On the Calculation of Jacobian Matrices by the Markowitz Rule for Vertex Elimination*

Andreas Griewank† and Shawn Reese‡

October 1991

Abstract. The evaluation of derivative vectors can be performed with optimal computational complexity by the forward or reverse mode of automatic differentiation. This approach may be applied to evaluate first and higher derivatives of any vector function that is defined as the composition of easily differentiated *elementary functions*, typically in the form of a computer program. The more general task of efficiently evaluating Jacobians or other derivative matrices leads to a combinatorial optimization problem, which is conjectured to be NP-hard. Here, we examine this vertex elimination problem and solve it approximately, using a greedy heuristic. Numerical experiments show the resulting Markowitz scheme for Jacobian evaluation to be more efficient than column by column or row by row evaluation using the forward or the reverse mode, respectively.

1 Basic Setting and Assumptions. Most nonlinear vector functions of practical interest are evaluated by computer programs in a high-level computer language such as Fortran or C. Conceptually, the execution of any such *evaluation program* can be viewed as a sequence of scalar assignments

$$v_j \quad = \quad \phi_j(v_i)_{i \in \mathcal{P}_j} \in \mathbf{R} \quad . \tag{1}$$

Here the index set

$$\mathcal{P}_j \ \subset \ \{1, 2, \ldots, j-1\} \tag{2}$$

contains the *predecessors* ,or *parents*, of j. In principle, the *elementary functions* ϕ_j may depend on all currently known variable values $v_j \in \mathbf{R}$ with $i < j$. In practice, most ϕ_j represent either binary arithmetic operations or univariate transcendental functions,

*This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

†Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439

‡Mathematical Sciences Department, Rensselaer Polytechnic Institute, Troy, NY 12181

in which case $\mathcal{P}_j$ contains only one or two indices, respectively. Currently, our software implementation breaks down the evaluation program into such unary and binary elementary functions, but we shall develop the theory without this restriction. For the sake of simplicity we shall assume that all ϕ_j are scalar valued, even though the inclusion of vector-valued elementary functions (e.g., BLAS) might be advantageous in terms of storage and computing time. Without loss of generality, we may assume that the variables v_j are numbered such that they can be combined into three vectors:

$$\begin{array}{rcll} x & \equiv & (v_1, v_2, \ldots & \ldots, v_{n-1}, v_n) \quad (independent) \\ z & \equiv & (v_{n+1}, \ldots & \ldots, v_{n+p}) \quad (intermediate) \\ y & \equiv & (v_{n+p+1}, \ldots & \ldots, v_{n+p+m}) \quad (dependent) \end{array}$$

It is assumed that, given the n components of the variable vector x, the p components of the intermediate vector z, and the m components of the dependent vector y are defined by the sequence of scalar functions (1) for $i = n+1, \ldots n+p+m$. For notational completeness one may define the first n elementary functions ϕ_j with $j \leq n$ as the identities $v_j = x_j$. This requires in particular that the components of x are mutually independent, and we shall similarly assume that the components of y do not occur as arguments in any elementary function. To comply with these restrictions, one may have to add identity assignments at the beginning and end of the evaluation program. For a substantial nonlinear problem, we may assume that p, the number of intermediate variables, is significantly larger than the sum $n + m$. Using induction on j, we may now impose the following key assumption.

Assumption 1 *For some fixed $x \in \mathbf{R}^n$ and $j = n+1, n+2, \ldots, n+p+m$ the elementary functions ϕ_j are well defined and have jointly continuous partial derivatives*

$$c_{ji}(v_i)_{i \in \mathcal{P}_j} \quad \equiv \quad \frac{\partial}{\partial v_i} \phi_j \quad \text{for} \quad i \in \mathcal{P}_j \tag{3}$$

on some neighborhood

$$\mathcal{N}_j \ \subset \ \mathbf{R}^{n_j} \quad \text{with} \quad n_j \equiv |\mathcal{P}_j| \tag{4}$$

of their respective arguments $(v_i)_{i \in \mathcal{P}_j}$.

This condition will be met during the evaluation of a Fortran or C program, unless the argument of a logarithm, square root, reciprocal, or inverse trigonometric function lies outside the respective domain. Except for powers with fractional exponents, all arithmetic operations and standard system functions are in fact analytic in the interior of their domains. In our complexity considerations, we shall make the following assumption for some suitable unit of computational cost:

Assumption 2 *All elementary functions ϕ_j and their gradients $\nabla \phi_j = (c_{ji})_{i \in \mathcal{P}_j}$ are evaluated such that*

$$1 \quad \leq \quad \frac{Cost\{ \ g_j \ += \ \gamma_j \cdot \nabla \phi_j(v_i)_{i \in \mathcal{P}_j} \ \}}{Cost\{ \ \phi_j(v_i)_{i \in \mathcal{P}_j} \ \}} \leq 3 \ , \tag{5}$$

where g_j is an arbitrary n_j-vector and γ_j an arbitrary scalar.

In other words, we assume that incrementing a given vector by a multiple of the gradient $\nabla \phi_j$ at a given point is no more than three times as expensive as evaluating the underlying elementary function ϕ_j at the same argument. This complexity assumption is reasonable on a single-processor machine, even if memory accesses are taken into account. For most arithmetic operations and elementary functions, the complexity bound is rather conservative; but for the multiplication operation, it is sharp (assuming that an addition is exactly half as expensive as a multiplication). On a multiprocessor, the accumulation of a

dot product ϕ_j requires at least $\log_2 n_j$ cycles; but the corresponding gradient incrementation is simply a SAXPY, which can be computed in $\mathcal{O}(1)$ time. More important, several v_j that do not depend directly or indirectly on each other can be evaluated concurrently on several processors. As we shall see later, the computational task of accumulating Jacobians offers even greater prospects for parallelism. Nevertheless, our main goal in this paper is to minimize the total number of arithmetic operations during the accumulation of Jacobians.

Obviously, equation (1) and Assumption 1 imply by the chain rule that on some neighborhoods of the given point x, the composite function

$$y = f(x) \quad f : \mathbf{R}^n \mapsto \mathbf{R}^m \tag{6}$$

is well defined and once continuously differentiable. It is also clear that the entries of the Jacobian matrix

$$J(x) \equiv \nabla_x f(x) \in \mathbf{R}^{m \times n} \tag{7}$$

must be computable from the *elementary partials* c_{ji} defined in Assumption 1. However, the standard version of the chain rule is not directly applicable, and—as we shall see—there are many ways in which the Jacobian can be *accumulated* (i.e., calculated from the c_{ji}).

Now suppose x is a differentiable function of some parameter. Denote differentiation with respect to this scalar by the superscript prime. Then it follows immediately from the chain rule that (1) implies for all $i > n$

$$v_j' \quad = \quad \sum_{i \in \mathcal{P}_j} c_{ji} \cdot v_i' \quad \in \quad \mathbf{R} \quad . \tag{8}$$

When the initial tangent vector x' is dense, the execution of this recurrence involves the multiplication of each elementary partial c_{ji} by exactly one value v_i' and the incrementation of the product to the corresponding v_j'. Thus it follows from Assumption 2 that the cost of one such *forward sweep* is bounded by

$$\frac{1}{3}\mathrm{Cost}\{\, y' = J(x)\, x' \,\} \quad \leq \quad \mathrm{Cost}\{f(x)\} \quad \equiv \quad \sum_{j=n+1}^{n+p+m} \mathrm{Cost}\{\phi_j\} \quad . \tag{9}$$

Here, the identity on the right is in fact an additional assumption that makes obvious sense on a single-processor machine. This complexity estimate is not very exciting, because the directional derivative $J(x)x'$ can be approximated by the divided difference $[f(x + \varepsilon x') - f(x)]/\varepsilon$ for little more than twice the effort of evaluating $f(x)$. However, it should be noted that the numerical result of a forward sweep is free of truncation error and would therefore be exact in infinite-precision arithmetic.

The vector y' obtained in the presence of round-off corresponds to the exact result for a nonlinear system whose elementary partials c_{ij} have been perturbed to

$$\tilde{c}_{ij} \;=\; c_{ij} \cdot (1 + \varepsilon_{ij})^{n_i} \quad \text{with} \quad |\varepsilon_{ij}| \leq \varepsilon \quad ,$$

where ε is the relative machine precision. Since the derivative values are usually obtained with a relative evaluation error of the same order, we can conclude that the calculation of first-directional derivatives is correct up to working accuracy. Similar estimates hold for the other accumulation procedures discussed in this paper. This assertion appears to contradict the usual notion that differentiation is an intrinsically ill-conditioned process. Our more positive result is derived from the assumption that most functions are provided as a composition of (almost) exactly differentiable building blocks, whereas the ill-conditioning assertion applies if the function is specified merely by an oracle that produces values with a certain absolute accuracy.

2 Sparse Forward and Reverse Accumulation of Jacobians. As an immediate consequence of (9), we see that the full Jacobian $J(x)$ can be calculated by n forward sweeps with x' ranging over all Cartesian basis vectors. The total effort for this simpleminded procedure is bounded by $3n$ times the cost of evaluating $f(x)$ once, an upper bound much worse than that for a straightforward difference approximation. An obvious source of waste is that for sparse x' some or many of the values v_j are constant, so that the corresponding assignment (8) could be skipped because the right-hand side vanishes identically. While such selective skipping is not easily implemented, one can simultaneously update all nonzero partials, namely, a sparse representation of the gradients

$$\nabla_x v_j \quad \equiv \quad (\partial v_j / \partial x_i)_{i=1\ldots n} \quad \in \quad \mathbf{R}^n \tag{10}$$

by the recurrence

$$\nabla_x v_j \quad += \quad c_{ji} \cdot \nabla_x v_i \quad \text{for} i \in \mathcal{P}_j \quad . \tag{11}$$

Here the gradients $\nabla_x v_i = \nabla x_i$ for $i \leq n$ must be defined as Cartesian basis vectors, and all other $\nabla_x v_i$ must be initialized to zero vectors. Excluding the possibility of exact cancellations and assuming that all intermediates v_i with $i \leq n + p$ actually do occur as arguments of some elementary function, one finds that the maximal number of nonzeros in any one of the gradients $\nabla \phi_j$ equals

$$\hat{n} \quad \equiv \quad \max \# \text{ of nonzeros in any row of J} \quad \leq \quad n \quad .$$

Under these reasonable assumptions we can thus conclude that in the sparse forward mode

$$\text{Cost}\{J(x)\}/Cost\{f(x)\} \quad \leq \quad 3\,\hat{n} \quad . \tag{12}$$

This upper bound is still conservative, but much more competitive with sophisticated differencing schemes, for example, the graph coloring approach developed by Coleman and Moré [Cole83a]. In fact, their grouping of columns into $\bar{n} \geq \hat{n}$ mutually independent sets can also be exploited in the forward mode, with each vector ∇v_i being compressed to $\bar{n}$ components. Correspondingly, one can group the rows of the Jacobian into $\bar{m} \geq \hat{m}$ structurally orthogonal sets, and then apply the reverse mode with each adjoint vector $\bar{v}_i$ being compressed to $\bar{m}$ nonzero components. In either case, the number of floating-point operations will be larger than for the corresponding *dynamically* sparse implementation, but there is no overhead for finding the next nonzero component. The relative efficiency of these alternative implementations depends strongly on the computing environment and the particular problem at hand.

Even if the Jacobian is not sparse, the sparse forward mode may work well, provided the evaluation is organized such that the intermediate gradients $\nabla \phi_j$ fill in only towards the very end. This effect has been documented by [Dixo91a] on the Helmholtz energy function. Thus, the sparse mode can be competitive with the following *reverse mode*, which yields gradients at a fixed cost relative to that of evaluating the underlying function but may require significantly more storage.

Using the successor sets

$$\mathcal{S}_i \quad \equiv \quad \{i < j \leq n + m + p \mid i \in \mathcal{P}_j\}\,, \tag{13}$$

we may associate with (8) the adjoint recurrence

$$\hat{v}_i \quad = \quad \sum_{j \in \mathcal{S}_i} c_{ji} \cdot \hat{v}_j \quad \in \quad \mathbf{R} \quad , \tag{14}$$

where the scalar quantities $\hat{v}_i$ represent the *sensitivities*

$$\hat{v}_i \quad \equiv \quad \frac{\partial}{\partial v_i}\, \hat{y}^T f(x) \quad . \tag{15}$$

Here the adjoint weights $\hat{y}_j = \hat{v}_{n+p+j}$ for $j \leq n$ may be chosen arbitrarily but are held constant with respect to the differentiation. We shall show in the next section that these adjoint quantities do, in fact, satisfy the recurrence (14).

By executing the recurrence (14) backwards (i.e., for $i = n+p+m, n+p+m-1, \ldots, n+1$), one obtains the corresponding adjoint $\hat{y} \in \mathbf{R}^m$ from the given vector $\hat{x}^T = \hat{y}^T J(x) \in \mathbf{R}^n$. Since every elementary partial occurs again exactly once, we have similarly to 9

$$\text{Cost}\{\ \hat{x} \equiv \hat{y}^T J(x)\ \} \quad \leq \quad 3\ \text{Cost}\{f(x)\} \quad . \tag{16}$$

By letting $\hat{y}$ range over all Cartesian basis vectors in the range $\mathbf{R}^m$ of f, one obtains the Jacobian in m reverse sweeps of the form (14). Clearly this reverse mode promises to be more efficient than the forward mode if there are many more independent than dependent variables. In particular, one obtains the still somewhat surprising result that single gradient vectors can be computed for a fixed multiple of the cost of evaluating the underlying scalar function. Here we are interested mainly in the case where m like n is of significant size, in which case a $3m$-fold cost penalty for evaluating the Jacobian is probably unacceptable. As in the forward mode, we may also calculate the sensitivities of all dependent variables with respect to each intermediate variable simultaneously, that is, calculate the vectors

$$\bar{v}_j \quad \equiv \quad (\partial y_k / \partial v_j)_{k=1\ldots m} \quad \in \quad \mathbf{R}^m \tag{17}$$

in one reverse sweep. The generally sparse vectors $\bar{v}_j$ satisfy exactly the same recurrence (14) as the scalar adjoints $\hat{v}_j$, which may be evaluated in the slightly more convenient form

$$\bar{v}_j \quad += \quad c_{kj} \cdot \bar{v}_k \quad \text{for} \quad \text{all} j \in \mathcal{P}_k \quad . \tag{18}$$

Here, it is assumed that all $\bar{v}_j$ with $j \leq n + p$ have been initialized to zero, and the $\bar{v}_{n+p+i} = \bar{y}_i$ are by definition constantly equal to Cartesian basis vectors.

Excluding again exact cancellations and assuming that all v_i with $i > n$ depend directly or indirectly on some independent variable, one finds that the maximal number of nonzeros in any one of the vectors $\bar{v}_j$ equals

$$\hat{m} \ \equiv \ \max \#\ \text{of nonzeros in any column of J} \quad \leq \quad m \ .$$

Under these reasonable assumptions, we can conclude that in the reverse mode

$$\text{Cost}\{J(x)\}/\text{Cost}\{f(x)\} \quad \leq \quad 3\,\hat{m} \quad . \tag{19}$$

Hence, one may expect that the reverse mode is more efficient for evaluating a particular Jacobian if the maximal number of nonzeros in any column is smaller than that in any row, and vice versa. As we shall see in Section 4, this rule of thumb appears to be more reliable than a criterion based on the dimensions n and m alone.

A potential drawback of sparse implementations is the need for indirect addressing, which may represent a serious obstacle to vectorization and parallelization. To avoid this effect, one may instead employ Coleman and Moré's technique [Cole83a] of grouping columns into $\bar{n} \geq \hat{n}$ mutually independent sets and then apply the forward mode with each vector ∇v_i being compressed to $\bar{n}$ components. Correspondingly, one can group the rows of the Jacobian into $\bar{m} \geq \hat{m}$ structurally orthogonal sets and then apply the reverse mode with each adjoint vector $\bar{v}_i$ being compressed to $\bar{m} \geq \hat{m}$ nonzero components. In either case, the number of floating-point operations will be larger than for the corresponding *dynamically* sparse implementation, but there is much less overhead. The relative efficiency of these alternative implementations depends strongly on the computing environment and the particular problem at hand. On problems with a few long columns or rows (in terms of nonzeros), the coloring approach is likely to be less efficient.

While the mixed modes to be discussed in the next section are likely to achieve a lower operations count, the sparse forward and to a lesser extent the reverse mode tend to be more economical in terms of storage requirement. If the assignments (1) and (18) are executed for $j = n+1, n+2, \ldots, n+p+m$ and $j = n+p+m, n+p+m-1, \ldots, n+1$, respectively, then the elementary partials c_{ji} occur in exactly the same order forwards and backwards. Moreover, this order is the one in which they are naturally generated during the original evaluation of the elementary functions (1) for $i = n+1, n+1, \ldots, n+p+m$. Hence, the values and indices of the elementary partials c_{ji} can be stored into a sequential data set, which can be paged to and from disk without substantial runtime penalties. Even if this *tape* is small enough to reside in core, the sequential access pattern is likely to reduce the number of cache misses.

The amount of randomly accessed memory needed for the forward or reverse propagation of scalar derivatives can be limited to a small multiple of the storage requirement of the original evaluation program. This separation between a moderately increased, randomly accessed memory and the potentially very large sequential tape has for example been implemented in the C++ package ADOL-C [Grie90a]. The RAM requirements of the dynamically sparse forward and reverse mode are at most $\hat{n} \leq n$ and $\hat{m} \leq m$ larger than that of the corresponding scalar sweeps. This increase in memory from the scalar to the sparse vector mode mirrors exactly the corresponding growth in arithmetic operations. The mixed modes discussed in the next session are likely to require substantially more in core storage than the forward or reverse mode. This is certainly true for our first experimental implementation.

3 Jacobian Accumulation as Elimination of Intermediate Vertices. As discussed by Iri [Irim91a], the relation between the variables v_j can be visualized by a computational graph with the integer vertices $0 < j \leq n+p+m$. We shall use the attributes of corresponding variables v_j and vertices j interchangeably. In particular, we shall refer to independent, intermediate, and dependent vertices or nodes. An arc connects i to j exactly when i belongs to $\mathcal{P}_j$, which means that the variable v_j directly depends on v_i. Hence the index sets $\mathcal{P}_j$ and $\mathcal{S}_i$ defined in the preceding section contain exactly the predecessors and successors of the vertex j, respectively.

With each arc we can associate the elementary partial c_{ji}. These *arc-values* represent multipliers rather than capacities, which are usually associated with arcs in flow networks. Since we are interested only in first derivatives at a particular point, the nature of the elementary functions ϕ_j is no longer of interest once the values c_{ji} of their partials have been obtained. In other words, we may consider the graph as representing the linearization of the composite function $y = f(x)$ at the current argument. The linearized relation (8) may be interpreted as a Kirchhoff-like law that must hold at all intermediate and dependent vertices, while it reduces to the initialization $v_i' = x_i'$ at the independent vertices.

The linear dependency graph defined above is acyclic, because the evaluation of the composite vector function $f(x)$ according to (1) could not be well defined if a pair of intermediate quantities v_j and v_i were mutually dependent on each other. The independent and dependent vertices are minimal and maximal with respect to the partial ordering induced by the acyclic graph on its vertices. There may be other minimal and maximal vertices, which represent constant initializations and computational dead ends, respectively.

As we have noted in the preceding section, the elementary partials c_{ji} (and hence the graph) can be viewed as a representation of the Jacobian $J(x)$ at the given point x. If the graph does not contain any intermediate vertices, it is bipartite, and its arcs represent exactly the nonzero entries of the Jacobian matrix J. Hence, we may try to calculate the Jacobian matrix associated with a general linear dependency graph by successively eliminating all its intermediate vertices without altering the input-output characteristic

between x' and y'. This idea was apparently first published in [Yosh87a].

Suppose we wish to eliminate one particular intermediate vertex j from the graph. This

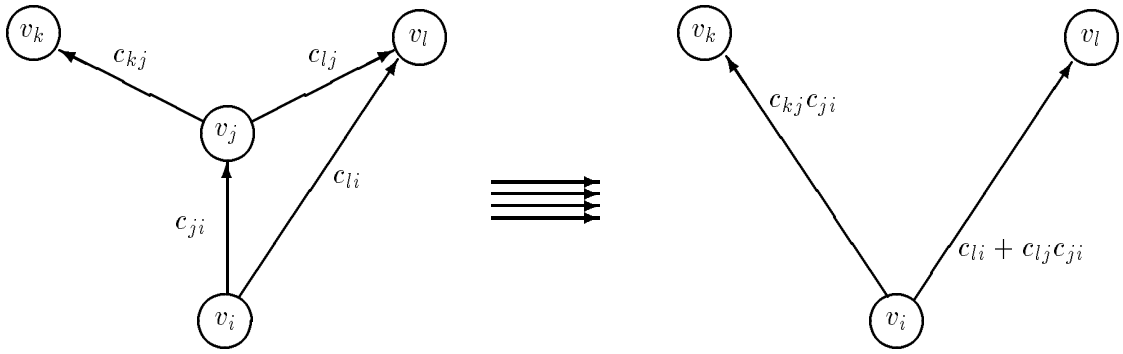

Figure 1: Elimination of intermediate vertex j from dependency graph

is equivalent to eliminating the value v_j' from the algebraic recurrence (8). For all $k \in \mathcal{S}_j$, we may substitute

$$\begin{aligned} v_k' &= c_{kj} \cdot v_j' + \sum_{j \neq i \in \mathcal{P}_k} c_{ki} \cdot v_i' \\ &= \sum_{i \in \mathcal{P}_j} c_{kj} \cdot c_{ji} \cdot v_i' + \sum_{j \neq i \in \mathcal{P}_k} c_{ki} \cdot v_i' \qquad (20) \\ &= \sum_{i \in \tilde{\mathcal{P}}_k} \tilde{c}_{ki} \cdot v_i', \end{aligned}$$

$$(21)$$

where $\tilde{\mathcal{P}}_k \equiv \mathcal{P}_j \cup \mathcal{P}_k - \{j\}$, and

$$\tilde{c}_{ki} = c_{ki} + c_{kj} \cdot c_{ji} \qquad (22)$$

for all index pairs $k \in \mathcal{S}_j, i \in \mathcal{P}_j$. Here we use the convention that $c_{ki} = 0$ if $i \notin \mathcal{P}_i$, in which case $\tilde{c}_{ki} = c_{kj} \cdot c_{ji}$ is the value of a new arc (i.e., we have fill-in). Similarly to the predecessor sets, one has to update the successor sets according to $\tilde{\mathcal{S}}_i \equiv \mathcal{S}_j \cup \mathcal{P}_i - \{j\}$ for all i in the predecessor set $\in \mathcal{P}_j$ of the vertex j being eliminated. Graphically, one simply has to connect all predecessors of j to all its successors and then eliminate j as well as all its incoming and outgoing arcs. After j has been eliminated, we may strip the tildes from the updated quantities and repeat the process until all nonzero arcs directly connect independent to dependent vertices. Thus the successive elimination of all intermediate vertices in a completely arbitrary order must result in the same bipartite graph representing the unique Jacobian. To avoid the tildes, one may write the elimination update in the incremental form

$$c_{ki} \mathrel{+}= c_{kj} \cdot c_{ji} \quad .$$

During the elimination process, each arc $i \to k$ may attain a sequence of values c_{ki}, which depend on the ordering. Of particular interest are the initial and the final values. The nonzero initial arc values are simply elementary partial derivatives. The final value of an arc is attained just before its origin or destination is eliminated, unless it directly connects an independent to a dependent vertex. In the latter case, the final arc value represents a nontrivial entry of the Jacobian. If at any stage of the elimination process an arc is the

only directed path connecting two particular vertices l and k, then the value c_{kl} is final and can be expressed in terms of initial arc values as

$$c_{kl} \quad \equiv \quad \sum_{\mathcal{J}: l \to k} \prod_{i} c_{j_i \, j_{i+1}} \quad , \tag{23}$$

where the paths $\mathcal{J}$ are of the form

$$\mathcal{J} \quad \equiv \quad \{l, j_1, j_2, \ldots, j_i, \ldots, k\} \quad . \tag{24}$$

This relation was apparently first established by Miller and Wrathall [Mill80a] and is also derived in [Irim91a]. Clearly, a separate evaluation of each Jacobian entry by the determinant-like explicit formula (23) would be extremely expensive, as common expressions are ignored and there may be an exponential number of paths.

If for some $k > n$, all arcs c_{ki} with $1 \leq n$ are actually computed, then their final values represent the gradient of the intermediate variable v_k with respect to the independent variables x. This situation arises exactly when all direct and indirect predecessors of k are eliminated before k. Similarly, if for some $i \leq n + p$ all arcs c_{ki} with $k > n + p$ are computed, then they represent the sensitivities of all dependent variables with respect to the intermediate variable v_i. To understand that the sparse forward and reverse modes discussed in the preceding section are special cases of the general elimination procedure considered here, we note that both factors c_{kj} and c_{ji} in the update formula (3) represent final arc values as the vertex j is just being eliminated. Now, if the the intermediate vertices are eliminated in their original order $j = n + 1, \ldots, n + p$, then all left factors c_{kj} are also initial and all right factors c_{ji} represent gradient components. Consequently, the update formula is in fact equivalent to the sparse equation (11). Similarly, if the intermediates are eliminated in the reverse order $j = n+p, n+p-1, \ldots, n+1$, then (3) reduces to the adjoint relation (18). Thus we conclude that the elimination procedure proposed in this section can in every respect be at least as efficient as the classical forward and reverse modes.

4 The Markowitz Heuristic. With $|\cdot|$ denoting cardinality for sets, we find that the elimination of the intermediate vertex j involves

$$\text{mark}(j) \quad \equiv \quad |\mathcal{P}_j| \cdot |\mathcal{S}_j| \tag{25}$$

multiplications, each one followed by an addition or the setting up of a new arc. The elimination of j effects the *Markowitz counts* $\text{mark}(i)$ and $\text{mark}(k)$ of all predecessors and successors of j. For any particular vertex, $\text{mark}(j)$ may increase or decrease as neighbors of j disappear, until j itself is eliminated. The total work for eliminating all intermediates in a particular ordering is equal to the sum

$$\text{Cost}\{J(x)\} \quad = \quad \sum_{n < j \leq n+p} \text{mark}(j) \quad , \tag{26}$$

where $\text{mark}(j)$ is the final Markowitz count at the time when j is being eliminated. Thus it would seem natural to look for an elimination ordering that minimizes this sum of Markowitz counts. Unfortunately, it appears that the exact solution of this combinatorial optimization problem is NP-hard. This conjecture is based on the close relationship to the Gaussian elimination problem considered in [Rose78a].

As is typical in combinatorial optimization, we may have to settle for a heuristic algorithm that is comparatively cheap to implement and that yields, in many cases, nearly optimal results. Here, the obvious greedy strategy is to eliminate a vertex whose Markowitz count is minimal at the current stage. Especially in the beginning, there tend to be many

ties with comparatively low Markowitz counts. We use as a primary tie-breaker strategy the maximization of the sum $|\mathcal{P}_j| + |\mathcal{S}_j|$, which gives the number of arcs that disappear as a result of the elimination of j. In the calculations reported here, the remaining ties were resolved at random. We have chosen test problems of significant size, which show that unidirectional modes can be quite inefficient in comparison to the other and that the Markowitz strategy appears to be consistently superior.

The first problem **Bratu1** is a discretization of the elliptic PDE

$$\Delta u \; + \lambda e^{u/(1+\mu u)} \quad = \quad 0$$

on a cylindrical geometry. Assuming rotational symmetry of solutions and using a piecewise linear discretization on a 60×30 grid, one obtains a system of $n = 1800$ equations in as many unknowns. The computational graph for the nonlinear part of this system contains 49,004 vertices and 58,428 arcs, which means that only about 20% of the elementary operations were binaries. The second test problem **Bratu2** is the same as the first, except that the two parameters λ and μ are now considered as variables, so that the originally sparse and symmetric Jacobian is appended by two dense columns. The computational graph for this underdetermined system contains 56,008 vertices and 81,638 arcs. Finally, we report calculations on the driven **Cavity** problem, a finite-difference discretization of the incompressible Navier-Stokes equation in a rectangle with constant fluid flow across one side. This nonlinear system has $n = 961$ independent variables, and its graph contains 86,939 vertices with 145,422 arcs. The calculations were carried out on a Sun 3 with the automatic differentiation package ADOL-C. Neither the properties of this package nor the computing environment matters, because we report in the following table only the total counts (26), which represent the number of multiplications needed for the accumulation of the Jacobian from the graph.

Table 1: Operation Counts for Jacobian Computations

Problem	Arcs	Forward	Reverse	Markowitz
Bratu1	58,428	43,763	55,006	43,763
Bratu2	81,678	112,047	231,250	83,319
Cavity	145,422	670,716	189,437	177,645

Possibly the most surprising observation is that the unfreezing of the two parameters λ and μ leads to a dramatic deterioration in the performance of the reverse mode. Clearly, the simple rule of thumb that the reverse mode is preferable to the forward mode if $m < n$, and vice versa, does not work in this case. However, the slightly more sophisticated idea of comparing the maximal number of nonzeros $\hat{m}$ and $\hat{n}$ defined in Section 2 points in the right direction. Because of the introduction of dense columns into the Jacobian, the integer $\hat{n}$ and hence the bound on the complexity of the reverse mode jumps from a small number to n, whereas $\hat{m}$, which bounds the complexity of the forward mode, is incremented only by two.

In each of these examples, the accumulation of the Jacobian by the Markowitz scheme and also by the better one of the unidirectional methods requires only about as many multiplications as there are arcs in the computational graph. Since the number of arcs is a good measure of the complexity of the underlying vector function, the Jacobian can be obtained for essentially the same number of floating-point operations. This highly desirable complexity ratio probably does not apply to functions whose Jacobian is dense. On a parallel machine, however, one can simultaneously eliminate sets of vertices that are independent, that is, not directly connected by an arc. In other words, the accumulation problem for the

Jacobian typically has much more (and certainly never less) concurrency than the underlying nonlinear evaluation problem.

By measuring complexity purely in terms of floating-point operations for the actual elimination process, we have neglected several other significant costs. First, there is the overhead of determining an appropriate elimination ordering according to the Markowitz rule or some other heuristic. As in the case of sparse linear system solving [Duff86a], this overhead cost is highly dependent on the implementation and can be substantially reduced by suitably relaxing the pure selection criterion. Fortunately, in contrast to the Gaussian elimination case, numerical stability is not a problem, since accumulating the Jacobian requires only multiplications and additions. Therefore, once a suitable elimination ordering has been determined, it can be mechanically applied at a sequence of points which might be generated by an iterative scheme or discrete time integration.

5 Conclusion and Discussion. The task of computing the Jacobian of a vector function defined by a computer program is reduced to the problem of successively eliminating intermediate vertices in a linearized computational graph. This accumulation procedure can be viewed mathematically as a generalization of the chain rule. For general vector functions, the accumulation of their Jacobians with minimal operation count is conjectured to be an NP-hard combinatorial optimization problem. The standard forward and reverse modes of automatic differentiation are limiting cases of the more general elimination procedure. These algorithms can be implemented in a dynamically sparse or statically compressed form. The selection of an elimination ordering by a greedy strategy based on the Markowitz count is found to yield significantly lower operations counts on two discretizations of partial differential equations. Probably the Markowitz approach requires significantly more core space than the reverse mode, which in turn is likely to require much more sequentially accessed disk space than the forward mode.

We hope that the strengths of the various accumulation methods can be combined in a hybrid scheme that performs Markowitz based eliminations in a core window that sweeps back and forth across the computational graph. Eventually, such an approach could be modified to compute Newton steps directly without forming the Jacobian, as suggested in [Muro87b] and [Grie90a]. In cases like our test problem **Bratu1** where the vector function f is the gradient of a scalar function, its evaluation can be programmed such that the corresponding computational graph is symmetric, as described by Dixon. [Dixo91a]. Then it would seem natural to maintain this symmetry during the elimination process for accumulating the Hessian or for directly calculating the Newton step.

References

[Cole83a] T. F. Coleman and J. J. Moré (1983) *Estimation of sparse Jacobian matrices and graph coloring problems* SIAM Journal on Numerical Analysis, Vol. 16, pp. 368–375.

[Dixo91a] L. W. C. Dixon *The use of automatic ifferentiation to calculate Hessian matrices and Newton steps*

[Duff86a] I. S. Duff, A. M. Erisman, and J. K. Reid (1986) *Direct methods for sparse matrices* Oxford Science Publications, Clarendon Press, Oxford.

[Grie89a] A. Griewank (1989) *On automatic differentiation*, in Mathematical Programming: Recent Developments and Applications, ed. M. Iri and K. Tanabe, Kluwer Academic Publishers, pp. 83–108.

[Grie90c] A. Griewank (1990) *Direct Calculation of Newton Steps without Accumulating Jacobians* in *Large-Scale Numerical Optimization,* T. F. Coleman and Yuying Li, eds., SIAM, pp. 115–137.

[Grie90a] A. Griewank, D. Juedes, and J. Srinivasan (1990) *ADOL-C, a package for the automatic differentiation of algorithms written in C/C++*, Preprint MCS-180-1190, Argonne National Laboratory, Argonne, Illinois.

[Grie91c] A. Griewank (1991) *Automatic Evaluation of First- and Higher-Derivative Vectors,* Proceedings of the Conference at Würzburg, Aug. 1990, Bifurcation and Chaos: Analysis, Algorithms, Applications, R. Seydel, F. W. Schneider,T. Küpper, and H. Troger, eds., Basel, Birkhäuser, pp. 124–137.

[Irim91a] M. Iri (1991) *Automatic differentiation and rounding error estimation - overview and history* This volume pp.

[Mill80a] W. Miller and C. Wrathall (1980) *Software for Roundoff Analysis of Matrix Algorithms*, Academic Press, New York.

[Muro87b] K. Murota (1987) *Menger-decomposition of a graph and its application to the structural analysis of a large-scale system of equations.* Discrete Applied Mathematics, Vol. 17 , pp. 107–134.

[Rose78a] D. J. Rose and R. E. Tarjan (1978) *Algorithmic aspects of vertex elimination on directed graphs.* SIAM J. A. M., Vol. 34, pp. 177–197.

[Yosh87a] T. Yoshida (1987). *Derivation of a Computational Process for Partial Derivatives of Functions Using Transformations of a Graph.* Transactions of IPSJ, Vol. 11, pp. 1112–1120.