Proceedings of ALGORITMY 2000
Conference on Scientific Computing, pp. 320-329

AN ENHANCED MARKOWITZ RULE FOR ACCUMULATING JACOBIAN MATRICES EFFICIENTLY

UWE NAUMANN *

Abstract. Jacobian matrices can be accumulated using either the forward or reverse mode of Automatic Differentiation. Alternatively, derivative code can be generated to compute the Jacobian matrix directly at the current argument. The minimization of the corresponding number of arithmetic operations leads to a computationally hard combinatorial optimization problem. A new powerful heuristic for its approximate solution will be presented. The resulting codes lead to a speedup of three and more for most problems.

Key words. Optimized Jacobian code, combinatorial optimization, vertex elimination.

AMS subject classifications. 90C27, 65Y99.

$$
\begin{aligned}
h_1 &= \tan(x_0)/(x_1 \cdot x_2) \\
h_2 &= x_1 \cdot x_2 \cdot \sinh(x_3) \\
\pi &= 4 \cdot \arctan(1) \\
y_0 &= \log(4.3) \cdot h_1 \cdot \sin(h_2) \\
y_1 &= \cos(h_1)/\exp(h_2) \\
y_2 &= \sqrt{h_1} \cdot \pi \cdot h_2
\end{aligned}
$$

FIG. 1.1. *Vector Function*

1. Motivating Example.

Jacobian matrices (or Jacobians) of vector functions which are given in form of a computer program are widely required in numerical computations such as simulation or optimization. Prior to a formal introduction to Jacobian accumulation techniques let us have a look at a simple example. Consider the system of non-linear equations shown in FIG. 1.1. It describes a function F mapping $\mathbb{R}^n \ni \mathbf{x} = (x_i)_{i=0..(n-1)}$, $n = 4$, onto $\mathbb{R}^m \ni \mathbf{y} = (y_j)_{j=0..(m-1)}$, $m = 3$. We denote the Jacobian of F by $F' = (\partial y_j / \partial x_i)_{i=0..3}^{j=0..2}$. After performing certain optimizations (e.g. elimination of common subexpressions) this program may be transformed into a *code list* by assigning the results of the elementary operations ($\{+, -, /, \cdot\}$) and intrinsic functions ($\{\sin, \exp, \sqrt{\ }, \ldots\}$) to unique intermediate variables $v_k = \varphi_k(v_{l_1}..v_{l_k})$ for $k = 1..p$ and arguments $v_{l_1}..v_{l_k}$ of φ_k. In most cases we are interested in computing Jacobian-vector products of the forms $\dot{\mathbf{y}} = F' \cdot \dot{\mathbf{x}}$ or $\bar{\mathbf{x}} = \bar{\mathbf{y}}^T \cdot F'$. *Automatic Differentiation (AD)* [Gri00] provides a fast and effective way to evaluate these products with machine accuracy based on the partial derivatives of the elementary operations and intrinsic functions and by simply applying the chain rule. This approach is completely different from the numerical approximation of derivatives through finite difference quotients.

Given the code list of F the function value and the partial derivatives of both the elementary operations and intrinsic functions at a given argument $\mathbf{x}_0$ can be calculated by a single *recording* [Gri00] evaluation of F. This is illustrated on the l.h.s. of FIG. 1.3. Setting $v_{i-3} = x_i$, $i = 0..3$, and $y_i = v_{i+12}$, $y = 0..2$, the intermediate function values are computed in parallel with the corresponding partial derivatives. The relation between the variables in the code list can be visualized by a directed acyclic computational graph (c-graph) with integer vertices (FIG. 1.2). In the *linearized* c-graph the partial derivatives are assumed to be attached as labels to the corresponding edges.

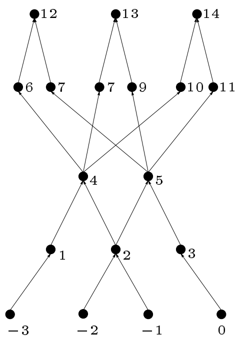

FIG. 1.2. *c-graph*

AD provides two basic modes for computing $\dot{\mathbf{y}}$ and $\bar{\mathbf{x}}$. They are referred to as the *forward* and *reverse* modes [Wen64],

*Computer Science, University of Hertfordshire, Hatfield, UK (U.1.Naumann@herts.ac.uk).

Recording	**Adjoint**	**Jacobian**
$c_{i,j}=0,\ (i,j=-3..14,\ i<j)$ $v_{i-3}=x_i,\ (i=0..3)$ $v_1=\tan(v_{-3})$ $c_{1-1}=1+\tan(v_{-3})^2$ $v_2=v_{-2}\cdot v_{-1}$ $c_{2,-2}=v_{-1};\ c_{2,-1}=v_{-2}$ $v_3=\sinh(v_0)$ $c_{3,0}=-\cosh(v_0)$ $v_4=v_1/v_2$ $c_{4,1}=1/v_2;\ c_{4,2}=-v_1/v_2^2$ $v_5=v_2\cdot v_3$ $c_{5,2}=v_3;\ c_{5,3}=v_2$ $v_6=\log(4.3)\cdot v_4$ $c_{6,4}=\log(4.3)$ $v_7=\sin(v_5)$ $c_{7,5}=\cos(v_5)$ $v_8=\cos(v_4)$ $c_{8,4}=-\sin(v_4)$ $v_9=\exp(v_5)$ $c_{9,5}=v_9$ $v_{10}=\sqrt{v_4}$ $c_{10,4}=1/(2v_{10})$ $v_{11}=4\cdot\arctan(1)\cdot v_5$ $c_{11,5}=4\cdot\arctan(1)$ $v_{12}=v_6\cdot v_7$ $c_{12,7}=v_6;\ c_{12,6}=v_7$ $v_{13}=v_8/v_9$ $c_{13,8}=1/v_9;\ c_{13,9}=-v_8/v_9^2$ $v_{14}=v_{10}\cdot v_{11}$ $c_{14,10}=v_{11};\ c_{14,11}=v_{10}$ $y_i=v_{i+12},\ \ (y=0..2)$	$\bar{v}_{i+12}=\bar{y}_i,\ \ (i=0..2)$ $\bar{v}_i=0,\ \ (i=-3..11)$ $\bar{v}_{11}\ {+}{=}\ c_{14,11}\cdot\bar{v}_{14}$ $\bar{v}_{10}\ {+}{=}\ c_{14,10}\cdot\bar{v}_{14}$ $\bar{v}_9\ {+}{=}\ c_{13,9}\cdot\bar{v}_{13}$ $\bar{v}_8\ {+}{=}\ c_{13,8}\cdot\bar{v}_{13}$ $\bar{v}_7\ {+}{=}\ c_{12,7}\cdot\bar{v}_{12}$ $\bar{v}_6\ {+}{=}\ c_{12,6}\cdot\bar{v}_{12}$ $\bar{v}_5\ {+}{=}\ c_{11,5}\cdot\bar{v}_{11}$ $\bar{v}_4\ {+}{=}\ c_{10,4}\cdot\bar{v}_{10}$ $\bar{v}_5\ {+}{=}\ c_{9,5}\cdot\bar{v}_9$ $\bar{v}_4\ {+}{=}\ c_{8,4}\cdot\bar{v}_8$ $\bar{v}_5\ {+}{=}\ c_{7.5}\cdot\bar{v}_7$ $\bar{v}_4\ {+}{=}\ c_{6,4}\cdot\bar{v}_6$ $\bar{v}_3\ {+}{=}\ c_{5,3}\cdot\bar{v}_5$ $\bar{v}_2\ {+}{=}\ c_{5,2}\cdot\bar{v}_5$ $\bar{v}_2\ {+}{=}\ c_{4,2}\cdot\bar{v}_4$ $\bar{v}_1\ {+}{=}\ c_{4,1}\cdot\bar{v}_4$ $\bar{v}_0\ {+}{=}\ c_{3,0}\cdot\bar{v}_3$ $\bar{v}_{-1}\ {+}{=}\ c_{2,-1}\cdot\bar{v}_2$ $\bar{v}_{-2}\ {+}{=}\ c_{2,-2}\cdot\bar{v}_2$ $\bar{v}_{-3}\ {+}{=}\ c_{1,-3}\cdot\bar{v}_1$ $\bar{x}_i=\bar{v}_{i-3},\ \ (i=0..3)$	$c_{4,-3}\ {+}{=}\ c_{4,1}\cdot c_{1,-3}$ $c_{5,0}\ {+}{=}\ c_{5,3}\cdot c_{3,0}$ $c_{12,4}\ {+}{=}\ c_{12,6}\cdot c_{6,4}$ $c_{12,5}\ {+}{=}\ c_{12,7}\cdot c_{7,5}$ $c_{13,4}\ {+}{=}\ c_{13,8}\cdot c_{8,4}$ $c_{13,5}\ {+}{=}\ c_{13,9}\cdot c_{9,5}$ $c_{14,4}\ {+}{=}\ c_{14,10}\cdot c_{10,4}$ $c_{14,5}\ {+}{=}\ c_{14,11}\cdot c_{11,5}$ $\underline{c_{12,-3}}\ {+}{=}\ c_{12,4}\cdot c_{4,-3}$ $\underline{c_{13,-3}}\ {+}{=}\ c_{13,4}\cdot c_{4,-3}$ $\underline{c_{14,-3}}\ {+}{=}\ c_{14,4}\cdot c_{4,-3}$ $c_{12,2}\ {+}{=}\ c_{12,4}\cdot c_{4,2}$ $c_{13,2}\ {+}{=}\ c_{13,4}\cdot c_{4,2}$ $c_{14,2}\ {+}{=}\ c_{14,4}\cdot c_{4,2}$ $\underline{c_{12,0}}\ {+}{=}\ c_{12,5}\cdot c_{5,0}$ $\underline{c_{13,0}}\ {+}{=}\ c_{13,5}\cdot c_{5,0}$ $\underline{c_{14,0}}\ {+}{=}\ c_{14,5}\cdot c_{5,0}$ $c_{12,2}\ {+}{=}\ c_{12,5}\cdot c_{5,2}$ $c_{13,2}\ {+}{=}\ c_{13,5}\cdot c_{5,2}$ $c_{14,2}\ {+}{=}\ c_{14,5}\cdot c_{5,2}$ $\underline{c_{12,-2}}\ {+}{=}\ c_{12,2}\cdot c_{2,-2}$ $\underline{c_{13,-2}}\ {+}{=}\ c_{13,2}\cdot c_{2,-2}$ $\underline{c_{14,-2}}\ {+}{=}\ c_{14,2}\cdot c_{2,-2}$ $\underline{c_{12,-1}}\ {+}{=}\ c_{12,2}\cdot c_{2,-1}$ $\underline{c_{13,-1}}\ {+}{=}\ c_{13,2}\cdot c_{2,-1}$ $\underline{c_{14,-1}}\ {+}{=}\ c_{14,2}\cdot c_{2,-1}$

FIG. 1.3. *Approaches to Jacobian Accumulation*

[Gri92] and represent two special choices of how to apply the chain rule to the code list. The reverse mode generates *adjoint code* shown in the middle of FIG. 1.3. Existing source code transformation AD tools [ADIFOR], [ODYSSEE], [TAMC] will deliver a more compact representation of the adjoint code. The explicit decomposition in FIG. 1.3 is merely used to make things easier to understand. For more advanced information on state-of-the-art adjoint code generation refer to [TAMC] or [Fau00]. After the initialization of $\bar{\mathbf{y}}$ the adjoint code computes $\bar{\mathbf{x}}=\bar{\mathbf{y}}^T\cdot F'(\mathbf{x}_0)$. Consequently, the Jacobian itself can be accumulated by setting $\bar{\mathbf{y}}$ equal to the Cartesian basis vectors $\mathbf{e}_i \in I\!R^3$, $i = 0..2$, successively.

Once all partial derivatives of the elementary operations and intrinsic functions are available the accumulation of $F' = F'(\mathbf{x}_0)$ may, in general, involve a varying number of evaluations of $\bar{v}_i = \bar{v}_i + c_{j,i} \cdot \bar{v}_j$, for $i, j \in \{-3..14\}$, $i \prec j$. (In FIG. 1.3 the more compact C-style notation "+=" is used instead.) Expressions of this type will be referred to as `maf`'s (multiply-add-fused). Our objective will be to minimize the number of `maf`'s required for the accumulation of F'.

The computation of F' by reverse propagation of the corresponding Cartesian basis vectors involves $m(p+n) = 45$ `maf`'s. Provided that trivial multiplications by 0 or 1 are not carried out this number could be decreased to 33. Alternatively one may choose to generate explicit *Jacobian code* which accumulates F' at the current argument $\mathbf{x}_0$ directly. This is illustrated on the r.h.s. of FIG. 1.3 where all entries of F' are underlined. In fact, this code is optimal regarding the number of `maf`'s performed, which is equal to 26 (compare SEC. 3).

2. Accumulation of Jacobians. Let

$$F' = F'(\mathbf{x}_0) = \left(\frac{\partial y_i}{\partial x_j}(\mathbf{x}_0)\right)_{j=0..(n-1)}^{i=0..(m-1)}$$

denote the Jacobian matrix of a non-linear vector function

$$F : I\!R^n \supseteq D \to I\!R^m : \quad \mathbf{x} \mapsto \mathbf{y} = F(\mathbf{x})$$

evaluated at a given argument $\mathbf{x}_0$ as in SEC. 1. The runtime of numerous numerical algorithms is dominated by the time it takes to accumulate F' or to evaluate products of the form $(I\!R^{m\times l_1} \ni) \dot{Y} = F'\dot{X}$ and $(I\!R^{l_2\times n} \ni) \bar{X} = \bar{Y}F'$, which represent the matrix versions of what was introduced in SEC. 1. They can be computed using the forward and reverse *vector* mode of AD [Gri00], respectively. This paper will discuss a new powerful heuristic for accumulating F' more efficiently. Unquestionably, the ideas will also be useful for computing higher order derivative tensors. AD will be looked at from the point of view of graph theory and combinatorial optimization.

F is assumed to be given as a computer program which decomposes into a sequence of scalar *elemental* functions $(I\!R \ni)\ v_j = \varphi_j(v_i)_{i\in P_j}$ where $j = 1..q$ and $P_j \subseteq \{(1-n)..p\}$, $p = q - m$. P_j is the set of indices of the arguments of φ_j and we denote its cardinality by $|P_j|$. Within F we distinguish between three types of variables $V = X \cup Z \cup Y$, referred to as independent $(X \equiv \{v_{1-n}..v_0\})$, intermediate $(Z \equiv \{v_1..v_p\})$, and dependent $(Y \equiv \{v_{p+1}..v_q\})$. We set $x_{i-1} \equiv v_{i-n}$, $i = 1..n$, and $y_{j-1} \equiv v_{p+j}$, $j = 1..m$. The direct dependence of v_j on v_i is denoted by $i \prec j$ with $\prec^*$ representing the transitive closure of this relation. A numbering $\mathcal{I} : V \to \{(1-n)..q\}$ of the variables of F is expected to be consistent, i.e. it must induce a topological order with respect to dependence as $i \prec^* j \Rightarrow \mathcal{I}(v_i) < \mathcal{I}(v_j)$.

Since the differentiation of F is based on the differentiability of its elemental functions it will be assumed that the φ_j, $j = 1..q$, have jointly continuous partial derivatives

$$c_{ji} \equiv \frac{\partial}{\partial v_i}\varphi_j(v_k)_{k\in P_j}, \quad i \in P_j, \tag{2.1}$$

on open neighborhoods $\mathcal{D}_j \subset I\!R^{n_j}$, $n_j \equiv |P_j|$, of their domain. Thus, the accumulation of F' can be regarded as a sequences of transformations in $I\!R^{(n+q)}$ applied to the *extended Jacobian*

$$\mathbb{R}^{(n+q)\times(n+q)} \ni B = B(\mathbf{x}_0) = (c_{ji})_{j,i=1-n}^{q} = \begin{cases} c_{ji} & \text{if } i \prec j \\ 0 & \text{otherwise} \end{cases}$$

such that, finally, the intersection of B's first n columns with its last m rows contains F'. Moreover, the order in which these transformations are applied to B should minimize the number of `maf`'s carried out. The above sequence of transformations on B may be regarded as the successive elimination of intermediate variables from F.

PROPOSITION 1. *Let*

$$\mathcal{E}_j(B) = \mathcal{E}_j B = B + B\mathbf{e}_j\mathbf{e}_j^T B - B\mathbf{e}_j\mathbf{e}_j^T - \mathbf{e}_j\mathbf{e}_j^T B. \tag{2.2}$$

Then

$$F' = Q_m \left(\prod_{j\in\{1..p\}} \mathcal{E}_j B \right) P_n^T. \tag{2.3}$$

$P_n \equiv [I_n, 0, \ldots, 0] \in \mathbb{R}^{n\times(n+q)}$, $Q_m \equiv [0, \ldots, 0, I_m] \in \mathbb{R}^{m\times(n+q)}$ *and* I_n *and* I_m *are the identity matrices in* $\mathbb{R}^{n\times n}$ *and* $\mathbb{R}^{m\times m}$, *respectively.* ◀

PROOF: The proof follows immediately from the *chain rule*:

$$\frac{\partial y_{j-p-1}}{\partial x_{i+n-1}} = \frac{\partial v_j}{\partial v_i} = \sum_{k\in P_j} c_{jk}\frac{\partial v_k}{\partial v_i}, \quad j = (p+1)..q, \quad i = (1-n)..0. \tag{2.4}$$

The c_{jk} are the partial derivatives defined by EQN. (2.1). In order to prove PROP. 1 we need to show that EQN. (2.4) can be transformed into explicit expressions for all entries of F'. This is done by eliminating all terms containing intermediate variables. We will sketch the procedure by eliminating the dependence of y_{j_1-p-1}, $j_1 \in \{(p+1)..q\}$, on some v_{k_1}, $k_1 \in P_{j_1}$. Starting with the isolation of the term, which is to be eliminated, followed by the substitution of the expression corresponding to $\frac{\partial v_{k_1}}{\partial x_i}$, $i = 0..(n-1)$, by the chain rule we get:

$$\left[\frac{\partial v_{j_1}}{\partial x_i}\right]^T_{i=0..(n-1)} \dot{\mathbf{x}} = \left(c_{j_1k_1}\left[\frac{\partial v_{k_1}}{\partial x_i}\right]^T_{i=0..(n-1)} + \sum_{k_1\neq k\in P_{j_1}} c_{j_1k}\left[\frac{\partial v_k}{\partial x_i}\right]^T_{i=0..(n-1)} \right)\dot{\mathbf{x}}$$

$$= \left(\sum_{k\in P_{k_1}} c_{j_1k_1}c_{k_1k}\left[\frac{\partial v_k}{\partial x_i}\right]^T_{i=0..(n-1)} + \sum_{k_1\neq k'\in P_{j_1}} c_{j_1k'}\left[\frac{\partial v'_k}{\partial x_i}\right]^T_{i=0..(n-1)} \right)\dot{\mathbf{x}}.$$

It follows a new expression for calculating $\dot{y}_{j_1-p-1}$, $j_1 = (p+1)..q$, which does not depend on $\dot{v}_{k_1}$, i.e. $\dot{v}_{j_1} = \sum_{k\in\tilde{P}_{j_1}} \tilde{c}_{j_1k}\dot{v}_k$ where

$$\tilde{P}_{j_1} \equiv P_{k_1} \cup P_{j_1} \setminus \{k_1\} \quad \text{and} \quad \tilde{c}_{j_1k} \equiv c_{j_1k} + c_{j_1k_1}c_{k_1k}. \tag{2.5}$$

$B + B\mathbf{e}_j\mathbf{e}_j^T B$ is equivalent to the elimination of the dependencies of v_j on all $v_i \prec v_j$ which implies the elimination of the dependencies of all v_k with $v_j \prec v_k$ on v_j. This must lead to the deletion of both the j-th column $(B\mathbf{e}_j\mathbf{e}_j^T)$ and the j-th row $(\mathbf{e}_j\mathbf{e}_j^T B)$ of B. The repeated application of this argument to all intermediate variables proves PROP. 1. ■

2.1. Computational Graphs. The computational graph $\mathbf{G} = (V, E)$ of F is a directed acyclic graph with $V = \{i | v_i \in F\}$ and $(i, j) \in E$ if $i \prec j$. We assume $\mathbf{G}$ to be linearized in the sense of SEC. 1, i.e. (i, j) is labelled with c_{ji}.

$\mathcal{E}_j$ represents the elimination of j from $\mathbf{G}$. Graphically, this is equivalent to connecting all predecessors of j ($i \in P_j$) with all its successors ($k \in S_j$) followed by updating the existing or generating new edge labels and, finally, the deletion of j. FIG. 2.1 is to illustrate this showing the elimination of 4. In correspondence with the chain rule we multiply the labels of successive edges (i, j) and (j, k) whereas we add the labels of parallel edges having both the same source and the same target. The addition of values of parallel edges will be referred to as *absorption*. Consequently, the elimination of an intermediate vertex j involves $\mu_j = |P_j| \cdot |S_j|$ scalar multiplications. μ_j is called the *Markowitz degree* of j [GrRe91]. $|P_j|$ and $|S_j|$ denote the numbers of predecessors and successors of j in $\mathbf{G}$, respectively.

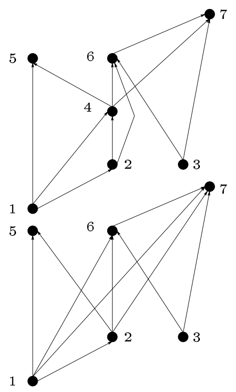

FIG. 2.1. 4

2.2. A combinatorial optimization problem. The expression $j \in \{1..p\}$ in EQN. (2.3) indicates that any of the $p!$ distinct orders in which the p intermediate variables can be eliminated will lead to the desired result in the form of F'. However, the computational costs in terms of the numbers of `maf`'s actually performed may vary drastically.

Example. The following function

$$y = F(\mathbf{x}) = \prod_{i=0}^{n-1} x_i$$

is due to Speelpenning [Spe80] and it will be used here to illustrate the savings that could become possible by choosing one particular elimination sequence instead of another (in the present case we will be looking at a factor of nearly 13). F could be evaluated by a program containing 48 intermediate variables for $n = 50$. We consider two alternative approaches to accumulating the gradient of F : the *forward vertex elimination mode* ($\mathbf{F}$) given by

$$F' = Q_m \left(\prod_{j=1..p} \mathcal{E}_j B \right) P_n^T$$

and the *reverse vertex elimination mode* ($\mathbf{R}$):

$$F' = Q_m \left(\prod_{j=p..1} \mathcal{E}_j B \right) P_n^T .$$

Eliminating the dependence of y on all intermediate variables in reverse order ($\mathbf{R}$) is optimal on this problem requiring 96 `maf`'s. Notice, that $\mathbf{F}$ performs 1224 `maf`'s.

Let M be a method for eliminating the intermediate variables in a certain order. We assume M to be deterministic. Let s_i, $i = 0..p$, denote permutations of i elements from $\{1..p\}$ such that $s_i \subset s_{i+1}$ for $i = 0..(p-1)$, i.e. if $s_k = (j_1, \ldots, j_k)$ then $s_k \subset s_{k+1} = (j_1, \ldots, j_k, j_{k+1})$. For a deterministic M j_{k+1} is defined uniquely by s_k. The s_k, $k = 0..p$, are also called *stages* of the elimination procedure.

Let $\mu_j^k = \mathtt{MAF}(\mathcal{E}_j A)$ where $A = \prod_{i \in S, S \subseteq \{1..p\} \setminus \{j\}} \mathcal{E}_i B$. A is the result of applying a set of transformations not including $\mathcal{E}_j$ to B leading to a stage s_k in the elimination procedure. According to EQN. (2.3) the Markowitz degree μ_j^k of v_j at stage s_k is equal to the product of the number of non-zero elements in the j-th row and the number of non-zero elements in the j-th column of A. The sum of the Markowitz degrees of all intermediate variables at the times of their respective elimination, i.e. $\mu = \mu(s_p) = \sum_{j=i_1}^{i_p} \mu_j^k$ where k is increased from 0 to $p-1$ correspondingly, is called the *overall Markowitz degree* of the elimination sequence s_p.

Our objective is to solve the *Optimal Vertex Elimination* (OVE) problem, which is about finding a method M such that $\mathtt{MAF}_{F'}(M) \to \min$ for arbitrary F. This is equivalent to minimizing the overall Markowitz degree. Since the OVE problem is conjectured to be NP-complete [GrRe91], [GaJo79] we will search for heuristics to solve it approximately.

3. Heuristics. Let $v_i < v_j$ denote the situation where a variable v_i is eliminated before v_j. Since our objective is to minimize the cost of computing F' it certainly makes sense to think about how cheaply a particular intermediate variable v_i can possibly be eliminated. A greedy approach to this problem would be to order all intermediate variables increasingly by their Markowitz degrees. At a particular stage s_j, $j \in \{0..p\}$, the variable with the lowest Markowitz degree is eliminated. In order to make this approach deterministic it has to be combined with at least one tie-break criterion. This approach is known as the *Lowest-Markowitz-Degree-First* (**LM**) strategy and has been proposed in [GrRe91] for the approximate solution of the OVE problem. As a heuristic for the minimization of the fill-in generated during the solution of sparse systems of linear equations the adaption to the OVE problem represents a robust and easy to implement way of calculating often nearly optimal elimination sequences. Formally **LM** is given by

$$s_k \in \{1, \ldots, p\} : \quad v_i < v_j \quad \Leftrightarrow \quad \mu_i^k \; < \; \mu_j^k. \tag{3.1}$$

At each stage the number of predecessors and successors of (possibly) every vertex in $\mathbf{G}$ has to be computed, which results in a complexity of $O(|V|^2)$. However, numerous tests showed that, in general, **LM** does not deliver optimal elimination sequences.

DEFINITION 1. *Let* $\mathbf{G} = (V, E)$ *be a c-graph. A variable* $v_j \in Y \cup Z$ *has the* **in-dependency degree** $\mathtt{id}_j = k$ *if there are paths connecting a maximum of* k *independent vertices with* j *in* $\mathbf{G}$. *A variable* $v_j \in X \cup Z$ *has the* **out-dependency degree** $\mathtt{od}_j = k$ *if a maximum of* k *dependent vertices are reachable from* j *in* $\mathbf{G}$. *For intermediate variables* $v_j \in Z$ *we define the* **dependency degree** *as* $\mathtt{dd}_j \equiv \mathtt{id}_j \cdot \mathtt{od}_j$.

The dependency degree $\mathtt{dd}_j$ of an intermediate variable v_j is equal to the Markowitz degree at which v_j would be eliminated at stage $s_p = s_p(M)$. To improve the **LM** heuristic the Markowitz degree will be evaluated relative to the dependency degree, which is invariant with respect to different vertex elimination sequences. Once we have tabulated it for every variable, which can be done at a cost of $O(|V|(|V| + |E|))$ [MeNä96], we can look it up at every stage s_i, $i = 0..p$. The computation of this *reachability matrix* dominates the entire calculation. As an instance of a dependency degree based heuristic the *Lowest-Extended-Markowitz-Degree-First* heuristic (**EM**) is defined as follows:

$$s_k \in \{1, \ldots, p\} : \quad v_i < v_j \quad \Leftrightarrow \quad \mu_i^k - \mathtt{dd}_i \; < \; \mu_j^k - \mathtt{dd}_j.$$

EM is more expensive than **LM** due to the computation of $\mathtt{id}_j$ and $\mathtt{od}_j$ for all intermediate variables. In many cases it delivers better elimination sequences which more than compensate the additional effort. Although, in general, **EM** does not deliver an optimal elimination sequence, it turns out to be the most consistent strategy compared with **F**, **R**, and **LM**. Let us have a closer look at the behavior of **EM** and the ideas behind it:

1. **EM** extends **LM** by an implicit tie-breaker. If at some stage s_k of the elimination procedure two variables v_i and v_j have the same (lowest) Markowitz degree $\mu_i^k = \mu_j^k$ then the one with the higher dependency degree is eliminated first. The idea behind this decision is that if $\mathtt{dd}_i > \mathtt{dd}_j$ then the probability that the Markowitz degree of v_i will be increased during the rest of the elimination procedure is higher than for v_j. So, one should use the opportunity to eliminate v_i cheaply.

2. Obviously, v_i should be eliminated before v_j if $\mu_i^k < \mu_j^k$ and $\mathtt{dd}_i > \mathtt{dd}_j$. Not only is the Markowitz degree of v_i lower, the probability of it being increased is also higher than for v_j.

3. $\mu_i^k < \mu_j^k$ and $\mathtt{dd}_i < \mathtt{dd}_j$ and $\mu_i^k - \mathtt{dd}_i < \mu_j^k - \mathtt{dd}_j$ yields the elimination of v_i before v_j. This case as well as the one before let **EM** inherit both strengths and weaknesses of **LM** as it behaves exactly in the same way.

4. If $\mu_i^k < \mu_j^k$ and $\mathtt{dd}_i < \mathtt{dd}_j$ such that $\mu_i^k - \mathtt{dd}_i > \mu_j^k - \mathtt{dd}_j$ **EM** would choose to eliminate v_j before v_i. Therefore $\mathtt{dd}_j$ must be much larger than $\mathtt{dd}_i$ to satisfy the above inequality. Consequently, the possible increase of the Markowitz degree of v_j is larger. To avoid taking any chances v_j should be eliminated. This decision is based on a *look ahead* to what could possibly happen during the rest of the elimination procedure. This "guess" and the tie-breaking feature may be regarded as the advantages of **EM** over **LM**.

Reconsider the motivating example from SEC. 1. FIG. 3.1 shows the computational graph after the pre-elimination of 8 vertices with both a single predecessor and successor at the optimal cost of 1 `maf` each. Thus, we are taken to stage s_8 with an overall Markowitz degree of $\mu(s_8) = 8$. There are 3 intermediate vertices left. **LM** eliminates v_2 (4 `maf`'s) followed by v_4 and v_5 (9 `maf`'s each), resulting in $\mathtt{MAF}_{F'}(\mathbf{LM}) = 30$. **EM** would *look ahead* to see that the Markowitz degrees of v_4 and v_5 are more likely to be increased than the one of v_2 due to $6-9 < 4-6$. If **F** was the primary tie-breaker of **EM** v_4 would be eliminated first. Hence $\mu_5^9 - \mathtt{dd}_5 < \mu_2^9 - \mathtt{dd}_2$ $(6-9 < 8-6)$ at s_9. Consequently, v_5 would be eliminated next followed by v_2 and $\mathtt{MAF}_{F'}(\mathbf{EM}) = 26$.

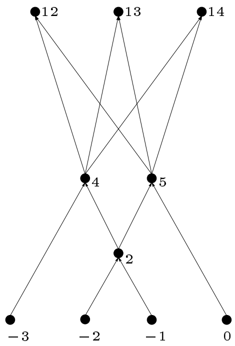

FIG. 3.1. $\mathbf{G}(s_8)$

4. Numerical Results, Summary, Outlook.

4.1. Case Study 1: Speelpenning's function. Both **R** and **EM** are optimal for $y = \prod_{i=0}^{n-1} x_i$ [Spe80]. **LM** has to rely on its primary tie-breaker. Only if it was **R** the above would also apply to **LM**. **EM** is able to solve the problem in any case as $\mathtt{dd}_i$, $i = 1..p$, gets larger with increasing i.

4.2. Case Study 2: Absorption function. The absorption function has the form $y = f(x)$ with $f : I\!R^n \to I\!R$ and

$$y = \prod_{i=0}^{n-1} \left(\varphi_i(x_i) \cdot \prod_{j=1}^{2(i+1)} x_i \right) .$$

This name is motivated by the shape of the c-graph, which is displayed in FIG. 4.1, and the behavior of f with respect to different vertex elimination strategies. It represents an example where as a result of the absorption of potentially new edges the reverse

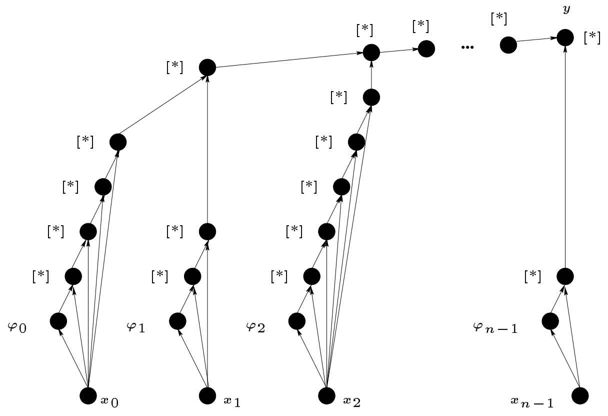

FIG. 4.1. *Absorption Function*

mode **R** is not optimal in terms of the number of `maf`'s required for the computation of the gradient. Looking at the c-graph of f we observe the following: **R** starts with the elimination of the $n-2$ vertices in the outer loop at a Markowitz degree of two each. It goes on with the $2(i+1)$ vertices of the inner loop, again, each of them at a cost of two followed by the n univariate scalar functions φ_i. Hence, $\mathtt{MAF}_{f'}(\mathbf{R}) = 2n^2 + 5n - 4$. Using the forward vertex elimination strategy **F** we would start with the φ_i followed by the elimination of all vertices of the inner loop at a Markowitz degree of one due to the exploitation of absorption. Finally, the outer loop vertices are eliminated successively with their costs incremented by one after each step: $\mathtt{MAF}_{f'}(\mathbf{F}) = \frac{3}{2}n^2 + \frac{3}{2}n - 1$. We would prefer some combination of **F** and **R** in order to be able to accumulate the gradient of f at a minimal cost by behaving as **F** on the inner loop while eliminating the vertices in the outer loop backwards as done by **R**. **LM** would certainly satisfy the first requirement. However, for the outer loop it would again have to be combined with **R** as primary tie-breaker.

EM is the optimal choice for this problem. It makes full use of absorption as the forward mode does while avoiding the incrementation of the vertex degrees by running through the outer loop backward resulting in $\mathtt{MAF}_{f'}(\mathbf{EM}) = n^2 + 4n - 4$.

4.3. Case Study 3: Unsaturated flow problem. This example is a two-dimensional unsaturated flow problem in a porous medium described in [CGRW92]. Its Jacobian F' is an extremely sparse 1989×1989 square matrix yielding a very regular sparsity structure arising from the underlying discretization grid. Notice, that **EM** does not use this knowledge. We have chosen to optimize the computation of the sub-matrix $\hat{D} = J(0..935, 936..1286)$ of the Jacobian. The corresponding c-graph consists of 351 minimal, 936 maximal, and 21177 intermediate vertices. Robey [CGRW92] recognized that the 351 rows of $\hat{D}$ could be computed in only 6 passes. This would result in an operations count of approximately 133 thousand. By fully exploiting the structural sparsity of the problem in **F** the cost could be decreased further to 73 thousand `maf`'s. With 29 thousand `maf`'s, the **EM** heuristic delivered the lowest known overall Markowitz degree, so far. This represents an improvement by a factor of more than 4! **LM** lead to a slightly larger operations count.

4.4. Further Results. By applying them to a selection of `MINPACK` test problems [ACM91] we have compared the heuristics **LM** and **EM** with the optimal unidirectional Newsam-Ramsdell method **NR** [NeRa83], [Gri00]. **NR** represents one of the most powerful static analyses for exploiting sparsity of the Jacobian. It gives a good impression on the efficiency of tools that are built on the forward and reverse modes of AD.

	p	**NR**	**LM**	**EM**
FDC	984	11000	1338	930
WAT	1683	11830	4240	4240
DIE	2499	50380	1659	1659
GDF	1625	18590	1430	1430

One observes that reductions in the operations count of nearly three up to 10 and more are achievable. A large part of the savings is due to the full exploitation of structural sparsity of the extended Jacobian. There are many problems where **LM** and **EM** deliver equivalent results. Although, of course, it is possible to come up with examples where **EM** would fail, among the more than 50 test problems there was none for which **LM** delivered a lower operations count than **EM**.

In [Nau99] we have compared the run times of an optimized Jacobian code and a state-of-the-art AD based code. The objective was to show that the theoretical improvements in the operations count could be "translated" into run time savings. Promising a reduction of the operations count by a factor of 4 the resulting optimized code ran 3.5 times faster than the original version.

4.5. Conclusion. For most real-world problems the generation of optimized derivative code based on either forward or backward vertex elimination sequences would result in remarkable savings in the overall operations count due to the full exploitation of structural sparsity of the extended Jacobian. Unfortunately, it is not clear a priori whether we should prefer the forward or the backward approach. Therefore it would be useful to exploit the strength of heuristics for determining nearly optimal vertex elimination sequences for almost all sorts of c-graphs. **EM** turned out to be very consistent, while being only slightly more expensive than the pure unidirectional methods. It could be worth to analyze selected evaluation routines deeper by using more costly optimization methods like dynamic programming or simulated annealing [Nau99]. However, the additional effort is justified only if the resulting derivative code is generated once and is then used over and over again.

REFERENCES

[ACM91] B. Averik, R. Carter, and J. More, *The Minpack-2 test problem collection*, TM No. 150, ANL, 1991.

[BBCG96] M. Berz, C. Bischof, G. Corliss, and A. Griewank, eds, *Computational differentiation: techniques, applications, and tools*, SIAM, Philadelphia, 1996.

[Bis96] C. Bischof, *Hierarchical approaches to automatic differentiation*, in [BBCG96], pp. 83–94.

[ADIFOR] C. Bischof, A. Carle, P. Hovland, P. Khademi, and A. Mauer, *ADIFOR 2.0 user's guide (revision D)*, Argonne National Laboratory, March 1995. Revised: June, 1998.

[CoGr91] G. Corliss and A. Griewank, eds, *Automatic differentiation: theory, implementation, and application*, SIAM, Philadelphia, 1991.

[CGRW92] G. Corliss, A. Griewank, T. Robey, and S. Wright, *Automatic differentiation applied to unsaturated flow — ADOL-C case study*, Preprint ANL/MCS–TM–162, Mathematical and Computer Science Division, Argonne National Laboratory, 1992.

[Fau00] C. Faure, *Adjoining strategies for multi-layered programs*, Optimisation Methods and Software, 2000. To appear.

[ODYSSEE] C. Faure and Y. Papegay, *Odyssee user's guide. Version 1.7*, Rapport technique 0224, INRIA, September 1998.

[GaJo79] M. Garey and D. Johnson, *Computers and intractability - a guide to the theory of NP-completeness*, W. H. Freeman and Company, San Francisco, 1979.

[TAMC] R. Giering and T. Kaminski, *Recipes for Adjoint Code Construction*, ACM Trans. On Math. Software, Vol. 24, Nr. 4, p437-474, 1998.

[Gri92] A. Griewank, *Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation*, Optimization Methods and Software 1 (1992), 35 - 54.

[Gri00] A. Griewank, *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*, Frontiers in Appl. Math., no. 19, SIAM, Philadelphia, 2000.

[GrRe91] A. Griewank and S. Reese, *On the calculation of Jacobian matrices by the Markowitz rule*, in [CoGr91], pp. 126-135.

[MeNä96] K. Mehlhorn and S. Näher, *LEDA, a platform for combinatorial and geometric computing*, Communications of ACM, Vol. 38, no. 1, pp. 96-102, 1995.

[Nau99] U. Naumann, *Efficient calculation of Jacobian matrices by optimized application of the chain rule to computational graphs* Ph.D. thesis, Dresden, 1999.

[NeRa83] G. Newsam and J. Ramsdell, *Estimation of sparse Jacobian matrices*, SIAM(SJADM), 4 (1983), pp. 404-417.

[Spe80] B. Speelpenning, *Compiling fast partial derivatives of functions given by algorithms*, Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana-Champaign, January 1980.

[Wen64] R. E. Wengert, *A simple automatic derivative evaluation program*, Comm. ACM, 7 (1964), pp. 463-464.