; & f_{1,2} = c_{4,1} * c_{1,-2} & \\
c_{7,2} += c_{7,3} * c_{3,2}; & f_{4,3} = c_{7,2} * c_{2,-1}; & f_{4,4} = c_{7,2} * c_{2,0} & \\
c_{3,-3} = c_{3,1} * c_{1,-3}; & c_{3,-2} = c_{3,1} * c_{1,-2}; & c_{3,-1} = c_{3,2} * c_{2,-1}; & c_{3,0} = c_{3,2} * c_{2,0}\\
f_{1,3} = c_{4,3} * c_{3,-1}; & f_{1,4} = c_{4,3} * c_{3,0} & & \\
f_{2,1} = c_{5,3} * c_{3,-3}; & f_{2,2} = c_{5,3} * c_{3,-2}; & f_{2,3} = c_{5,3} * c_{3,-1}; & f_{2,4} = c_{5,3} * c_{3,0}\\
f_{3,1} = c_{6,3} * c_{3,-3}; & f_{3,2} = c_{6,3} * c_{3,-2}; & f_{3,3} = c_{6,3} * c_{3,-1}; & f_{3,4} = c_{6,3} * c_{3,0}\\
f_{4,1} = c_{7,3} * c_{3,-3}; & f_{4,2} = c_{7,3} * c_{3,-2} & &
\end{array}$$

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 Eq. (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, $k \leq \omega$?

[4] We use the C-style notation $a+ = b$ to denote the incrementation of a by b.

[5] Turn the graph in Fig. 1 upside down and use a little imagination to verify the appropriateness of the naming.

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 OJA 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

$$y_v = x_v * \prod_{j=1}^{|C_v|} c_j^v \tag{8}$$

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

$$\prod_{j=1}^{|C_v|} c_j^v = c_1^v * c_2^v * \cdots * c_{|C_v|}^v$$

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

$$x_v * \prod_{j=1}^{|C_v|} c_j^v = x_v * c_1^v * c_2^v * \cdots * c_{|C_v|}^v = (\ldots((x_v * c_1^v) * c_2^v)\ldots) * c_{|C_v|}^v$$

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

$$f_{v,v} = \prod_{j=1}^{|C_v|} c_j^v$$

for $v = 1, \ldots, |C|$. 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 OJA.

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

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

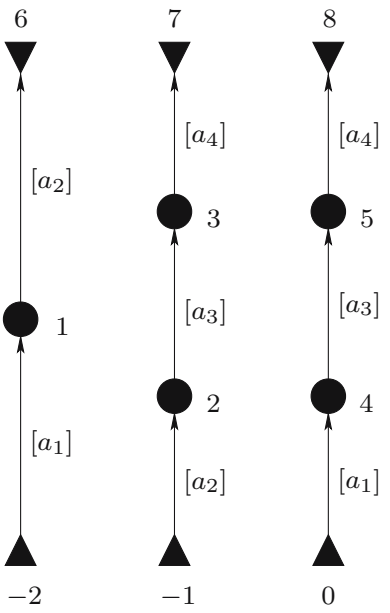

Fig. 2 Shows the corresponding linearized computational graph

be a solution of the OJA problem. We claim that a solution of Ensemble Computation is obtained by substitution of $\cup$ for $*$ in Eq. (9). According to the definition of OJA 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 Eq. (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 OJA for Example 2 comes as a vector function $F : \mathbb{R}^{3+4} \to \mathbb{R}^3$ defined by the following system of equations:

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

The Jacobian accumulation code according to Eq. (7) is

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

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

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

While the straight-forward reduction from Ensemble Computation in the proof of Theorem 1 is sufficient to show the NP-completeness of OJA 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 $\mathbf{x} \in \mathbb{R}^{|C|}$, $\mathbf{a} \equiv (a_j)_{j=1,\ldots,|A|} \in \mathbb{R}^{|A|}$ is a vector containing all elements of A, and $F : \mathbb{R}^{|C|+|A|} \to \mathbb{R}^{|C|}$ defined as

$$y_v = \varphi^v_{|C_v|-1}(\ldots \varphi^v_2(\varphi^v_1(x_v * c^v_1) * c^v_2) \ldots \; c^v_{|C_v|-1}) * c^v_{|C_v|} \tag{10}$$

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

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

$$f_{v,v} = c^v_{|C_v|} * \prod_{j=1}^{|C_v|-1} \frac{\partial \varphi^v_j(v)}{\partial v}(v^v_j) * c^v_j$$

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

$$v^v_j = \begin{cases} x_v * c^v_j & \text{if } j = 1, \\ \varphi^v_j(\varphi^v_1(x_v * c^v_1) * c^v_j) & \text{if } j = 2, \\ \varphi^v_{j-1}(\ldots \varphi^v_1(x_v * c^v_1) * \cdots * c^v_{j-1}) * c^v_j & \text{if } j = 3, \ldots, |C_v| - 1. \end{cases}$$

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

Example 5 Consider

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

The Jacobian accumulation code according to Eq. (7) is

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

A Jacobian accumulation code that solves the OJA problem with $\Omega = 4 + \sum_{v=1}^{|C|} |C_v| - |C| = 9$ is given as

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

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

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 $F : I\!R^{|A|!+|A|} \to I\!R^{|A|!}$ defined as

$$y_v = \varphi_{|A|-1}^v(\ldots \varphi_1^v(x_v * c_1^v) * \cdots c_{|A|-1}^v) * c_{|A|}^v \tag{11}$$

for $v = 1, \ldots, |A|!$ such that the $c_1^v, \ldots, c_{|A|}^v$ range over all permutations of the elements in A. An example computational graph is shown in Fig. 3 for $|A| = 3$, where $A = \{a, b, c\}$ and the partial derivatives of the φ_j^v are denoted by d_j for $j = 1, \ldots, (|A| - 1)|A|!$. Assuming again that the $\frac{\partial \varphi_j^v(v)}{\partial v}(v_j^v)$ are algebraically independent we observe that the function evaluation takes $(2|A| - 1)|A|!$ operations whereas the Jacobian can be accumulated as a function of the local partial

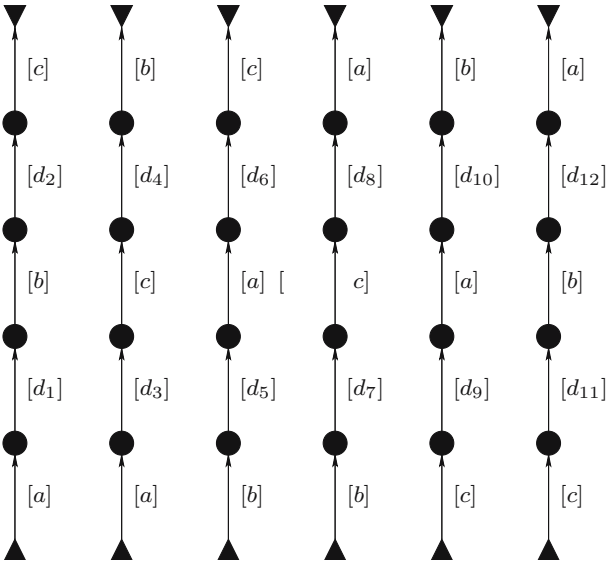

Fig. 3 Cheap Jacobians – an extreme case

derivatives at a cost of $2 + (|A| - 1)|A|!$ operations. For the example in Fig. 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 Eq. (2) (resp. Eq. (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 $\mathbb{R}^n$ (resp. $\mathbb{R}^m$). 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 [23].

The accumulation of the Jacobian for a function with the Bat graph $G = (V, E)$ displayed in Fig. 1 takes $n \times |E| = 4 \times 12 = 48$ 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 OJA with a scalar dependent variable in Eq. (1).

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

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

$$y = \sum_{v=1}^{|C|} y_v = \sum_{v=1}^{|C|} \left(x_v * \prod_{j=1}^{|C_v|} c_j^v \right).$$

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 Eq. (1), is NP-complete by considering the Jacobian of

$$y_v = x * \prod_{j=1}^{|C_v|} c_j^v \quad v = 1, \ldots, |C|.$$

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

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

are shown in Figs. 4a and 5, respectively. The graph in Fig. 4b is obtained by applying static elimination techniques for edges that carry label 1 to the graph in Fig. 4a 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.

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

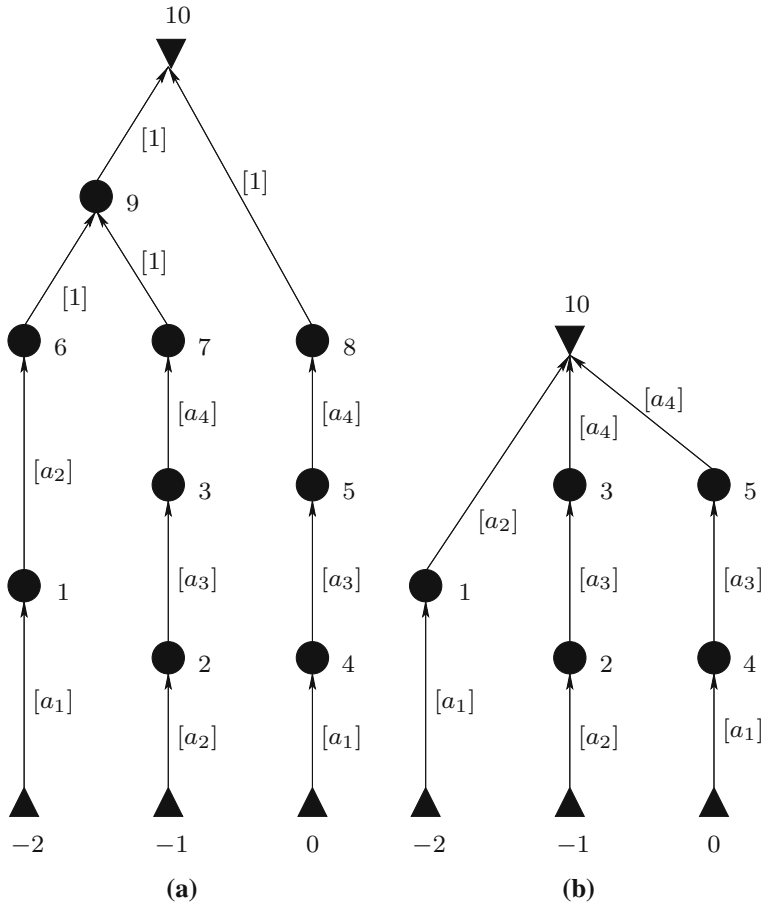

Fig. 4 Linearized computational graph for Eq. (12) before (**a**) and after (**b**) folding of constant local partial derivatives

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

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

defined as

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

The proof is similar to the proof of Theorem 1.

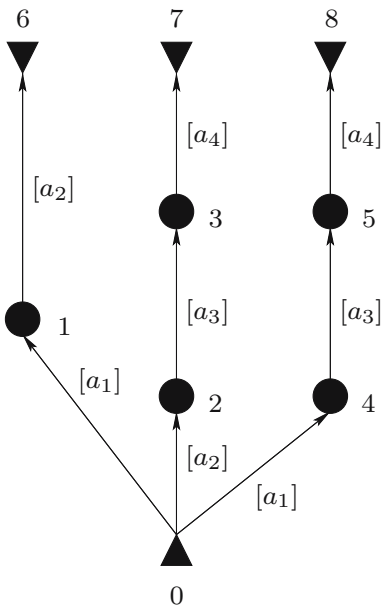

Fig. 5 Linearized computational graph for Eq. (13)

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

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

defined as

$$y = \sum_{\nu=1}^{|C|} \left(x_\nu * c^\nu_1 * \cdots * c^\nu_{|C_\nu|-1} * \bar{y}_\nu \right) = \sum_{\nu=1}^{|C|} \left(x_\nu * \prod_{j=1}^{|C_\nu|-1} c^\nu_j \right) * \bar{y}_\nu.$$

□

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

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

$$y_\nu = \frac{x^q_\nu}{q!} \prod_{j=1}^{|C_\nu|} c^\nu_j.$$

for $\nu = 1, \ldots, |C|$ instead of Eq. (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_v with respect to x_v is equal to $\prod_{j=1}^{|C_v|} c_j^v$, 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 Griewank and Steihaug at Humboldt University Berlin [1].

The simplicity of the result of reducing Ensemble Computation to OJA 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 [24,25]. 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 OJA 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, [9,11,15,22] and http://www.autodiff.org) remains a major technical challenge.

Acknowledgements My thanks go to A. Griewank and the anonymous referees for numerous highly useful comments on the manuscript.

References

1. Personal communication with Steihaug, T. Bergen University, Norway, and Griewank, A. at Humboldt University Berlin (2005)
2. Baur, W., Strassen, V.: The complexity of partial derivatives. Theoret. Comput. Sci. **22**, 317–330 (1983)
3. Berz, M., Bischof, C., Corliss, G., Griewank, A. (eds.) Computational differentiation: techniques, applications, and tools. In: Proceedings Series. SIAM Philadelphia (1996)
4. Bischof, C., Haghighat, M.: Hierarchical approaches to automatic differentiation. In: [3], pp. 82–94
5. Bücker, M., Corliss, G., Hovland, P., Naumann, U., Norris, B. (eds.) Automatic Differentiation: Applications, Theory, and Tools. Lecture Notes in Computational Science and Engineering, vol. 50. Springer, Berlin Heidelberg New York (2005)
6. Corliss, G., Faure, C., Griewank, A., Hascoët, L., Naumann, U. (eds.) Automatic Differentiation of Algorithms – From Simulation to Optimization. Springer, Berlin Heidelberg New York (2002)
7. Corliss, G., Griewank, A. (eds.) Automatic Differentiation: Theory, Implementation, and Application. Proceedings Series. SIAM Philadelphia (1991)
8. Garey, M., Johnson, D.: Computers and Intractability – A Guide to the Theory of NP-completeness. W. H. Freeman and Company, San Francisco (1979)
9. Giering, R., Kaminski, T.: Applying TAF to generate efficient derivative code of Fortran 77-95 programs. In: Proceedings of GAMM 2002, Augsburg, Germany (2002)
10. Griewank, A.: Evaluating derivatives. Principles and techniques of algorithmic differentiation. Frontiers in Applied Mathematics, vol. 19. SIAM, Philadelphia (2000)
11. Griewank, A., Juedes, D., Utke, J.: ADOL–C, a package for the automatic differentiation of algorithms written in C/C++. ACM Trans. Math. Softw. **22**(2), 131–167 (1996)
12. Griewank, A., Naumann, U.: Accumulating Jacobians as chained sparse matrix products. Math. Prog. **3**(95), 555–571 (2003)
13. Griewank, A., Reese, S.: On the calculation of Jacobian matrices by the Markovitz rule. In: [7], pp. 126–135
14. Griewank, A., Vogel, O.: Analysis and exploitation of Jacobian scarcity. In: Proceedings of HPSC Hanoi. Springer, Berlin Heidelberg New York (2003)
15. Hascoët, L., Pascual, V.: Tapenade 2.1 user's guide. Technical report 300, INRIA (2004)
16. Heath, M.: Scientific Computing. An Introductory Survey. McGraw-Hill, New York (1998)
17. Kelley, C.: Solving Nonlinear Equations with Newton's Method. SIAM Philadelphia (2003)
18. Naumann, U.: Elimination techniques for cheap Jacobians. In: [6], chap. 29, pp. 247–253 (2001)
19. Naumann, U.: Cheaper Jacobians by simulated annealing. SIAM J. Opt. **13**(3), 660–674 (2002)
20. Naumann, U.: Optimal accumulation of Jacobian matrices by elimination methods on the dual computational graph. Math. Prog. **3**(99), 399–421 (2004)
21. Naumann, U., Utke, J.: Optimality-preserving elimination of linearities in Jacobian accumulation. Electron. Trans. Numer. Anal. (ETNA) **21**, 134–150 (2005)
22. Naumann, U., Utke, J., Wunsch, C., Hill, C., Heimbach, P., Fagan, M., Tallent, N., Strout, M.: Adjoint code by source transformation with open ad/f. In: Proceedings of the European Conference on Computational Fluid Dynamics (ECCOMAS CFD 2006). TU Delft (2006)
23. Speelpenning, B.: Compiling fast partial derivatives of functions given by algorithms. Ph.D. Thesis, University of Chicago (1980)
24. Walther, A.: Program reversal schedules for single- and multi-processor machines. Ph.D. Thesis, Institute of Scientific Computing, Technical University Dresden (1999)
25. Walther, A., Griewank, A.: New results on program reversals. In: [6], chapt. 28, pp. 237–243. Springer, Berlin Heidelberg New York (2001)