Mathematical Programming manuscript No.
(will be inserted by the editor)

Uwe Naumann

Optimal Accumulation of Jacobian Matrices by Elimination Methods on the Dual Computational Graph

the date of receipt and acceptance should be inserted later

Abstract. The accumulation of the Jacobian matrix F' of a vector function $F : I\!R^n \rightarrow I\!R^m$ can be regarded as a transformation of its linearized computational graph into a subgraph of the directed complete bipartite graph $K_{n,m}$. This transformation can be performed by applying different elimination techniques that may lead to varying costs for computing F'.

This paper introduces face elimination as the basic technique for accumulating Jacobian matrices by using a minimal number of arithmetic operations. Its superiority over both edge and vertex elimination methods is shown. The intention is to establish the conceptual basis for the ongoing development of algorithms for optimizing the computation of Jacobian matrices.

Key words. Jacobian matrices, computational graphs, elimination techniques, automatic differentiation

1. Introduction to Accumulation of Jacobian Matrices

The efficient computation of accurate derivative information for mathematical models is central to many scientific, economic, and engineering problems. Without it the highly desirable step from simulation to optimization often cannot be made. First-order derivatives of nonlinear vector functions given as computer programs that are written in some imperative programming language, such as C or Fortran, play an especially important role in modern scientific computing. *Automatic differentiation* (AD) [4], [7], [8], [13] allows us to compute such derivative information efficiently with machine accuracy.

The Jacobian matrix (or simply Jacobian) of a nonlinear vector function $\mathbf{y} = F(\mathbf{x})$, $F : I\!R^n \supseteq D \rightarrow I\!R^m$, evaluated at a given argument $\mathbf{x}_0$, is defined as follows:

$$(I\!R^{m\times n} \ni)\ F' = F'(\mathbf{x}_0) = \left(\frac{\partial y_i}{\partial x_j}(\mathbf{x}_0)\right)_{i=1,\ldots,m,\ j=1,\ldots,n} \quad .$$

Following the standard notation as in [13], we assume the computer program that implements F to decompose into a sequence of q assignments of the values of scalar elemental functions φ_j to unique intermediate variables v_j. The *code list* of F is given as

$$(I\!R \ni)\ v_j = \varphi_j(v_i)_{i\prec j} \quad , \tag{1}$$

Mathematic and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, e-mail: naumann@mcs.anl.gov

where $j = 1, \ldots, q$ and $p = q - m$. The direct dependence of v_j on v_i is denoted by $i \prec j$. The notation $i \prec^* j$ is used if there exist $k_1, \ldots, k_\nu$ such that $i \prec k_1 \prec k_2 \prec \ldots \prec k_\nu \prec j$. So, $P_j = \{i : i \prec j\}$ is the index set of the arguments of φ_j, and we denote its cardinality by $|P_j|$. Similarly, $S_j = \{i : j \prec i\}$ is the index set of the $|S_j|$ elemental functions that have v_j as an argument. We distinguish between independent $(\{v_{1-n}, \ldots, v_0\})$, intermediate $(\{v_1, \ldots, v_p\})$, and dependent $(\{v_{p+1}, \ldots, v_q\})$ variables, and we set $x_i \equiv v_{i-n}$, $i = 1, \ldots, n$, and $y_j \equiv v_{p+j}$, $j = 1, \ldots, m$. The *computational graph* (or *c-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 $i \prec j$. Moreover, $V = X \cup Z \cup Y$, where $X = \{1-n, \ldots, 0\}$, $Z = \{1, \ldots, p\}$, and $Y = \{p+1, \ldots, q\}$. An edge $k \equiv (i, j) \in E$ has a source $i = \texttt{src}(k)$ and a target $j = \texttt{tgt}(k)$. Throughout this paper, edges are addressed by their uniquely assigned indices and vertices as sources, or targets, of edges. This is our main notational change from [13].

We assume $\mathbf{G}$ to be linearized in the sense that the local partial derivatives

$$c_j \equiv \frac{\partial}{\partial v_{\texttt{src}(j)}} \varphi_{\texttt{tgt}(j)}(v_k)_{k \prec \texttt{tgt}(j)} \tag{2}$$

are attached to the corresponding edges $j \in E$. We rely on the existence of jointly continuous partial derivatives for all elemental functions φ_i, $i = 1, \ldots, q$, on open neighborhoods $\mathcal{D}_i \subset I\!R^{|P_i|}$ of their respective domains. Edges in E are numbered so that, w.l.o.g., $E \subset I\!N$, where $I\!N$ denotes the natural numbers. Parallel edges having the same source and target are merged immediately by adding the values of the local partial derivatives labeling them.

F' can be accumulated by using the *forward (vector) mode* of AD [36], which computes Jacobian-matrix products $\dot{Y} = F'\dot{X}$ by forward propagation of (sets of) tangents as described in [13, Chapter 3]. For this purpose one simply sets $\dot{X} = I_n$, where I_n denotes the identity matrix in $I\!R^{n \times n}$. Similarly, the *reverse (vector) mode* of AD, which computes matrix-Jacobian products, (see [18] for a discussion of its origins) can be applied to get F' by setting $\bar{Y} = I_m \in I\!R^{m \times m}$ in $\bar{X} = \bar{Y}F'$. By the chain rule and with $|E|$ denoting the number of edges in $\mathbf{G}$, this approach would require the evaluation of $n|E|$ and $m|E|$ *fused multiply-add* (`fma`) operations $c_k = c_k + c_j c_i$, where $\texttt{src}(j) \equiv \texttt{tgt}(i)$, $\texttt{src}(i) \equiv \texttt{src}(k)$, and $\texttt{tgt}(j) \equiv \texttt{tgt}(k)$, respectively (see Lemma 6). Some modern floating-point units can perform these operations in the same time as single scalar multiplications [19]. In other words, they give the addition on top of a multiplication “for free.” Other architectures, including Compaq Alpha, SGI and SUN Ultra SPARC machines, do not perform `fma`’s but can perform two independent adds or multiplies per clock cycle [10]. They therefore would complete one multiply-add in two clock cycles but average two multiply-adds per cycle if they can pipeline the operations. This paper assumes the number of `fma`’s as the measure of complexity. The number of `fma`’s is equal to the number of scalar floating-point multiplications because the latter represents an upper bound for the number of additions performed. The task of finding a way to compute F' by using a minimal number of `fma`’s will be referred to as the optimal Jacobian accumulation (OJA) problem. The approach taken in this paper is to consider the accumulation of the Jacobian as a transformation of the c-graph **G** into **G**$'$, a subgraph of the directed complete bipartite graph $K_{n,m}$ such that the labels on the edges in **G**$'$ are exactly the nonzero entries of F'.

From the chain rule it follows that an entry of the Jacobian can be computed by multiplying the edge labels over all paths connecting the corresponding $i \in X$ and $j \in Y$ followed by adding these products [23]. This can be expressed as

$$\frac{\partial v_j}{\partial v_i} = \sum_{[i \to j]} \prod_{k \in [i \to j]} c_k \quad , \tag{3}$$

where $[i \to j]$ denotes a path leading from i to j and $k \in E$ is an edge contained within $[i \to j]$. In other words, F' can be accumulated by enumerating all paths connecting minimal (in X) with maximal (in Y) vertices in **G**. This paper introduces methods for transforming **G** into **G**$'$. The paper is structured as follows. In Section 2 we discuss an example motivating new elimination techniques. In Section 3 we introduce face elimination and derive both edge and vertex eliminations as special cases. In Section 4 we prove the superiority (in terms of `fma`'s) of edge over vertex elimination and of face over edge elimination In Section 5 we briefly discuss algorithms for constructing near-optimal elimination sequences. In Section 6 we draw conclusions.

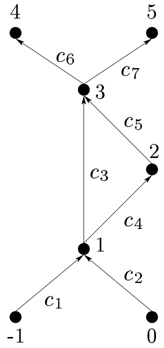

Fig. 1. G

2. Motivating Example

Consider the vector function $F : I\!R^2 \to I\!R^2$ given by the following code list:

$$v_1 = v_{-1}v_0;\ v_2 = \sin(v_1);\ v_3 = v_1 v_2;\ v_4 = \cos(v_3);\ v_5 = \exp(v_3).$$

Its c-graph **G** is shown in Figure 1 with the local partial derivatives attached to the corresponding edges. They take the following values:

$$c_1 = v_0;\ c_2 = v_{-1};\ c_3 = v_2;\ c_4 = \cos(v_1);\ c_5 = v_1;\ c_6 = -\sin(v_3);\ c_7 = v_5.$$

We are looking for an efficient way to transform **G** into **G**$'$, as shown in Figure 2. According to equation (3) the Jacobian of F is given explicitly by

$$F' = \begin{pmatrix} c_1c_3c_6 + c_1c_4c_5c_6 & c_2c_3c_6 + c_2c_4c_5c_6 \\ c_1c_3c_7 + c_1c_4c_5c_7 & c_2c_3c_7 + c_2c_4c_5c_7 \end{pmatrix} \quad . \tag{4}$$

Computing F' this way involves 20 `fma`'s. Both forward and reverse modes of AD perform $2|E| = 14$ `fma`'s, where $n = m = 2$. As F' is dense, seed matrix compression techniques [31], [9] are not applicable. Sparse forward and reverse modes [13, Chapter 6] reduce the cost of accumulating F' to 12 `fma`'s at the expense of performing sparse vector arithmetic. The reader may wish to verify that

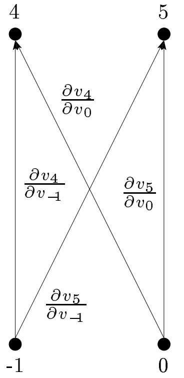
4
5
$\frac{\partial v_4}{\partial v_0}$
$\frac{\partial v_4}{\partial v_{-1}}$
$\frac{\partial v_5}{\partial v_0}$
$\frac{\partial v_5}{\partial v_{-1}}$
-1
0

Fig. 2. $\mathbf{G}'$

$$F' = \begin{pmatrix} c_6 s & c_6 t \\ c_7 s & c_7 t \end{pmatrix} \qquad (5)$$

can be computed at a cost of only 7 `fma`'s by preaccumulation of $r = c_3 + c_5 c_4$, $s = rc_1$, and $t = rc_2$.

This paper introduces the theoretical framework for constructing derivative code that implements equation (5), in general. Undoubtedly, the fact that improvements by a factor of two and more are possible even on this simple example is very promising. So far, the underlying theory has been looked at only very briefly [15], [5]. In [24] we proposed several optimization methods for Jacobian code and presented numerical results. Chapter 8 in [13] is dedicated to elimination techniques and contains some of the ideas presented here. Several specific optimization methods are currently being developed based on the elimination techniques to be introduced [29], [35].

3. Elimination Techniques

Before introducing elimination techniques, we motivate the feasibility of the general approach in practice. The generation of efficient code for accumulating F' is regarded as a compile time activity. Therefore, $\mathbf{G}$ must be available at compile time. However, computer programs may contain loops with variable bounds and branches that make the structure of $\mathbf{G}$ dependent on the actual argument at which F is evaluated. The structure of $\mathbf{G}$ may change dynamically.

Basic blocks [1] represent the primary candidates for an accumulation of *local* Jacobians using elimination techniques (see also [6]). Their c-graphs can be built at compile time, and their sizes are usually small. The question is: What can we gain?

Let F be given as a basic block, and consider the computation of $\dot{Y} = F'\dot{X}$, $\dot{X} \in I\!R^{n \times l}$. For simplicity, we assume that $m = n$ and that $l \gg n$. Techniques for exploiting this inequality are also referred to as *interface contractions* [5]. With the forward vector mode of AD, the number of `fma`'s performed is equal to $l|E|$. Suppose that the forward vector mode of AD is applied to preaccumulate F' by forward propagation of I_n, as described in Section 2, followed by the evaluation of the matrix product $F'\dot{X}$. The first preaccumulation would cost at most $n|E|$ `fma`'s, and the matrix product adds $n^2 l$. The (worst case) question is: Under which circumstances $l|E|$ will become larger than $n^2 l + n|E|$? The answer depends on the given problem (see also [13, Chapter 8]). For example, if $l \geq 2n$ and $|E| \geq 4n^2$, this approach would yield savings of a factor of at least two.

Formally, for the preaccumulation to be more efficient, we need

$$\begin{aligned} n|E| + n^2 l &< l|E| \\ n^2 l &< |E|(l-n) \\ |E| &> \frac{n^2 l}{l-n} \\ |E| &> \frac{n^2}{1-\frac{n}{l}} \quad . \end{aligned}$$

That is, the more complicated the function (large $|E|$) or the greater the number of Jacobian-vector products required (large l), the greater the savings due to preaccumulation. Moreover, the cost of accumulating the Jacobian itself can be reduced significantly by applying the elimination techniques, to be introduced next.

3.1. Dual Computational Graphs

All elimination techniques discussed in this paper are based on the elimination of *transitive dependences* between variables in F. A variable v_j depends transitively on v_i via v_k if $i \prec k \prec j$. In general, there is no structural representation for eliminating such dependences in $\mathbf{G}$; that is, it cannot be expressed by modifying either V or E. A richer data structure is required, namely, a directed variant of the line graph of $\mathbf{G}$.

For $i, j \in E$ we write $i \prec j \Leftrightarrow \texttt{tgt}(i) = \texttt{src}(j)$. The transitive closure of this relation between edges is also denoted by $\prec^*$.

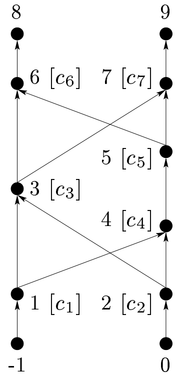

Fig. 3. $\tilde{\mathbf{G}}$

Definition 1 *The* dual c-graph $\tilde{\mathbf{G}} = (\tilde{V}, \tilde{E})$ *of* $\mathbf{G}$ *consists of vertices* $\tilde{V} = \tilde{X} \cup \tilde{Z} \cup \tilde{Y}$, *such that* $\tilde{X} \cap \tilde{Z} = \tilde{X} \cap \tilde{Y} = \tilde{Y} \cap \tilde{Z} = \varnothing$, *and edges* $\tilde{E} = \tilde{E}_X \cup \tilde{E}_Z \cup \tilde{E}_Y$, *with* $\tilde{E}_X \cap \tilde{E}_Z = \tilde{E}_X \cap \tilde{E}_Y = \tilde{E}_Y \cap \tilde{E}_Z = \varnothing$. *It is defined by the following construction:*

1. $\tilde{Z} = E$;
2. $(i,j) \in \tilde{E}_Z$ *for all* $i, j \in E$ *and* $i \prec j$;
3. $\forall\, i \in X \ : \ i \in \tilde{X}$;
4. $\forall\, i \in Y \ : \ i - p + |E| \in \tilde{Y}$;
5. $(i,j) \in \tilde{E}_X$ *for* $i \in \tilde{X}$, $j \in \tilde{Z}$, *and* $\texttt{src}(j) = i$ *in* $\mathbf{G}$;
6. $(i,j) \in \tilde{E}_Y$ *for* $i \in \tilde{Z}$, $j \in \tilde{Y}$, *and* $\texttt{tgt}(i) = j - |E| + p$ *in* $\mathbf{G}$.

In other words, intermediate vertices $\tilde{Z}$ in $\tilde{\mathbf{G}}$ are introduced for all edges in $\mathbf{G}$. Two of them are adjacent in $\tilde{\mathbf{G}}$ whenever the corresponding edges are incident in $\mathbf{G}$. We write $i \hat{=} (j,k)$ if $i \in \tilde{Z}$ corresponds to $(j,k) \in E$. Additional minimal vertices $\tilde{X}$ and maximal vertices $\tilde{Y}$ are required to represent the fact that two

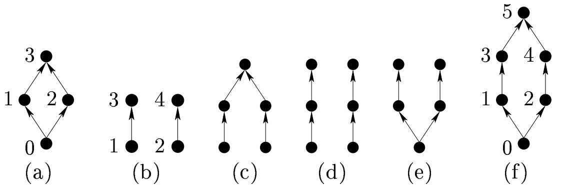

Fig. 4. Motivation for definition of dual c-graph

edges in $\mathbf{G}$ share either the same minimal source or the same maximal target. Consequently, the vertices in $\tilde{\mathbf{G}}$ are given by

$$\tilde{V} = \tilde{X} \cup \tilde{Z} \cup \tilde{Y} = \{1-n, \ldots, 0\} \cup \{1, \ldots, |E|\} \cup \{|E|+1, \ldots, |E|+m\}.$$

The edge labels in $\mathbf{G}$ are attached to the corresponding intermediate vertices in $\tilde{\mathbf{G}}$. The precedence relation "$\prec$" is extended to all vertices in $\tilde{\mathbf{G}}$ as $i \prec j \Leftrightarrow (i,j) \in \tilde{E}$.

Figure 3 shows the dual of the c-graph in Figure 1. The set of intermediate vertices $\tilde{Z}$ corresponds to the edges E in the c-graph $\mathbf{G}$. For example, vertex $1 \in \tilde{Z}$ corresponds to the edge $(-1,1) \in E$ that is labelled c_1 in $\mathbf{G}$ (step 1 in Definition 1). There is an edge in $\tilde{E}_Z$ connecting a vertex $i \in \tilde{Z}$ with a vertex $j \in \tilde{Z}$ if the edges corresponding to i and j in G share a common vertex, more precisely, if $i \hat{=} (k_1, k_2)$ and $j \hat{=} (k_2, k_3)$ for some $k_1, k_2, k_3 \in V$. For example, the vertices 4 and 5 are connected by the edge $(4,5) \in \tilde{E}_Z$ because the edge labelled c_4 in the c-graph $\mathbf{G}$, that is $(1,2) \in E$, precedes the edge labelled c_5, that is $(2,3) \in E$, at vertex $2 \in V$ (step 2 in Definition 1).

Two minimal vertices -1 and 0 are introduced to represent assumed edges leading into the minimal vertices in $\mathbf{G}$ (step 3 in Definition 1). They are connected to the vertices 1 and 2, respectively (step 5 in Definition 1). A similar action is performed for the maximal vertices in $\mathbf{G}$ (steps 4 and 6 in Definition 1) leading to the vertices 8 and 9 and the edges $(6,8)$ and (7,9) in $\tilde{\mathbf{G}}$. This extension of the dual c-graph is required to represent the fact that two edges emanating from minimal vertices in $\mathbf{G}$ can share the same source. Consider, for example, the c-graph in Figure 4(a). The subgraph of the dual c-graph obtained by performing steps 1 and 2 of Definition 1 is shown in Figure 4(b). The structural property of $\mathbf{G}$ that the edges $(0,1)$ and $(0,2)$ emanate from the same vertex is not represented. The same applies to the edge $(1,3)$ and $(2,3)$ that share the same target. In other words, the graph depicted in Figure 4(b) could as well be the dual of any of the c-graphs shown in Figure 4(c)–(e). Steps 3–6 in Definition 1 are required to make the mapping between $\mathbf{G}$ and $\tilde{\mathbf{G}}$ bijective.

Dual c-graphs are uniquely characterized by the following two properties.

1. All intermediate vertices belong to exactly two directed complete bipartite subgraphs of $\tilde{\mathbf{G}}$. They are minimal in one and maximal in the other.
   Intermediate vertices in $\mathbf{G}$ are mapped onto complete bipartite subgraphs (or bicliques) $K_{\nu,\mu}$ of $\tilde{\mathbf{G}}$. The ν incoming edges of some $i \in Z$ represent the minimal vertices in $K_{\nu,\mu}$. Similarly, the μ outgoing edges correspond

to the maximal vertices in $K_{\nu,\mu}$. Obviously, each minimal vertex in $K_{\nu,\mu}$ is connected with all maximal vertices because the corresponding edges are incident in $\mathbf{G}$. For example, in Figure 3, the biclique spanned by the locally minimal vertices 3 and 5 and the locally maximal vertices 6 and 7 corresponds to vertex 3 in $\mathbf{G}$.

2. Minimal vertices in $\tilde{\mathbf{G}}$ belong to one directed complete bipartite subgraph, and $i \in \tilde{X}$, and $i \prec j$ implies $|P_j| = 1$. Analogous, maximal vertices belong to a single directed complete bipartite subgraph of $\tilde{\mathbf{G}}$, and $i \in \tilde{Y}$, and $j \prec i$ implies $|S_j| = 1$. Here, the notations P_j and S_j are used as defined in Section 1 with the precedence relation $\prec$ applied to edges in $\mathbf{G}$.
   As a result of the construction described in Definition 1, all minimal vertices in $\tilde{\mathbf{G}}$ are also minimal in the bicliques whose maximal vertices represent the edges emanating from the corresponding minimal vertices in $\mathbf{G}$. Examples are the subgraphs induced by vertex -1 and vertex 1 in Figure 3, or the biclique spanned by the vertices 0, 1, and 2 in Figure 4(f). By Definition 1, the maximal vertices in these bicliques can have only one predecessor in $\tilde{\mathbf{G}}$. A similar argument applies to the maximal vertices in $\tilde{\mathbf{G}}$.

Further consequences of the definition of dual c-graphs are the following.

Lemma 1. *Let $i, j \in \tilde{V}$ such that $(i,j) \in \tilde{E}$. Then $\nexists k \in \tilde{V} : i \prec^* k \prec^* j$.*

Proof. Let $k \in \tilde{V}$ be such that $i \prec k \prec j$. Furthermore, let $(a,b), (c,d), (e,f) \in E$ be such that $(a,b) \hat{=} i$, $(c,d) \hat{=} k$, and $(e,f) \hat{=} j$. Then

$$\begin{array}{lcl} i \prec j & \Rightarrow & b = e \\ i \prec k & \Rightarrow & b = c \\ k \prec j & \Rightarrow & d = e \quad . \end{array}$$

The above implies that $c = d$, which represents a contradiction to the definition of $\mathbf{G}$ as a directed acyclic graph. The lemma follows by induction over the length of the path that connects i and j while containing k. □

Lemma 2. *Whenever two vertices in $\tilde{V}$ share a common predecessor (successor), their predecessor (successor) sets are identical.*

Proof. Let $i, j, k \in \tilde{V}$ be such that $k \prec i$ and $k \prec j$. The edges corresponding to i and j in $\mathbf{G}$ have the same source. Therefore any edge that is incident to either i or j must be incident to the other. □

Definition 2 *A semicycle is defined as two vertex disjoint paths sharing the same source and the same target.*

For example, the subgraph of $\tilde{\mathbf{G}}$ that is induced by the vertices 2, 3, 4, 5, and 6 in Figure 3 is a semicycle.

Lemma 3. *The length of any semicycle in $\tilde{\mathbf{G}}$ is greater than or equal to five.*

Proof. From Lemma 1, it follows that the length of any semicycle must be at least equal to four. However, this situation corresponds to parallel edges in $\mathbf{G}$, which contradicts its definition. A semicycle of length five in $\tilde{\mathbf{G}}$ represents a *triangle* in $\mathbf{G}$, that is, a subgraph consisting of three vertices $i, j, k \in V$ such that $(i, j), (j, k), (i, k) \in E$. □

For example, the triangle spanned by the vertices 1, 2, and 3 in Figure 1 corresponds to the semicycle induced by the vertices 2, 3, 4, 5, and 7 in Figure 3. Note that this mapping is not bijective. Other related semicycles of length five in $\tilde{\mathbf{G}}$ are spanned by the vertex sets $\{1, 3, 4, 5, 7\}$, $\{1, 3, 4, 5, 6\}$, and $\{2, 3, 4, 5, 6\}$.

3.2. Face Elimination

The elimination of transitive dependences can be interpreted as the elimination of edges in $\tilde{\mathbf{G}}$. This modification of the dual c-graph is also referred to as *face elimination* in order to distinguish between edge elimination in $\tilde{\mathbf{G}}$ and $\mathbf{G}$ (see Section 3.3 for edge elimination in $\mathbf{G}$). Two intermediate vertices in $\tilde{\mathbf{G}}$ are adjacent if the corresponding edges are incident in $\mathbf{G}$. This property implies that the transitive dependence to be eliminated does actually exist. A vertex $j \in \tilde{V}$ is called *isolated* if either $P_j = \emptyset$ or $S_j = \emptyset$.

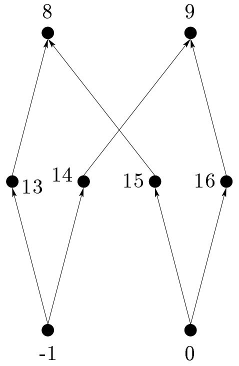

Fig. 5. $\tilde{\mathbf{G}}'$

Rule 1 *Face elimination is defined for all intermediate faces $(i, j) \in \tilde{E}_Z$ as follows:*

1. *If there exists a vertex $k \in \tilde{V}$ such that $P_k = P_i$ and $S_k = S_j$, then set $c_k = c_k + c_j c_i$* (absorption)*; else $\tilde{V} = \tilde{V} \cup \{k'\}$ with a new vertex k' such that $P_{k'} = P_i$ and $S_{k'} = S_j$* (fill-in) *and labeled with $c_{k'} = c_j c_i$.*
2. *Remove (i, j) from $\tilde{E}$.*
3. *Remove $i \in \tilde{V}$ if it is isolated. Otherwise, if there exists a vertex $i' \in \tilde{V}$ such that $P_{i'} = P_i$ and $S_{i'} = S_i$, then*
   - *set $c_i = c_i + c_{i'}$* (merge)*;*
   - *remove i'.*
4. *Repeat Step 3 for $j \in \tilde{V}$.*

Rule 1 is illustrated in Figure 6. The dual c-graph $\tilde{\mathbf{G}}$ of Figure 3 can be transformed into a tripartite form as shown in Figure 5 by applying face elimination successively. The corresponding face elimination sequences are also referred to as complete. Obviously, for the dual c-graph in Figure 3 there are a large number of different face elimination sequences. We consider an example. Let us start with the elimination of $(1, 3) \in \tilde{E}$. An absorbing vertex does not exist leading to the generation of fill-in. The new vertex 10 is connected by new edges to all successors of 3. A new edge connects the single predecessor of 1 with 10. Vertex 10 is labeled with $c_{10} = c_3 c_1$. Neither 1 nor 3 is isolated or can be merged

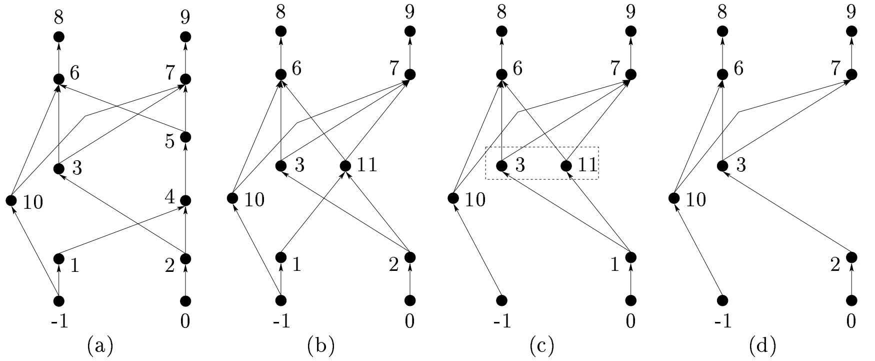

Fig. 6. Face elimination

with other vertices, and we get the graph in Figure 6(a). Edge $(4, 5)$ is eliminated next, thus leaving both 4 and 5 isolated and resulting in their removal. We observe that $P_3 \subsetneq P_4$ and $S_3 = S_5$, leading to the generation of fill-in in the form of the new vertex 11. Its label is $c_{11} = c_5 c_4$. In the resulting graph, shown in Figure 6(b), we eliminate $(1, 11)$, leading to the removal of vertex 1 because of isolation. Absorption takes place as $P_{10} = P_1$ and $S_{10} = S_{11}$. In other words, the fill-vertex that is generated as a result of eliminating $(1, 11)$ has the same predecessors and successors as vertex 10. Therefore it is absorbed and $c_{10} = c_{10} + c_{11} c_1$. The result is shown in Figure 6(c). Moreover, $P_3 = P_{11}$ and $S_3 = S_{11}$. Hence, 3 and 11 can be merged, leading to the graph in Figure 6(d), where $c_3 = c_3 + c_{11} = c_3 + c_5 c_4$. By eliminating $(2, 3)$ next, only five more face eliminations are required to transform $\tilde{\mathbf{G}}$ into a tripartite form, as displayed in Figure 5. The corresponding code for accumulating the Jacobian becomes

$$c_{10} = c_3 c_2;\; c_{11} = c_5 c_4;\; c_{10} = c_{10} + c_{11} c_1;\; c_3 = c_3 + c_{11};\; c_{12} = c_3 c_2$$

$$c_{13} = c_6 c_{10};\; c_{14} = c_7 c_{10};\; c_{15} = c_6 c_{12};\; c_{16} = c_7 c_{12} \quad .$$

Fully inlining all intermediate local partial derivatives verifies that $c_{13}, \ldots, c_{16}$ are, in fact, the Jacobian entries as in (4). It remains to be shown that this graph is the dual of $\mathbf{G}'$ from Figure 2. Therefore, face elimination must always lead to a result after performing a finite number of steps. The resulting graph must be a dual directed bipartite graph, and the labels of the intermediate vertices must be modified corresponding to the chain rule.

Lemma 4 (termination). *Every complete sequence of face eliminations that can be applied to a dual c-graph is finite.*

Proof. We show that the sum L over the lengths of all paths connecting minimal with maximal vertices in $\tilde{\mathbf{G}}$ is strictly monotonically decreasing under face elimination.

Consider the elimination of a face $(i, j) \in \tilde{E}_Z$. If absorption takes place, then L is decreased by the sum over all paths containing (i, j) of their respective lengths. In the case of fill-in being generated, the length of each path through (i, j) is decreased by one. By merging two vertices k and k' the value of L is decreased by the sum over all paths through k of their respective lengths. □

Lemma 5 (structural correctness). *The result of eliminating all intermediate faces in a dual c-graph* $\tilde{\mathbf{G}}$ *is the dual c-graph* $\tilde{\mathbf{G}}'$ *of a bipartite c-graph* $\mathbf{G}'$.

Proof. We need to show that face elimination transforms any dual c-graph $\tilde{\mathbf{G}}$ into a tripartite graph such that every intermediate vertex has exactly one predecessor and one successor. This implies that the graph is a dual directed bipartite graph and therefore structurally equivalent to a possibly sparse rectangular matrix.

Edges connecting a minimal and a maximal vertex directly do not exist in a dual c-graph $\tilde{\mathbf{G}}$ and they cannot be generated by face elimination. Since face elimination is defined only for edges $(i, j) \in \tilde{E}_Z$ (that is, $i \in \tilde{Z}$ and $j \in \tilde{Z}$) it must result in a tripartite graph.

Consider $\tilde{\mathbf{G}}$ as derived from a c-graph $\mathbf{G}$ following the construction in Definition 1. All successors of minimal vertices in $\tilde{\mathbf{G}}$ have in-degree one. This remains true as long as edges emanating from successors of minimal vertices are not eliminated. Consider the elimination of an edge $(j, k) \in \tilde{E}_Z$ such that $(i, j) \in \tilde{E}_X$. If the absorbing vertex for (j, k) exists, then the removal of (j, k) (possibly leading to the removal of either j or k or both) is the only structural modification that $\tilde{\mathbf{G}}$ is subjected to. Obviously, this would not violate the property that successors of minimal vertices have a unique predecessor. Now, suppose that fill-in is generated as a result of eliminating (j, k). In this case i gets a new successor that has i as its only predecessor. As above, (j, k) is removed from $\tilde{\mathbf{G}}$. Again, successors of minimal vertices have in-degree one in the resulting graph. Because of the symmetry of face elimination, a similar argument applies to all predecessors of maximal vertices. □

Lemma 6 (numerical correctness). *Let* $\mathbf{G}$ *be the c-graph of a vector function* F, *and let* $\tilde{\mathbf{G}}$ *be the corresponding dual c-graph. If face elimination transforms* $\tilde{\mathbf{G}}$ *into* $\tilde{\mathbf{G}}'$ *as in Lemma 5, then the labels on the intermediate vertices in* $\tilde{\mathbf{G}}'$ *are exactly the nonzero entries of the Jacobian* F'.

Proof. For $G = (V, E)$, where $V = X \cup Z \cup Y$, $|Z| = p$, and for $j \in Y$ and $i \in X$, equation (3) can be rewritten as

$$\frac{\partial v_{j-|E|+p}}{\partial v_i} = \sum_{[i \to j]} \prod_{k \in [i \to j]} c_k \quad , \tag{6}$$

where $[i \to j]$ denotes a vertex path connecting the corresponding $i \in \tilde{X}$ with $j \in \tilde{Y}$ in $\tilde{\mathbf{G}}$. The c_k are the labels of the vertices in $\tilde{\mathbf{G}}$ and we define $c_i = c_j = 1$ for $i \in \tilde{X}$ and $j \in \tilde{Y}$. Face elimination is equivalent to performing one of the multiplications in equation (6) whereas merging two vertices in $\tilde{\mathbf{G}}$ means to add their values. With the structural changes in $\tilde{\mathbf{G}}$ resulting from face elimination it is obvious that the values computed in equation (6) are invariant. □

The following two results ensure that merging vertices in $\tilde{\mathbf{G}}$ is not a recursive procedure, which would increase the complexity of face elimination considerably.

Lemma 7. *By the elimination of* $(i,j) \in \tilde{E}_Z$ *no vertex other than either* i *or* j *becomes mergeable.*

Proof. The elimination of (i,j) results in $P_j = P_j \setminus \{i\}$ and $S_i = S_i \setminus \{j\}$. If there is a vertex j' in $\tilde{\mathbf{G}}$ such that $P_{j'} = P_j \setminus \{i\}$ and $S_{j'} = S_j$ before the elimination of (i,j), then j and j' can be merged after. A similar argument applies to i.

If the absorbing vertex k for (i,j) exists in $\tilde{\mathbf{G}}$, then all vertices other than i and j that are mergeable after the elimination of (i,j) must have been mergeable before.

Let the absorbing vertex for (i,j) not be in $\tilde{\mathbf{G}}$. Fill-in is generated as $k \in \tilde{V}$ such that $P_k = P_i$ and $S_k = S_j$. If k could be merged with some k', then $P_k = P_{k'}$ and $S_k = S_{k'}$ and k' would be the absorbing vertex for (i,j).

Let some successor j' of j become mergeable with some j'' as a result of inserting k and $(k,j') \in \tilde{E}_Z$. Then (k,j'') must be in $\tilde{E}_Z$, hence implying that $j'' \in S_j$. Furthermore, j' and j'' must have been mergeable before the elimination of (i,j). A similar argument applies to predecessors of k. □

Lemma 8. *Merging two vertices in* $\tilde{\mathbf{G}}$ *cannot result in other vertices becoming mergeable.*

Proof. Let $i,j \in \tilde{V}$ be mergeable (that is, $P_i = P_j$ and $S_i = S_j$). Let $i',j' \in \tilde{V}$ become mergeable as a result of merging i and j. W.l.o.g. let $S_{i'} \neq S_{j'}$ before merging i and j. More precisely, $A = S_{i'} \cap S_{j'}$ such that $S_{i'} = A \cup \{i\}$ and $S_{j'} = A \cup \{j\}$. In this case, however, P_i contains i' but not j' and P_j contains j', but not i', and therefore $P_i \neq P_j$, a contradiction to i and j being mergeable. □

3.3. Edge and Vertex Elimination

Suppose $\tilde{\mathbf{G}}$ is the dual of a c-graph $\mathbf{G}$. Then the simultaneous elimination of all edges emanating from some $j \in \tilde{E}$ in $\tilde{\mathbf{G}}$ results in a graph $\tilde{\mathbf{G}}^*$, which is again a dual c-graph. This procedure is referred to as front elimination of the edge corresponding to j in $\mathbf{G}$. Similarly, the simultaneous elimination of all edges leading into some $j \in \tilde{E}$ in $\tilde{\mathbf{G}}$ is referred to as back elimination of j in $\mathbf{G}$. Edge elimination can be defined directly on the level of the underlying c-graph $\mathbf{G}$ as follows.

Rule 2 (Edge Elimination) *1. The back elimination of* $k = (i,j) \in E$ *is performed by introducing new edges* $k' = (i',j) \in E$ *for all* i' *with* $i' \prec i$ *and* $i' \not\prec j$. *The new edge labels are set to* $c_{k'} = c_k c_{k''}$, *where* $k'' = (i',i) \in E$. *For all* $k' = (i',j) \in E$ *the existing edge labels are updated according to* $c_{k'} = c_{k'} + c_k c_{k''}$.

2. $k = (i,j) \in E$ *is front eliminated by introducing new edges* $k' = (i,j') \in E$ *for all* j' *with* $j \prec j'$ *and* $i \not\prec j'$. *The new edge labels are set to* $c_{k'} = c_{k''} c_k$

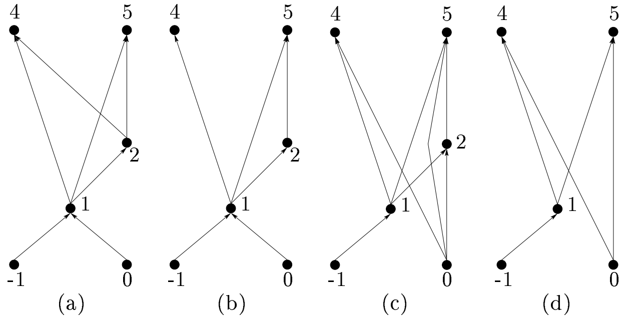

Fig. 7. Vertex and Edge Elimination

where $k'' = (j, j') \in E$. If $k' = (i, j') \in E$, then the existing edge labels are updated according to $c_{k'} = c_{k'} + c_{k''} c_k$.

In both cases (i, j) is deleted. If this deletion leads to either i or j becoming isolated, then the respective vertex is also removed from $\mathbf{G}$ together with all incident edges.

The number of `fma`'s involved in the front elimination of an edge (i, j) is equal to the number of successors of j, that is, $|\{j' : j \prec j'\}|$. Analogously, the back elimination of the same edge would cost $|\{i' : i' \prec i\}|$ `fma`'s. Edges emanating from (leading into) minimal (maximal) vertices in $\mathbf{G}$ cannot be back (front) eliminated. This fact follows immediately from the fact that only intermediate edges can be eliminated in $\tilde{\mathbf{G}}$.

The elimination of a vertex in $\mathbf{G}$ can be formulated as a special case of edge elimination in two ways.

Rule 3 (Vertex Elimination 1) *A vertex $i \in V$ is eliminated from $\mathbf{G}$ by front elimination of all its in-edges.*

Rule 4 (Vertex Elimination 2) *A vertex $i \in V$ is eliminated from $\mathbf{G}$ by back elimination of all its out-edges.*

It is easy to verify that Rule 3 and Rule 4 are equivalent. Being a special case of edge elimination, vertex elimination can be defined on the c-graph. The number of `fma`'s involved in the elimination of a vertex i is equal to $|\{i' : i' \prec i\}||\{i'' : i \prec i''\}|$ and is often referred to as the *Markowitz degree* of i.

Figure 7 illustrates the structural modification of $\mathbf{G}$ caused by edge and vertex elimination. From left to right, we eliminate vertex 3 from the c-graph shown in Figure 1, followed by the back elimination of $(2, 4)$, the front elimination of $(0, 1)$, and the back elimination of $(2, 5)$, which is equivalent to the elimination of vertex 2. The labels on the edges are updated according to the rules above.

4. Optimal Jacobian Accumulation Problem

Three combinatorial optimization problems can be defined by building on the elimination techniques introduced in Section 3. They are referred to as the ver-

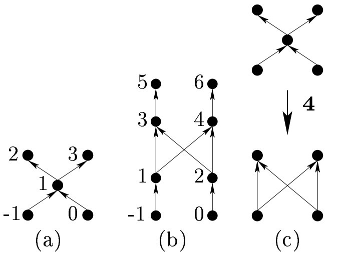

Fig. 8. Example: Metagraphs

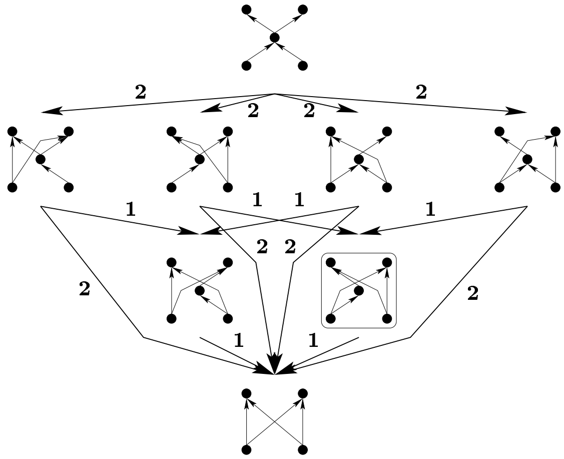

Fig. 9. Edge metagraph $\mathbf{M}_e$

tex elimination (VE), the edge elimination (EE), and the face elimination (FE) problems. The objective is to minimize the number of `fma`'s required to accumulate the Jacobian using the corresponding elimination technique. FE is the most general of the three. It remains unclear, however, whether FE $\equiv$ OJA, where OJA refers to the optimal Jacobian accumulation problem as introduced in Section 1.

All three elimination problems can be interpreted as shortest path problems in metagraphs. The vertices in $\mathbf{M}_f$ ($\mathbf{M}_e$, $\mathbf{M}_v$) are defined as all graphs that can be constructed by using face (edge, vertex) elimination starting with the original dual c-graph $\tilde{\mathbf{G}}$ (or its underlying c-graph $\mathbf{G}$). $\tilde{\mathbf{G}}$ is represented by the unique source of the metagraph. The unique sink of the metagraph represents $\tilde{\mathbf{G}}'$, which is equivalent to the Jacobian. Two vertices s and t in a metagraph are adjacent in a directed sense, that is, there is an edge (s,t) in the metagraph, if the graph represented by t can be obtained by performing a corresponding single elimination in the graph represented by s. All edges in the metagraph are labeled with distances reflecting the computational cost of performing the corresponding face (edge, vertex) elimination. In particular, all edges represent a distance of one in $\mathbf{M}_f$. From the definition of the elimination techniques it

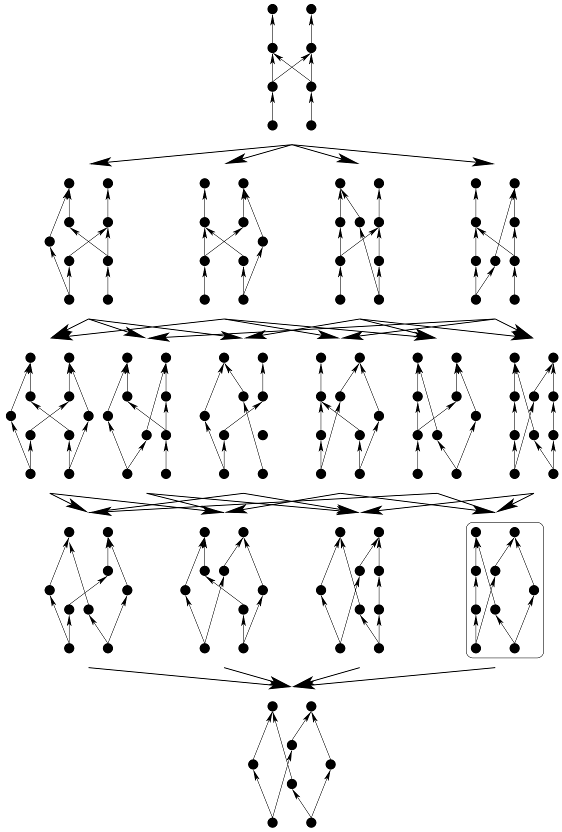

Fig. 10. Face metagraph $\mathbf{M}_f$

follows immediately that $V(\mathbf{M}_v) \subseteq V(\mathbf{M}_e) \subseteq V(\mathbf{M}_f)$, where $V(G)$ denotes the vertex set of some graph G. Here, $V(\mathbf{M}_e) \subseteq V(\mathbf{M}_f)$ means that for each vertex i in $\mathbf{M}_e$ there is a corresponding vertex j in $\mathbf{M}_f$ such that the graph associated with j is the dual of the graph associated with i. From the algorithmic point of view, the metagraphs are of only limited practicality, since their size grows exponentially with the size of the original c-graph.

Consider, for example, the simple c-graph in Figure 8(a). It consists of two minimal, a single intermediate, and two maximal vertices. Its dual is depicted in Figure 8(b). The vertex metagraph $\mathbf{M}_v$ turns out to be trivial, containing only the source (representing the original c-graph) and the sink (corresponding to the bipartite c-graph that represents the Jacobian). Both are connected by a single directed edge labeled with a cost of four (the cost of eliminating vertex 1). $\mathbf{M}_v$ is shown in Figure 8(c).

The edge metagraph contains eight vertices corresponding to intermediate c-graphs that can be constructed from the original c-graph by successive edge eliminations. It is shown in Figure 9. Again, its source is the original c-graph, and the objective is to find the shortest path to the sink (the bipartite c-graph).

The edges are labeled with the respective costs for front or back eliminating an edge. It is straightforward to verify that all paths in $\mathbf{M}_e$ have length four. In other words, any edge elimination sequence is optimal.

A similar effect can be observed for the face metagraph shown in Figure 10. The source is the dual c-graph from Figure 8(b) and the sink is equivalent to the Jacobian. Fourteen different intermediate graphs can be constructed by using face elimination. Obviously, all edge labels in $\mathbf{M}_f$ are equal to one. Again, the length of any path in $\mathbf{M}_f$ is equal to four.

For the extremely simple c-graph in Figure 8(a), vertex, edge, and face elimination techniques are equivalent with regard to the OJA problem. The fact that $\mathbf{M}_e \subseteq \mathbf{M}_f$ can be verified easily. For example, an intermediate c-graph in Figure 9 and its dual in Figure 10 are highlighted (framed).

The problem of minimizing the fill-in regarded as the number of fill edges under a vertex elimination strategy was shown to be NP-complete [11] by Herley [16] in an unpublished adaption of a note by Gilbert [12] on a result by Rose and Tarjan [32] about vertex elimination techniques for solving sparse linear systems. So far, it remains unclear whether the same is true for edge and face elimination and whether this result can be used as a basis for showing the NP-completeness of the OJA problem. In particular we have the following result.

Lemma 9. *A vertex elimination sequence that minimizes the fill-in does not necessarily minimize the number of* `fma`*'s performed.*

A vertex elimination sequence that minimizes the number of `fma`*'s performed does not necessarily minimize the fill-in.*

Proof. Consider the complete graph $\mathbf{G} = K_5 = (V, E)$, where $V = X \cup Z \cup Y$ such that $X = \{0\}$, $Z = \{1, 2, 3\}$, and $Y = \{4\}$. The edges $(i, j) \in E$ are such that $i < j$. None of the six different vertex elimination sequence generates fill-in. However, it is easy to verify that the cost of the vertex elimination sequence $[1, 2, 3]$ is six whereas, for example, $[2, 1, 3]$ would involve seven `fma`'s.

Consider the graph on the left in Figure 12. There are two different vertex elimination sequences which both involve 12 `fma`'s. Notice, that $[1, 2]$ generates 10 fill edges, while $[2, 1]$ results in 11. □

So far, no motivation has been given for introducing face and edge elimination in addition to the conceptually much easier vertex elimination technique. The reasons are formulated as Proposition 1 and Proposition 2. The basic ideas were presented in [27]. To prove them, we require some further results.

Branch-and-bound algorithms [3] have proved useful for solving combinatorial optimization problems. We use the idea in the proof of Lemma 12. Suitable lower bounds are crucial ingredients of this argument. In [28] we showed that the sum over all intermediate vertices of their minimal Markowitz degrees is a lower bound for the solution of the OJA problem. The minimal Markowitz degree of a vertex $j \in V$ is defined as the product $|\underline{P_j}||\underline{S_j}|$, where $\underline{P_j}$ and $\underline{S_j}$ are minimal X-j and j-Y separating vertex sets in $\mathbf{G}$, respectively. Two other lower bounds are established below for smaller classes of c-graphs.

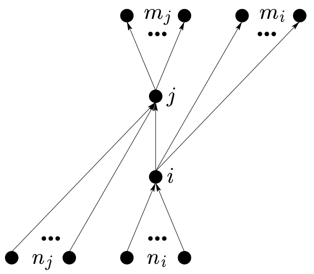

Fig. 11. Proof of Lemma 11

Definition 3 *A directed acyclic graph is called absorption-free if any two vertices are connected by at most a single directed path.*

Absorption-free graphs do not contain semicycles as defined in Definition 2. For example, trees are absorption-free. Notice that the dual of an absorption-free c-graph is also absorption-free. This result follows from Definition 1.

Lemma 10. *The cost of vertex elimination cannot be undercut by either edge or face elimination for c-graphs with at most one intermediate vertex.*

Proof. Let $Z = \{j\}$ in $\mathbf{G}$, and, w.l.o.g, let $\mathbf{G}$ be connected. Then any pair of minimal and maximal vertices is connected by a path containing an intermediate vertex. Consequently, $|P_j||S_j|$ is a lower bound for the cost of accumulating the corresponding Jacobian. This is exactly the cost of eliminating j. □

Lemma 11. *Vertex elimination is optimal for absorption-free c-graphs with two intermediate vertices i and j. If $(i,j) \in E$, then the optimal vertex elimination sequence is $[i,j]$ if $|P_i| \leq |S_j|$ and $[j,i]$ if $|P_i| \geq |S_j|$. If $(i,j) \notin E$, then any vertex elimination sequence is optimal, and the minimal cost is $|P_i||S_i| + |P_j||S_j|$.*

Proof. If $(i,j) \notin E$, then Lemma 10 can be applied separately to i and j. Let $(i,j) \in E$. To determine the optimal vertex elimination sequence and its cost, let $|P_i| = n_i$, $|S_i| = m_i + 1$, $|P_j| = n_j + 1$, and $|S_j| = m_j$. There are two different vertex elimination sequences. Their respective costs yield the inequality

$$n_i(m_i + 1) + (n_j + n_i)m_j \leq m_j(n_j + 1) + (m_j + m_i)n_i .$$

Its solution is given as $n_i \leq m_j$, which proves the second part of the lemma. The situation is illustrated in Figure 11.

Suppose that there is an edge elimination sequence whose cost undercuts the cost of an optimal vertex elimination sequence. The Jacobian can be partitioned into the following three independent parts:

$$A = \left(\frac{\partial v_k}{\partial v_l}\right)_{l \in P_i}^{k \in S_i \setminus \{j\}} , \quad B = \left(\frac{\partial v_k}{\partial v_l}\right)_{l \in P_j \setminus \{i\}}^{k \in S_j} , \quad C = \left(\frac{\partial v_k}{\partial v_l}\right)_{l \in P_i}^{k \in S_j} .$$

Here, *independent* means that the costs of accumulating A, B, and C are mutually independent. In other words, $[A, B, C]$ represents a decomposition of the

Jacobian F' such that the minimal cost of accumulating F' is exactly the sum of the minimal costs of accumulating A, B, and C. Obviously, this is true for any decomposition where the corresponding subgraphs of $\mathbf{G}$ share at most paths of (edge-)length one.

By Lemma 10 the computation of A and B involves exactly $n_i m_i$ and $n_j m_j$ `fma`'s, respectively. In the c-graph corresponding to C any edge elimination sequence that does not yield a corresponding vertex elimination sequence would have to be a mixture of back elimination of outedges of j and front elimination of inedges of i. The cost of such an elimination sequence is given by

$$\sum_{k=1}^{l}\left(n_k\left(\left(\sum_{r=1}^{k-1} m_r\right)+1\right)+m_k\left(\left(\sum_{r=1}^{k} n_r\right)+1\right)\right)-\mu \quad , \tag{7}$$

where

$$\sum_{k=1}^{l} n_k = |P_i| \quad , \qquad \sum_{k=1}^{l} m_k = |S_j| \quad ,$$

and

$$\mu = \begin{cases} m_l & \text{if } \sum_{r=1}^{l-1} m_r < |S_j| \\ n_l & \text{if } \sum_{r=1}^{l-1} m_r = |S_j| \end{cases} \quad .$$

It is easy to verify that this value is always greater than or equal to the cost of an optimal vertex elimination sequence. □

The expression in equation (7) describes the cost of any edge elimination sequence for accumulating C. For example, if all inedges of i are front eliminated followed by the back elimination of the outedges of j, then $l = 1$, and the cost is equal to the cost of the vertex elimination sequence i, j, that is,

$$n_1 + m_1(n_1+1) - m_1 = |P_i|(|S_j|+1) \quad .$$

Similarly, if all outedges of j are back eliminated followed by the front elimination of the inedges of i, then $l = 2$, $n_1 = 0$, and the cost is equal to the cost of the vertex elimination sequence j, i, that is,

$$m_1 + n_2(m_1+1) - n_2 = |S_j|(|P_i|+1) \quad .$$

The back elimination of some outedges of j, say m_1 of them, followed by the front elimination of all inedges of i and the back elimination of the remaining outedges of j implies $l = 2$, $n_1 = 0$, $n_2 = |P_i|$, and a total cost of

$$m_1 + n_2(m_1+1) + m_2(n_1+n_2+1) - m_2 = m_1 + |P_i|(|S_j|+1) \quad .$$

Proposition 1 *An optimal edge elimination sequence may involve fewer* `fma`*'s than does an optimal vertex elimination sequence when applied to the same c-graph.*

Proof. An example for which the proposition is true is presented on the left side of Figure 12. The two possible vertex elimination sequences both require 12 `fma`'s. Back elimination of edge $(2,6)$ before the elimination of vertices 1 and 2 results in 11 `fma`'s. □

The argument leading to the construction of the example c-graph can be found in Chapter 2 in [24]. Because of its shape (turn it 90 degrees clockwise) the graph on the left side of Figure 12 is referred to as the *lion graph.* Recall the metagraph formulation of the EE and VE problems. The set of all valid vertex eliminations is contained within the set of all valid edge eliminations. Hence, the optimal vertex elimination sequence is contained within the set of all edge elimination sequences, and thus the optimal edge elimination sequence performs at most the same number of arithmetic operations as the optimal vertex elimination sequence. The lion graph represents one example where the optimal edge elimination sequence involves fewer operations than does the optimal vertex elimination sequence.

The maximal difference over all c-graphs between the number of `fma`'s performed by an optimal vertex and an optimal edge elimination sequence for the same c-graph is referred to as *vertex-edge discrepancy.* Proposition 1 gives rise to the fundamental question of how large this value can become. In [24] we showed that it is equal to $\frac{1}{2(\sqrt{2}-1)} \approx 1.207$ for c-graphs containing two intermediate vertices. The problem remains open for c-graphs with more than two intermediate vertices.

Lemma 12. *The cost of an optimal edge elimination sequence for the lion graph is equal to eleven.*

Proof. Using the bounds established above, we develop a branch-and-bound argument showing that starting with the elimination of any edge other than $(2,6)$ eventually leads to an increased number of `fma`'s. At every single step we decide whether to continue branching into (branch) or to disregard the subtree (bound). This decision is based on the current lower bound λ, which is defined as the sum of the costs of the edge eliminations performed so far and the values resulting from the lower bounds for the cost of the remaining elimination.

For example, if we start with the front elimination of either $(-1,1)$ or $(0,1)$, then this involves the evaluation of two `fma`'s. A lower bound on the cost of eliminating the remaining edges is given by the minimal Markowitz degree of 1, which is equal to two, plus the minimal Markowitz degree of 2, which is equal to eight, both after the elimination of $(-1,1)$ or $(0,1)$. The sum of these values gives $\lambda = 2+2+8 = 12$. This value is already larger than the best-known value for the number of `fma`'s, which is equal to 11. Therefore, this subtree can be excluded from further consideration. This argument is represented in the decision tree as follows:

- $(-1,1)$ or $(0,1)$ front $\Rightarrow \lambda = 2+2+8 = 12$ ([28]) $\Rightarrow$ bound.

On the other side, if we start with the back elimination of $(2,3)$, $(2,4)$, or $(2,5)$, then this costs one `fma`. A lower bound for eliminating the remaining edges is $6+3 = 9$. Both values add up to ten, which is below the best value known so far.

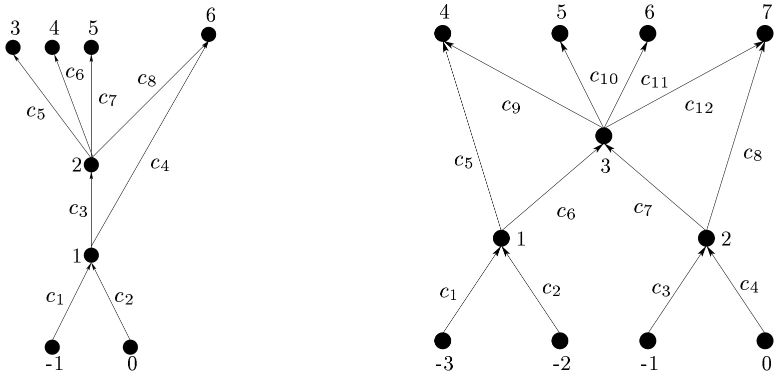

Fig. 12. Lion (left) and bat (right)

Consequently, we must descend into the subtree because we cannot eliminate the possibility that a better solution can be found. This situation is represented as follows:

- $(2,3)$, $(2,4)$, or $(2,5)$ back $\Rightarrow \lambda = 1+6+3 = 10$ ([28]) $\Rightarrow$ branch.

We assume that $(2,3)$ is eliminated first. Because of symmetry, similar results hold for $(2,4)$ and $(2,5)$. The entire decision tree is as follows:

- $(-1,1)$ or $(0,1)$ front $\Rightarrow \lambda = 2+2+8 = 12$ ([28]) $\Rightarrow$ bound.
- $(1,2)$ front $\Rightarrow \lambda = 4+8 = 12$ (Lemma 10) $\Rightarrow$ bound.
- $(1,2)$ back $\Rightarrow \lambda = 2+2+8 = 12$ ([28]) $\Rightarrow$ bound.
- $(1,6)$ back $\Rightarrow \lambda = 2+2+8 = 12$ (Lemma 11) $\Rightarrow$ bound.
- $(2,3)$ $(2,4)$, or $(2,5)$ back $\Rightarrow \lambda = 1+6+3 = 10$ ([28]) $\Rightarrow$ branch.
  - $(-1,1)$ or $(0,1)$ front $\Rightarrow \lambda = 1+3+3+6 = 13$ ([28]) $\Rightarrow$ bound.
  - $(1,2)$ front $\Rightarrow \lambda = 1+3+8 = 12$ (Lemma 10) $\Rightarrow$ bound.
  - $(1,2)$ back $\Rightarrow \lambda = 1+2+4+6 = 13$ ([28]) $\Rightarrow$ bound.
  - $(1,6)$ back $\Rightarrow \lambda = 1+2+4+6 = 13$ (Lemma 11) $\Rightarrow$ bound.
  - $(2,4)$ or $(2,5)$ back $\Rightarrow \lambda = 1+1+8+2 = 12$ ([28]) $\Rightarrow$ bound.
  - $(1,3)$ back $\Rightarrow \lambda = 1+2+4+3 = 10$ ([28]) $\Rightarrow$ branch.
    - $(-1,1)$ or $(0,1)$ front $\Rightarrow \lambda = 1+2+2+2+6 = 13$ ([28]) $\Rightarrow$ bound.
    - $(1,2)$ front $\Rightarrow \lambda = 1+2+3+6 = 12$ (Lemma 10) $\Rightarrow$ bound.
    - $(1,2)$ back $\Rightarrow \lambda = 1+2+2+2+6 = 13$ ([28]) $\Rightarrow$ bound.
    - $(1,6)$ back $\Rightarrow \lambda = 1+2+2+2+6 = 13$ (Lemma 11) $\Rightarrow$ bound.
    - $(2,4)$ or $(2,5)$ back $\Rightarrow \lambda = 1+2+1+6+2 = 12$ ([28]) $\Rightarrow$ bound.
    - $(2,6)$ back $\Rightarrow \lambda = 1+2+1+4+4 = 12$ (Lemma 11) $\Rightarrow$ bound.
  - $(2,6)$ back $\Rightarrow \lambda = 1+1+6+4 = 12$ (Lemma 11) $\Rightarrow$ bound.
- $(2,6)$ back $\Rightarrow \lambda = 1+4+6 = 11$ (Lemma 11) $\Rightarrow$ **solution**.

The solution can be obtained only by eliminating $(2,6)$ first. Furthermore, by Lemma 11, the optimal elimination sequence for the remainder of the graph is given by $[1,2]$, resulting in a optimal cost of eleven `fma`'s. □

Proposition 2 *An optimal face elimination sequence may involve fewer* `fma`*'s than does an optimal edge elimination sequence when applied to the same c-graph.*

Proof. Consider the c-graph on the right side of Figure 12. Because of its shape it is referred to as the *bat graph* (turn it upside down). We show that there is a face elimination sequence whose cost undercuts the cost of an optimal edge elimination sequence for the bat graph. Similarly to the proof of Lemma 11, we consider the following decomposition of the corresponding Jacobian matrix:

$$A = \left(\frac{\partial v_k}{\partial v_l}\right)_{l\in\{-3,-2\}}^{k\in\{4,5,6,7\}}, \quad B = \left(\frac{\partial v_k}{\partial v_l}\right)_{l\in\{-1,0\}}^{k\in\{4,5,6,7\}}.$$

Both A and B correspond to a lion graph and can therefore be accumulated at an optimal cost of 11 `fma`'s, respectively. Consequently, the whole Jacobian can be obtained at a cost of 22 `fma`'sbased on the decomposition into A and B. It is easy to verify that eliminating the two faces corresponding to the transitive dependences $1 \prec 3 \prec 4$ and $2 \prec 3 \prec 7$, followed by the elimination of 1 and 2 and of all the remaining faces, results in a cost of 22 `fma`'s for accumulating the Jacobian. To show that the decomposition into A and B contradicts the concept of edge elimination, we note that

- $(3,4)$ is back eliminated before $(1,3)$ in A,
- $(1,3)$ is back eliminated before $(3,7)$ in B,
- $(3,7)$ is back eliminated before $(2,3)$ in B, and
- $(2,3)$ is back eliminated before $(3,4)$ in A.

The contradiction follows immediately from this cyclic dependence.

Furthermore, we need to verify that there is no other edge elimination sequence that could possibly yield a cost of 22 `fma`'sor less. Considering all possibilities for eliminating the first edge in $\mathbf{G}$, we notice that

- front elimination of c_1 or c_2 or back elimination of c_5, c_6, c_{10}, c_{11}, or c_{12} increases the cost of accumulating A by at least one, and
- front elimination of c_3 or c_4 or back elimination of c_7, c_8, c_9, c_{10}, or c_{11} increases the cost of accumulating B by at least one.

Furthermore, these edge eliminations do not decrease the cost of accumulating the other half of the Jacobian, respectively.

In summary, partitioning the bat graph into two lion graphs allows us to compute the Jacobian at a cost of 22 `fma`'s. The accumulation of the two halves of the Jacobian cannot be expressed as an edge elimination sequence in the bat graph. Any edge elimination sequence is bound to increase the cost of accumulating the Jacobian by at least one. This completes the proof. □

Comments similar to those made in connection with the proof of Proposition 1 apply to the preceding argument. In particular, the optimal edge elimination sequence is contained within the set of all valid face elimination sequences. The bat graph represents an example of an optimal face elimination sequence not being contained within the set of all feasible edge elimination sequences.

Proposition 2 raises questions regarding the vertex-face and edge-face discrepancies. The search for an answer is the subject of ongoing research. After all, Proposition 2 is the reason for introducing the concept of dual c-graphs together with face elimination.

5. Algorithms

The idea of optimizing the computation of Jacobians by different vertex elimination sequences in linearized c-graphs was suggested in [15]. A greedy heuristic strategy based in the *Markowitz criterion* (the vertex with the lowest Markowitz degree is eliminated next) adapted from the theory on sparse linear systems was explored there as well. In [5] the same problem was regarded as a shortest path problem in the vertex metagraph $\mathbf{M}_v$ and an exponential dynamic programming algorithm was discussed. Edge elimination and the corresponding vertex-edge discrepancy were proposed in [24] together with a number of heuristic strategies for optimizing the elimination sequences. Face elimination and the edge-face discrepancy are introduced in this paper.

A variety of test problems were considered in [24] in connection with different strategies for vertex and edge elimination. In most cases the number of `fma`'s required for the accumulation of the Jacobians was decreased by factors between two and ten. In [34] these theoretical savings were shown to result in corresponding runtime savings by generating the code for computing the Jacobian.

First improvements to the greedy Markowitz heuristic [15] were proposed in [25] in the form of the relative Markowitz heuristic for vertex elimination. Further ideas were presented in [24] based on global information such as the number of paths in the c-graph or their overall length. Various new Markowitz-type heuristics for vertex, edge, and face elimination are proposed in [2]. Chapter 8 in [13] contains an example of Markowitz degree-based heuristics not performing well on evolutions. The performance of the Jacobian codes for a Roe-flux CFD kernel generated using various heuristics was investigated in [34].

Jacobian matrices can be evaluated as chained products of local extended Jacobians as described in [14]. The well-known dynamic programming algorithm for optimizing chained products of dense matrices with different dimensions [17] can be adapted for the sparse case at the cost of a reasonable overhead. Instead of working with integers representing the dimensions of the respective dense factors, the sparsity pattern of all intermediate subproducts must be computed explicitly.

Simulated annealing [22] was first applied to the problem of finding nearly optimal vertex elimination sequences in [26]. In this case it proved to be rather straightforward to define a feasible neighborhood relation and corresponding rearrangements. The impact of different annealing schedules based on both homogeneous and inhomogeneous Markov chains on the convergence of the algorithm was shown with the help of numerous test problems. A more theoretical analysis of vertex elimination in the light of logarithmic cooling schedules for inhomogeneous Markov chains was presented in [30].

6. Summary and Conclusion

The aim of this paper was to present a theoretical basis for developing optimized derivative code motivated by the potential runtime savings that could be observed using the techniques mentioned in Section 5. Face elimination in dual computational graphs was introduced as a technique for solving the corresponding combinatorial optimization problem, and its superiority over edge and vertex elimination in linearized computational graphs was shown. Here, superiority is understood in terms of the number of fused multiply-add operations performed by an optimal elimination sequence. Various results were shown that are likely to have an important impact on the further development of elimination algorithms.

It still remains to be seen, whether face elimination can be regarded as the most general Jacobian accumulation procedure, that is, a sequence of multiplications and additions starting from the elementary partial derivatives and yielding all nonzero Jacobian entries. Further investigations are needed to prove this conjecture.

We see two potential fields of application for the techniques presented here and for further algorithms to be developed on the basis of these techniques. First, the preaccumulation of local Jacobians at the level of basic blocks should become a fundamental feature of differentiation enabled compilers. Second, optimized Jacobian code can be generated for routines within heavily used numerical libraries (e.g., BLAS [33] or the NAG numerical library [20]) or simulation codes (e.g., MIT general circulation model [21]). Highly efficient implementations of robust heuristics will probably be preferred in the former case to keep the compile time minimal. More expensive methods such as simulated annealing can be tried if the compile time is not a crucial factor.

Acknowledgments

The face elimination rule presented in this paper is the result of numerous discussions with A. Griewank at Technical University Dresden, Germany.

We thank the referees for many helpful comments and suggestions.

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38

References

1. A. Aho, R. Sethi, and J. Ullman. *Compilers. Principles, Techniques, and Tools.* Addison-Wesley, Reading, MA, 1986.

2. A. Albrecht, P. Gottschling, and U. Naumann. Markowitz-type heuristics for computing Jacobian matrices efficiently. In *Proceedings of International Conference on Computational Science*. Springer, LNCS, 2003. To appear.
3. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer, Berlin, 1999.
4. M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors. *Computational Differentiation: Techniques, Applications, and Tools*, Proceedings Series, Philadelphia, 1996. SIAM.
5. C. Bischof and M. Haghighat. Hierarchical approaches to Automatic Differentiation. In *[4]*, pages 82–94, 1996.
6. C. Bischof, P. Khademi, A. Boucharicha, and A. Carle. Efficient computation of gradients and Jacobians by dynamic exploitation of sparsity in Automatic Differentiation. *Optimization Methods and Software*, 7:1–39, 1997.
7. G. Corliss, C. Faure, A. Griewank, L. Hascoet, and U. Naumann, editors. *Automatic Differentiation of Algorithms – From Simulation to Optimization*, New York, 2002. Springer.
8. G. Corliss and A. Griewank, editors. *Automatic Differentiation: Theory, Implementation, and Application*, Proceedings Series, Philadelphia, 1991. SIAM.
9. A. Curtis, M. Powell, and J. Reid. On the estimation of sparse Jacobian matrices. *J. Inst. Math. Appl.*, 13:117–119, 1974.
10. R. Van der Wijngaart and W. Saphir. On the efficacy of source code optimizations for cache-based processors. Technical report, NAS, June 2000.
11. M. Garey and D. Johnson. *Computers and Intractability - A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, San Francisco, 1979.
12. J. Gilbert. A note on the NP-completeness of vertex elimination on directed graphs. *J. Alg. Disc. Meth.*, 1(3):292–294, September 1980.
13. A. Griewank. *Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Applied Mathematics. SIAM, Philadelphia, 2000.
14. A. Griewank and U. Naumann. Accumulating Jacobians as chained sparse matrix products. *Mathematical Programming*, 3(95):555–571, 2003.
15. A. Griewank and S. Reese. On the calculation of Jacobian matrices by the Markovitz rule. In *[8]*, pages 126–135, 1991.
16. K. Herley. A note on the NP-completeness of optimum Jacobian accumulation by vertex elimination. Presentation at: Theory Institute on Combinatorial Challenges in Computational Differentiation, 1993.
17. E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, Rockville, 1978.
18. M. Iri. History of Automatic Differentiation and rounding error estimation. In *[8]*, pages 3–17, 1991.
19. R. Jessani and M. Putrino. Comparison of single- and dual-pass multiply-add fused floating-point units. *IEEE Trans. Comp.*, 47(9):927–936, 1998.
20. NAG Numerical Libraries. `http://www.nag.co.uk/numeric/numerical_libraries.asp`, NAG Ltd., Oxford, UK.
21. J. Marshall, C. Hill, L. Perelman, and A. Adcroft. Hydrostatic, quasi-hydrostatic and nonhydrostatic ocean modeling. *J. Geophysical Research*, 102, C3:5,733–5,752, 1997.
22. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087–92, 1953.
23. W. Miller and C. Wrathall. *Software for Roundoff Analysis of Matrix Algorithms*. Academic Press, New York, 1980.
24. U. Naumann. *Efficient Calculation of Jacobian Matrices by Optimized Application of the Chain Rule to Computational Graphs*. PhD thesis, Technical University Dresden, Feb. 1999.
25. U. Naumann. An enhanced Markowitz rule for accumulating Jacobians efficiently. In K. Mikula, editor, *ALGORITHMY'2000 Conference on Scientific Computing*, pages 320–329. Slovak University of Technology, Bratislava, Slovakia, September 2000.
26. U. Naumann. Cheaper Jacobians by Simulated Annealing. *SIAM J. Opt.*, 3(13):660–674, March 2002.
27. U. Naumann. Elimination techniques for cheap Jacobians. In *[7]*, pages 247–253, 2002.
28. U. Naumann. On optimal Jacobian accumulation for single expression use programs. Preprint ANL-MCS/P944-0402, Argonne National Laboratory, 2002.
29. U. Naumann and A. Albrecht. Combinatorial optimization methods for fast derivative code. Case for Support EPSRC Grant GR/R38101/01, University of Hertfordshire, Hatfield, UK, 2001.

30. U. Naumann and P. Gottschling. Prospects for Simulated Annealing in Automatic Differentiation. In K. Steinhöfel, editor, *SAGA 2002 - Stochastic Algorithms, Foundations and Applications*, volume 2264 of *LNCS*. Springer, Berlin, 2001.
31. G. Newsam and J. Ramsdell. Estimation of sparse Jacobian matrices. *SIAM J. Alg. Dis. Meth.*, 4:404–417, 1983.
32. D. Rose and R. Tarjan. Algorithmic aspects of vertex elimination on directed graphs. *J. Appl. Math.*, 34(1):176–197, January 1978.
33. Basic Linear Algebra Subprograms. `http://www.netlib.org/blas/`.
34. M. Tadjouddine, S. Forth, and J. Pryce. AD tools and prospects for optimal AD in CFD flux calculations. In *[7]*, pages 255–261, 2002.
35. M. Tadjouddine, S. Forth, J. Pryce, and J. Reid. Performance issues for vertex elimination methods in computing Jacobians using Automatic Differentiation. In *Proceedings of the ICCS 2000 Conference*, volume 2330 of *Springer LNCS*, pages 1077–1086, 2002.
36. R. Wengert. A simple automatic derivative evaluation program. *Comm. ACM*, 7:463–464, 1964.