Optimizing the Accumulation of Jacobians by Edge Elimination in the Computational Graph

Uwe Naumann

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Optimizing the Accumulation of Jacobians by Edge Elimination in the Computational Graph

Uwe Naumann

N° 3659

April 1999

THÈME 2

ISSN 0249-6399

INRIA
SOPHIA ANTIPOLIS

Optimizing the Accumulation of Jacobians by Edge Elimination in the Computational Graph

Uwe Naumann *

Thème 2 — Génie logiciel
et calcul symbolique
Projet TROPICS

Rapport de recherche n° 3659 — April 1999 — 25 pages

Abstract: The chain rule - fundamental for Automatic Differentiation (AD) - can be applied to computational graphs representing vector functions in arbitrary orders resulting in different operations counts for the calculation of their Jacobian matrices. Very few authors have dealt with this interesting subject so far and there is no generally accepted terminology for handling these combinations of the forward and reverse modes of AD. The minimization of the number of arithmetic operations required for the calculation of the complete Jacobian leads to a computationally hard combinatorial optimization problem.

In this paper we will give a formal description of this problem, which also is sometimes referred to as the *cross-country elimination problem in computational graphs*, in terms of a shortest path problem in the so-called metagraph. The well-known strategy of eliminating vertices will be refined by introducing the elimination of edges. We will show that edge elimination is in general superior to vertex elimination with respect to the operations count. As an outlook we will present a selection of methods for solving the *general edge elimination problem* heuristically

Key-words: Edge elimination, Vertex-edge discrepancy, Metagraph

* e-mail: Uwe.Naumann@sophia.inria.fr

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, B.P. 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : 04 92 38 77 77 - International : +33 4 92 38 77 77 — Fax : 04 92 38 77 65 - International : +33 4 92 38 77 65

Optimisation de l'Accumulation des Jacobiennes par l'Élimination des Arcs du Graphe de Calcul

Résumé : Nous présentons une nouvelle approche de calcul de Jacobiennes basée sur l'élimination des arcs dans le graphe de calcul. L'objectif de cette méthode est la minimisation du nombre de multiplications scalaires nécessaires à l'accumulation de la Jacobienne d'une fonction vectorielle à un argument. Ce rapport décrit le problème d'optimisation combinatoire à résoudre.

Mots-clés : Élimination des arcs, minimisation du nombre de multiplications, métagraphe.

Contents

1 Fundamentals

1.1 Motivation and outline

Research in the field of **Automatic Differentiation (AD)** dates back as far as 1964, when R. E. Wengert ([Wen64]) first proposed the automation of the calculation of derivatives by a program in form of the basic forward mode. Since then a lot of work has been done in order to make AD faster and more widely applicable (see [CoGr91] and [BBCG96]). AD is a fast and convenient way for calculating directional derivatives of vector functions numerically up to machine precision provided that it is given as a computer program. Notice, that AD is completely different from the approach to computing derivatives numerically through divided differences.

The computation of the Jacobian matrix of a vector function

$$F: \quad \mathbb{R}^n \supseteq D \to \mathbb{R}^m \quad : \quad \mathbf{x} \mapsto \mathbf{y} = F(\mathbf{x}) \tag{1}$$

is essential for many numerical algorithms. For practical reasons most currently available AD software packages provide only two approaches to calculating the Jacobian matrix of F at the current argument – the forward and the reverse modes which are based on the application of the chain rule to F in two different ways. However, the chain rule can be applied to computational graphs of vector functions in an arbitrary order, which results in different operations counts for the calculation of the Jacobian matrix J. The general task of efficiently evaluating J using an approach which is sometimes referred to as *cross-country elimination* is conjectured to be NP-hard. The fact that Jacobians can be calculated by eliminating intermediate vertices in computational graphs is well known since the late 1980's. There are a few papers by various authors that motivate a closer look at this topic [GrRe91], [Bis96]. However, there is no generally accepted terminology for dealing with these combinations of the forward and reverse modes of AD, so far.

Our objective is to compute J as fast as possible. Therefore, we will focus on the minimization of the number of scalar multiplications involved in this process, by finding optimal application sequences of the chain rule. The approach presented here must be seen in the context of making AD more efficient and it is supposed to enable the users to gain deeper insight into the structure of problems. Our goal is to provide the framework for the search of a solution of the computationally hard combinatorial optimization problem which is to minimize the number of scalar multiplications involved in the accumulation of the complete Jacobian.

1.2 Evaluation routines

In our approach we expect the vector function F to be such that it can be decomposed into a sequence of scalar valued functions $\varphi : \mathbb{R}^d \supseteq D_\varphi \to \mathbb{R}$ that take a vector $\mathbf{u} \in \mathbb{R}^d$ as their argument and return a value $w = \varphi(\mathbf{u})$. We call such functions **elemental**. In most cases the vector function F is given as a computer program written in some high-level programming

language such as `C`, `C++` or `Fortran`. This specification of F is called **evaluation routine** if it can be broken down into a sequence of scalar assignments of the form

$$v_j = \varphi_j(v_i)_{i \in P_j} \quad \in I\!R, \tag{2}$$

by assigning the result of every elemental function φ_j that occurs in the program to a unique intermediate variable v_j. By P_j [S_j] we denote the set of indices of all predecessors [successors] of the intermediate variable v_j with respect to the implicit order within the evaluation routine. Thus, φ_j acts on an argument vector $\mathbf{u} \in I\!R^d$ where $d = |P_j|$. The fact that the result v_j of an elemental function φ_j depends on some argument v_i directly is denoted by $v_i \prec v_j$ and the transitive closure of this relation by $v_i \prec^* v_j$.

We assume the evaluation procedure of interest to be fixed in the sense that there are no conditional branches in it. By unrolling loops and by substituting calls to subroutines with the corresponding code it is conceptually possible to assign the result of every elemental function to a unique intermediate variable. According to Equation (2) we can represent this evaluation routine as the corresponding sequence of scalar assignments which we will refer to as the **evaluation trace** (also **single assignment code**) For a given evaluation trace we distinguish between the n independent, $X = \{v_{1-n}, \ldots, v_0\}$, the m dependent, $Y = \{v_{q+1}, \ldots, v_{q+m}\}$, and q intermediate variables $Z = \{v_1, \ldots, v_q\}$, where Z and Y represent the results of the $q+m$ elemental functions in F. The sets X, Z, and Y are assumed to be disjoint. Since our objective is to calculate the complete Jacobian of a given vector function which consists of the partial derivatives of the m dependent variables $y_0, \ldots, y_{m-1}$ with respect to each of the n independent variables $x_0, \ldots, x_{n-1}$ it is fundamental to assume the following:

Assumption 1.1 *Given an evaluation routine of a vector function defined by Equation (1), for some fixed argument the elemental functions φ_j occurring in Equation (2) are well defined for $j = 1, 2, \ldots, q+m$ and have jointly continuous partial derivatives*

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

of their respective arguments on some neighborhood $\mathcal{D}_j \subset I\!R^{n_j}$ with $n_j \equiv |P_j|$.

1.3 A few words on Automatic Differentiation

The ideas behind AD are based on the application of the **chain rule** to evaluation routines of vector functions F as described by Equation (1). AD is based on the application of the **chain rule** one possible representation of which is given by

$$\frac{\partial v_j}{\partial x_i} = \sum_{l \in P_j} \frac{\partial}{\partial v_l} \varphi_j(v_k)_{k \in P_j} \cdot \frac{\partial v_l}{\partial x_i} \tag{3}$$

and which is illustrated in Figure 1. Considering the dependency of v_j on the independent variables x_i we can calculate $\partial v_j / \partial x_i$ as the sum over v_j's predecessor's sensitivities $\partial v_l / \partial x_i$, $l \in P_j$, each of them multiplied by the corresponding local partial derivative c_{jl} of the

elemental function φ_j with respect to the variables v_l. Analogous, we can look at the dependency of y_k on the assumed independent variable v_j. In this case we compute $\partial y_k/\partial v_j$ as the sum over y_k's sensitivities with respect to the successors of v_j, again multiplied with the local partial derivatives of the elemental functions φ_l, $l \in S_j$. Building on the representation of F as a single assignment code in connection with Assumption 1.1 about the differentiability of the occurring elemental functions follows by the chain rule that F is well defined and once continuously differentiable. Its Jacobian matrix $J(\mathbf{x}) \equiv \nabla_{\mathbf{x}} F(\mathbf{x}) \in I\!R^{m \times n}$ is computable from the local partial derivatives c_{ji}.

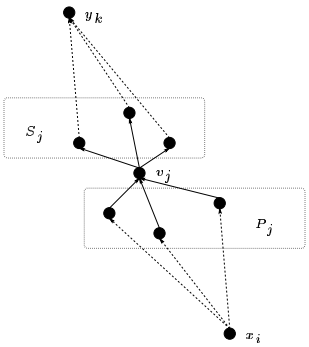


Figure 1: Chain Rule

In this section we will give brief descriptions of the forward and reverse modes of AD. This short introduction into AD will not be exhaustive by far. Those who are not familiar with the subject should refer to [RaCo96] and [Iri91]. More advanced aspects of AD are covered by [Chri93], [Kub91], [GUW97], [BKBC96], and others.

1.3.1 Forward mode

The (scalar) **forward mode** of AD exploits the chain rule in the usual way. Consider an evaluation routine given as a single assignment code and let again P_j denote the set of all indices $l < j$ such that v_j depends directly on $v_l \in X \cup Z$. If we substitute for a given single assignment code $\dot{v}_j \equiv \partial v_j/\partial x_i$ then the chain rule represented by Equation (3) takes the form

$$\dot{v}_j = \sum_{l \in P_j} c_{jl} \cdot \dot{v}_l \qquad \text{for} \quad j = 1, \ldots, p+m \tag{4}$$

to compute the partial derivatives of the intermediate and dependent variables with respect to each of the independent variables. The initialization of $\dot{\mathbf{x}}$ followed by one forward sweep corresponding to Equation (4) gives us $\dot{\mathbf{y}} = J \cdot \dot{\mathbf{x}}$. Thus, we can compute the complete Jacobian performing exactly n forward sweeps. If $\mathbf{e}_i$ $(i = 0, \ldots, n-1)$ denotes the i-th Cartesian basis vector in $I\!R^n$ then setting $\dot{\mathbf{x}}_i = \mathbf{e}_i \in I\!R^n$ for $i = 0, \ldots, n-1$ gives us successively the n columns of $J \in I\!R^{m \times n}$. Equation (4) can be regarded as the non-incremental version of the tangent accumulation procedure. Using the abbreviation $a \mathrel{+}= b$ for $a := a + b$ as in `C` we can write it in its incremental form as

$$\dot{v}_j \mathrel{+}= c_{jl} \cdot \dot{v}_l \quad \text{for} \quad j \in S_l \quad \text{and} \quad l = 1-n, \ldots, p.$$

In this case it is necessary to initialize $\dot{v}_j = 0$ for $j = 1, \ldots, p+m$.

Notation. We use bold letters for vectors to be able to distinguish them from scalar values. Hence, $\dot{\mathbf{x}}_i$ denotes one possible initialization for calculating a directional derivative

in forward mode while x_i represents the i-th component of the argument vector $\mathbf{x}$. Matrices will be denoted by capital letters.

1.3.2 Reverse mode

A different way to apply the chain rule is exploited in the (scalar) **reverse mode** of AD. Introducing the notation $\bar{v}_l \equiv \partial y_i / \partial v_l$ we get the following equation which describes the backward propagation of gradients in the reverse mode:

$$\bar{v}_l = \sum_{j \in S_l} c_{jl} \cdot \bar{v}_j \qquad \text{for} \quad l = p, \ldots, 1-n. \tag{5}$$

Here, the chain rule is applied to the so-called **adjoints** $\bar{v}_l$. After initializing $\bar{\mathbf{y}}$, one reverse sweep gives us $\bar{\mathbf{x}} = \bar{\mathbf{y}}^T \cdot J$. The complete Jacobian can be computed by performing m reverse sweeps. Setting $\bar{\mathbf{y}}_i = \mathbf{e}_i \in I\!R^m$ for $i = 0, \ldots, m-1$ delivers the m rows of J.

Equation (5) represents the non-incremental version of the adjoint evaluation procedure. Rather than using this method the accumulation of the adjoints can also be regarded as an incremental approach with $\bar{v}_l += c_{jl} \cdot \bar{v}_j$ for $l \in P_j$ and $j = p+m, \ldots, 1$. Again we assume that the adjoints $\bar{v}_l$ have been initialized to zero for $l = p, \ldots, 1-n$. The main difference between the two methods is that the non-incremental approach requires global information represented by arguments of several elemental functions whereas the incremental does not. Notice that in the forward mode of AD it is the other way around.

Rather then accumulating the Jacobian by performing n subsequent scalar forward sweeps [m scalar reverse sweeps] we could also propagate bundles of $q \geq 1$ tangents [gradients] during one forward [reverse] sweep. This is equivalent to computing the matrix product

$$(I\!R^{m \times q} \ni) \;\; \dot{Y} = \nabla_{\mathbf{x}} F(\mathbf{x}) \, \dot{X} \qquad \left[(I\!R^{n \times q} \ni) \;\; \bar{X} = \bar{Y}^T \nabla_{\mathbf{x}} F(\mathbf{x}) \right].$$

If we set $q = n$ [$q = m$] and initialize $\dot{X}$ [$\bar{Y}$] as the corresponding identity matrix we can apply the local elementary partial derivatives to whole vectors, thus ending up with just one forward [reverse] sweep. This is what we will refer to as the forward [reverse] **vector mode** of AD.

1.4 From evaluation programs to computational graphs

The relation between the variables in a given evaluation routine can be visualized by a directed acyclic graph $CG = (V, E)$ which contains all information needed to be able to compute the entries of the Jacobian. From now on we will refer to CG as the **computational graph** (also **c-graph**) We will distinguish between $n \equiv |X|$ minimal [independent], $p \equiv |Z|$ intermediate and $m \equiv |Y|$ maximal [dependent] vertices and analogous to the evaluation trace we will enumerate them as follows:

$$X \equiv \{v_{1-n}, \ldots, v_0\}, \quad Z \equiv \{v_1, \ldots, v_p\}, \quad Y \equiv \{v_{p+1}, \ldots, v_{p+m}\}.$$

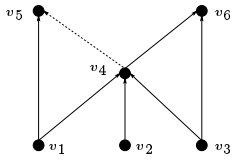
The numbering $\mathcal{I} : V(CG) \to \{1-n, \ldots, p+m\}$ of the vertices in CG has to be **consistent** in the sense that it induces a topological order of the vertices of CG with respect to dependency, i.e.

$$v_i \prec^* v_j \quad \Rightarrow \quad \mathcal{I}(v_i) < \mathcal{I}(v_j).$$

There exists a directed edge (i, j) connecting a vertex v_i with a vertex v_j in CG if v_j directly depends on v_i in F. According to Assumption 1.1 labels c_{ji}, representing the local partial derivatives, are attached to all edges $(i, j) \in E$. The $v_i \in X$ are the only minimal and the $v_j \in Y$ the only maximal vertices in CG. All intermediate vertices $v_k \in Z$ lie on a path connecting an independent with a dependent vertex. We observe that v_i depends on v_j in F if and only if there exists a path connecting v_i with v_j in CG.

2 The Chain Rule Applied to Computational Graphs

As described in the previous section the forward and reverse modes of AD can be used for computing the complete Jacobian J of a vector function F given by Equation (1). Both modes represent special ways to apply the chain rule to an evaluation routine or its representation as a single assignment code. We are looking for a method of transforming the corresponding c-graph of F such that we get the Jacobian J at the lowest possible cost in terms of the number of scalar multiplications involved in this process. In fact, if by successively eliminating all vertices representing intermediate variables in the underlying evaluation program (which is equivalent to eliminating all edges having either an intermediate vertex as source, or having such a vertex as target, or both, i.e. all *intermediate edges*) we get to a stage, at which the c-graph represents a subgraph of the complete bipartite graph $K_{n,m}$ and the labels $c_{ji} \equiv \partial y_j / \partial x_i$ $(i = 0, \ldots, n-1, j = 0, \ldots, m-1)$ on the edges connecting the minimal vertices with the maximal ones are exactly the nonzero entries of the Jacobian matrix of F. The elimination of intermediate edges represents the elemental action that we will build on in our approach. It can be regarded as the chain rule applied to evaluation routines represented by graphs.



Considering (the scalar mode of) AD applied to a (scalar) c-graph CG we can associate scalar *derivative objects* $\tilde{v}_j$ with the vertices $v_j \in CG$. In particular we have $\tilde{v}_j \equiv \dot{v}_j$ in forward mode and $\tilde{v}_j \equiv \bar{v}_j$ in reverse mode. Later we will distinguish between different types of derivative values depending on both the mode and the type of the graph (see Section 2.5).

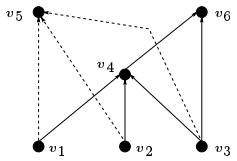


Figure 2: $(4, 5)$

2.1 Forward elimination of edges

Suppose we want to eliminate the term for a particular intermediate vertex v_i from the recurrence described by Equation (4). This is equivalent to eliminating the corresponding

edge (i,j) from the c-graph. It may be regarded as a *forward* elimination in the sense that (i,j) is eliminated as part of the computation of $\dot{v}_j$ representing the derivative value of its target. Our discussion will be supported by Figure 2 where the elimination of $(4,5)$ is illustrated.

We start with the isolation of the term that contains the factor $\dot{v}_i$ followed by the substitution of the expression corresponding to $\dot{v}_i$ by Equation (4):

$$\dot{v}_j = c_{ji} \cdot \dot{v}_i + \sum_{i \neq l \in P_j} c_{jl} \cdot \dot{v}_l = \sum_{l \in P_i} c_{ji} \cdot c_{il} \cdot \dot{v}_l + \sum_{i \neq l \in P_j} c_{jl} \cdot \dot{v}_l.$$

This results in a new calculation rule for $\dot{v}_j$ without any terms containing $\dot{v}_i$:

$$\dot{v}_j = \sum_{l \in \tilde{P}_j} \tilde{c}_{jl} \cdot \dot{v}_l \quad \text{where} \quad \tilde{P}_j \equiv P_i \cup P_j \setminus \{i\} \quad \text{and} \quad \tilde{c}_{jl} \equiv c_{jl} + c_{ji} \cdot c_{il}. \tag{6}$$

Equation (6) determines the way in which the labels on the edges in the c-graph have to be updated. Graphically, the forward elimination of an edge (i,j) is equivalent to connecting all predecessors of v_i with v_j (provided they have not been connected before as we do not permit multiple edges) followed by updating the existing or generating the new local partial derivatives and, finally, the deletion of (i,j). In correspondence with the chain rule we multiply the values of successive edges (i,j) and (j,k) whereas we add the values of parallel edges having the same source and the same target.

We observe that forward elimination of edges is exactly what happens during the non-incremental forward propagation of tangents in the forward mode of AD. Computing $\dot{v}_j$ is equivalent to the successive forward elimination of all intermediate edges leading into v_j. This process will terminate as soon as all these edges have a minimal source $x_i \in X$. At this stage the labels on these edges are exactly the partial derivatives of v_j with respect to the corresponding x_i.

The forward elimination of an edge (i,j) involves a number of scalar multiplications that is equal to the cardinality of the predecessor set of its source v_i. We will call this number the **in-degree** or **forward Markowitz degree** of (i,j) and denote it by $|P_{(i,j)}|$.

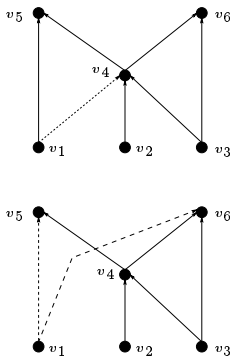


Figure 3: $(1,4)$

2.2 Backward elimination of edges

Equation 5 describes the application of the chain rule to the adjoints in the reverse mode of AD. Again, we intent to eliminate the term for an intermediate vertex v_i from this recurrence, which is equivalent to the elimination of the corresponding edge (l,i) from the c-graph. This edge elimination may be regarded as a *backward* elimination as it is part of the computation of $\bar{v}_l$, which is always built on the adjoint values of successors of v_l in the c-graph.

Analogous to the forward case we isolate the product involving $\bar{v}_i$ followed by the substitution of the recurrence given for $\bar{v}_i$ by Equation (5) to get

$$\bar{v}_l = \sum_{j \in \tilde{S}_l} \tilde{c}_{jl} \cdot \bar{v}_j \quad \text{where} \quad \tilde{S}_l \equiv S_i \cup S_l \setminus \{i\} \quad \text{and} \quad \tilde{c}_{jl} \equiv c_{jl} + c_{il} \cdot c_{ji}. \tag{7}$$

The graphic interpretation of the backward edge elimination is shown in Figure 3. As in the case of forward elimination we simply have to insert new edges connecting v_l with all successors of v_i (if they do not exist already) and generate new or update the existing edge labels correspondingly. Finally, (l, i) is removed from the c-graph. This is what happens during the backward propagation of gradients in the non-incremental reverse mode of AD. Computing $\bar{v}_l$ is equivalent to the successive backward elimination of all intermediate edges emanating from v_l in the c-graph. The process terminates when all these edges have maximal sinks. It takes $|S_{(i,j)}|$ scalar multiplications to eliminate an edge (i, j) backward. $|S_{(i,j)}|$ denotes the **out-degree (backward Markowitz degree)** of (i, j) which is equal to the number of successors of the target v_j.

2.3 Termination

The repeated application of the chain rule yields a general expression for the partial derivative c_{kl}:

$$c_{kl} \equiv \sum_{(l,\ldots,k)} \prod_i c_{j_i j_{i+1}}$$

which is to be understood as the sum over all paths $(l, \ldots, k)$ connecting v_l with v_k of the chained products of the local partial derivatives labeling all edges in $(l, \ldots, k)$. In order to avoid confusion we should examine the c_{kl} more closely. Let v_l and v_k be connected by d distinct paths in CG. Then there are exactly

$$\sum_{i=1}^{d} \binom{d}{i} \quad = \quad 2^d - 1$$

different values c_{kl}^h, $(h = 1, \ldots, 2^d - 1)$ that could possibly be computed while traversing the metagraph. Consider the subgraph of CG which is induced by the vertices lying on paths connecting v_l with v_k and having v_l as its only minimal and v_k as its only maximal vertex. We could think of it as the c-graph for the scalar valued function $v_k = f_k(v_l)$ regarding all arguments to binary functions "coming from outside the c-graph of f" as constants. In this case $f_k'(v_l) = \partial v_k / \partial v_l$ is well-defined and we set $c_{kl} = f_k'(v_l)$. We then have $c_{kl} = c_{kl}^{2^d - 1}$. So, we have to eliminate all edges lying on paths connecting v_l with v_k in order to calculate the unique final value for c_{kl}.

Let the number of distinct paths in CG be denoted by $\mathcal{N}$ and let $\mathcal{L}$ stand for the sum of their total lengths. The length of a given path connecting an independent vertex with a dependent one is defined as the number of edges it consists of. If $\mathcal{N}$ is the number of

products of the form $c_{j_1 i_1} c_{j_2 i_2} \dots c_{j_k i_k}$ which are involved in the process of calculating the entries of the Jacobian naively one after the other then $\mathcal{L}$ is precisely the number of scalar multiplications that are performed.

Let $n^b(v_i)$ $[n^f(v_i)]$ denote the number of different paths leading into [emanating from] a vertex $v_i \in CG$. Initialize $n^b(v_i) = 1$ for $i \in X$ and $n^f(v_i) = 1$ for $i \in Y$. If we denote the total length of all paths leading into [emanating from] a given vertex v_i by $l^b(v_i)$ $[l^f(v_i)]$ we get with

$$l^b(v_i) = n^b(v_i) + \sum_{j \in P_i} l^b(v_j) \quad \text{and} \quad l^f(v_i) = n^f(v_i) + \sum_{j \in S_i} l^f(v_j)$$

for the total length of all paths in CG:

$$\mathcal{L} = \sum_{j \in X} l(v_j) = \sum_{j \in Y} l(v_j) = \sum_{j \in X} l^f(v_j) = \sum_{j \in Y} l^b(v_j).$$

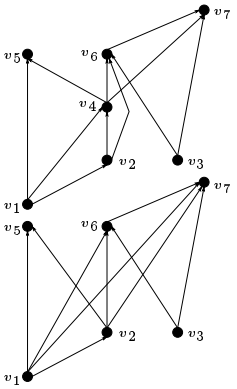


Figure 4: v_4

We have initialized $l^b(v_i) = 0$ for $i \in X$ and $l^f(v_i) = 0$ for $i \in Y$. Regarding the effect which the elimination of an edge from CG will have on the total length of all different paths consider all paths containing (i, j) of which there are $n^b(v_i) \cdot n^f(v_j)$. Each of them is either reduced in length by one or coalesces with another one so that

$$\mathcal{L}_{new} \leq \mathcal{L}_{old} - n^b(v_i) \cdot n^f(v_j).$$

This is true for both the forward and the backward elimination of (i, j). We observe that $\mathcal{L}$ is strictly monotonically decreasing during the process of eliminating edges in c-graphs. We may conclude that edge elimination will always come to an end. This seems to be obvious but it is certainly crucial for our approach. Furthermore, both $\mathcal{L}$ and $\mathcal{N}$ will play an important role in the design of reliable heuristics for solving the combinatorial optimization problem which will be introduced in Section 3.

2.4 Vertex-edge discrepancy

Obviously, the forward [backward] elimination of all out-edges [in-edges] of a vertex v_i leads to the elimination of v_i itself (Figure 4). Consequently, the elimination of an intermediate vertex v_i involves $|P_i| \cdot |S_i|$ scalar multiplications which is usually referred to as the **Markowitz degree** of v_i [GrRe91]. So far, the application of the chain rule to c-graphs has been interpreted as vertex elimination. Even the few attempts to minimize the number of multiplications needed to calculate the Jacobian are based on the idea of eliminating vertices (see for example [Bis96] or [GrRe91]). In fact, this is not entirely true since there are problems where the optimal vertex elimination sequence does not minimize the number of multiplications. Let us illustrate this with the help of an example displayed in Figure 5. It shows a c-graph of a problem with two independent, five dependent, and two intermediate

variables. There are two different vertex and numerous different edge elimination orders. The two vertex elimination orders result in

$$((v_i, v_k) \hat{=} 5 + 10 =) \;\; 15 \quad \text{and} \quad ((v_k, v_i) \hat{=} 4 + 10 =) \;\; 14$$

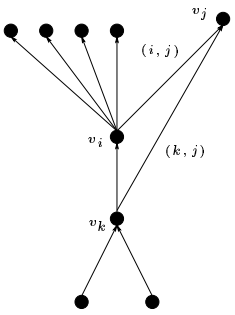


Figure 5: Surprise

multiplications. So, using a pure vertex elimination strategy on this very simple example gives us the Jacobian for a cost of at least 14 multiplications. Now, suppose we eliminate (i, j) separately before v_k followed by the elimination of v_i. This would take only

$$(((i, j), v_k, v_i) \quad \hat{=} \quad 1 + 4 + 8 =) \;\; \mathbf{13}$$

multiplications which is obviously less than $14 = \min\{(v_i, v_k), (v_k, v_i)\}$ in the pure vertex elimination mode. Thus, any possible vertex elimination sequence will not be able to compute the Jacobian using the minimal number of multiplications in this example. The only way to get the required minimum is to find an optimal edge elimination order.

The fact that vertex elimination is not necessarily optimal gives rise to a fundamental question: *"How bad" is the "best" vertex elimination sequence compared to the "best" edge elimination sequence in general?* We have shown that this **vertex-edge discrepancy** does not exceed a factor of $(\sqrt{2}(2 - \sqrt{2}))^{-1}$ for c-graphs containing two intermediate vertices. The problem remains unsolved for the general case involving c-graphs with $p > 2$ intermediate vertices. However, building on numerous test results we conjecture that the general vertex-edge discrepancy will be less or equal to 2.

2.5 Classification

So far, we have assumed the derivative objects associated with the vertices in the c-graph to be scalars or a set of independent scalar values in vector mode where we have an element-wise application of the local partial derivatives to the components of a vector. In general, we will distinguish between different *types* of c-graphs, depending on whether the values associated with each of their vertices are scalars or vectors. Moreover, we will look at different *modes* of calculating directional derivatives based on the number of tangents [gradients] that are computed by one forward [reverse] sweep through the c-graph.

Definition 2.1 *A c-graph CG is called* **vector c-graph** *if the elemental functions Φ_j associated with each its vertices v_j are vector valued, i.e. if*

$$\Phi_j : I\!R^{d_j^{\mathtt{pred}}} \supseteq D \to I\!R^{d_j} \qquad for \quad j = 1, \ldots, p + m$$

and with $d_j^{\mathtt{pred}} = \sum_{i \in P_j} d_i$. If $d_j = 1$ for all $j = 1 \ldots, p + m$ then CG is called **scalar c-graph** *and the elemental functions are denoted by φ_j.*

Derivative vectors $\tilde{\mathbf{v}}_j \in I\!R^{|V|}$ are associated with every vertex of a vector c-graph.

Definition 2.2 *A c-graph CG is said to be regarded in the context of* **vector mode** *of AD if during one (forward, reverse, or cross-country) sweep the local partial derivatives labeling the edges of CG are applied to the $q \geq 1$ components of a vector simultaneously. We speak of the* **scalar mode** *if $q = 1$.*

Building on the above definitions we introduce the classification shown in Figure 6. The *scalar mode on scalar c-graphs* represents the simplest form of looking at the propagation of directional derivatives in a c-graph of a given vector function. The extension to bundles of tangents and gradients leads to the *vector mode on scalar c-graphs* which we have referred to as the vector mode of AD up to this point.

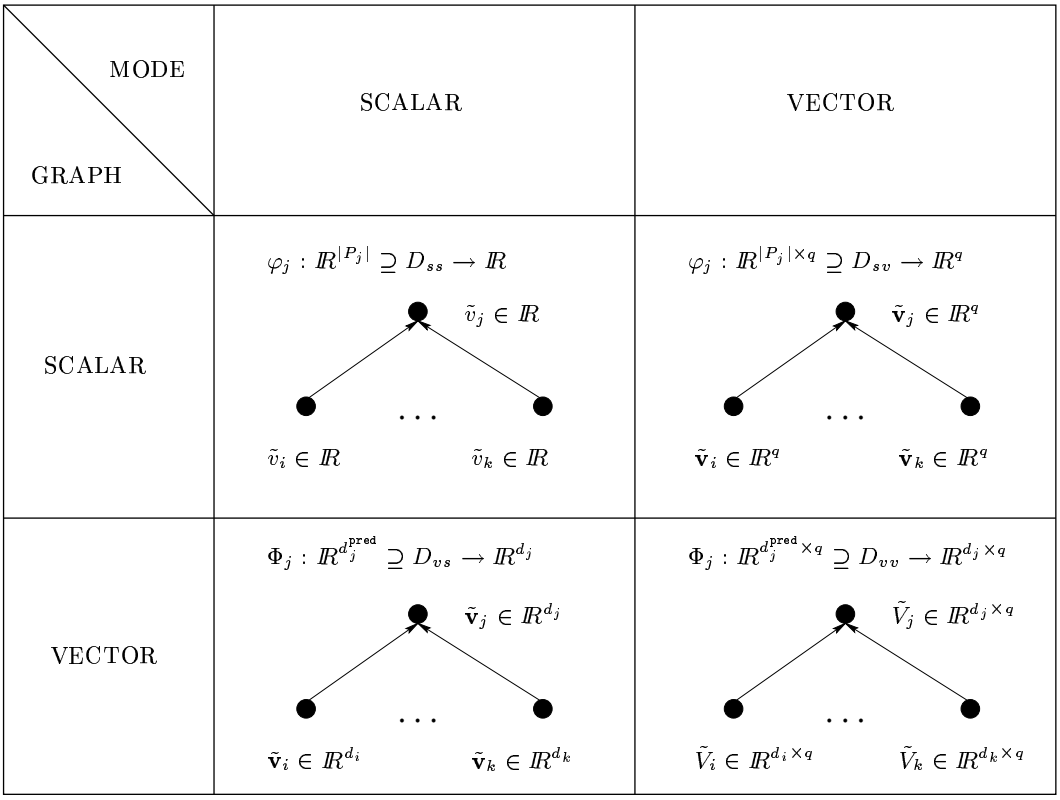


Figure 6: Classification

There is no need to introduce the *scalar mode on vector c-graphs* separately as it is useful to regard it as a special case of the *vector mode on vector c-graphs*, which we will also refer to as the **general vector mode**. In the general vector mode we allow the elemental functions obtained by recording the evaluation trace to be vector-valued. Recapitulate

that, so far, we have always decomposed the evaluation program into a sequence of scalar assignments of the form $v_j = \varphi_j(v_i)_{i \in P_j}$. By Assumption 1.1 and using the well-known analytical expressions for the partial derivatives of the intrinsic functions provided by most high-level programming languages this allows the straight forward computation of the values of the local partial derivatives of any $\varphi_j : I\!R^{|P_j|} \supseteq D \to I\!R$ for $j = 1, \ldots, p+m$ with respect to each of their arguments v_i $(i \in P_j)$ for the given value. Classic AD exploits this information for the computation of directional derivatives by forward [backward] propagating tangents [gradients] or the corresponding bundles in vector mode. Suppose we extend the term *elemental function* such that we do not necessarily restrict its range to $I\!R$ but allow *elemental* assignments to vectors as unique intermediate variables. This enables us to process more complex evaluation programs efficiently by, for example, regarding subroutines as elementals as exploited in the *hierarchical elimination* approach. This enables us to structure large-scale problems hierarchically which represents the only practicable way to apply the cross-country elimination method to problems resulting in c-graphs which consist of several ten thousand intermediate vertices. The c-graphs of most of these problems may not even fit into the memory of the available computer systems. So, a hierarchical treatment will even make sense if the chain rule is not applied in the cross-country mode but in the basic forward and reverse modes of AD.

In general vector mode a single assignment code is given by

$$\mathbf{v}_j = \Phi_j(\mathbf{v}_i)_{i \in P_j} \quad \text{where} \quad \Phi_j : I\!R^{|P_j|} \supseteq D \to I\!R^{d_j} \quad \text{for} \quad j = 1, \ldots, p+m$$

and the local partial derivatives become local Jacobian matrices

$$C_{jk} = \frac{\partial}{\partial \mathbf{v}_k} \Phi_j(\mathbf{v}_i)_{i \in P_j} \in I\!R^{d_j \times d_k} \quad \text{with} \quad j \in \{1, \ldots, p+m\} \quad \text{and} \quad k \in \{1-n, \ldots, p\}.$$

Analogous to the classic vector version of AD we can write down expressions for the forward and reverse modes. Again, we will propagate bundles of $q \geq 1$ tangents [gradients] forward [backward].

General forward vector mode (non-incremental form):

$$(I\!R^{d_k \times q} \supseteq) \ \dot{V}_k = \sum_{j \in P_k} C_{kj} \dot{V}_j \qquad \text{for} \quad k = 1, \ldots, p+m.$$

General reverse vector mode (incremental form):

$$(I\!R^{q \times d_j} \supseteq) \ \bar{V}_j \mathrel{+}= \bar{V}_k C_{kj} \qquad \text{for} \quad j \in P_k \quad \text{and} \quad k = p+m, \ldots, 1.$$

Expressions for the two remaining, but for reasons described in Section 1.3 not very practicable modes can be derived easily. Notice that in general reverse mode we interpret the $\bar{V}_j$ as bundles of q row vectors. Let us have a closer look at the elimination of edges in vector c-graphs. For $\mathbf{v}_i \in I\!R^{d_i}$, $\mathbf{v}_j \in I\!R^{d_j}$, and $\mathbf{v}_k \in I\!R^{d_k}$ and with both $(i,j) \in CG$ and $(j,k) \in CG$

we get the local Jacobians $C_{ji} \in \mathbb{R}^{d_j \times d_i}$ and $C_{kj} \in \mathbb{R}^{d_k \times d_j}$. The forward [backward] elimination of all out-edges [in-edges] of a given intermediate vertex $v_j \in CG$ is equivalent to eliminating v_j itself. Since the edge labels are matrices the elimination of edges involves matrix products. Suppose we eliminate (j,k) forward. Then we get $C_{ki} += C_{kj} \cdot C_{ji}$ for all $i \in P_j$ using the earlier introduced C-style notation. As in scalar mode we connect all predecessors of v_j with v_k and perform the corresponding matrix multiplications to calculate the labels of the inserted edges. If an edge from v_i to v_k already exists then we additionally have to build the corresponding sum of the two matrices. Thus, we get the following number of multiplications involved in the forward elimination of an edge (j,k) in the general vector mode:

$$\mathbf{OPS_F}\{(j,k)\} = d_k d_j \sum_{i \in P_j} d_i.$$

Anticipating the beginning of Section 3, we will always count multiplications as a complexity measure, ignoring the number of additions involved in the calculation. Especially in this case where $d_k d_j \sum_{i \in P_j} d_i$ scalar multiplications are performed compared to at most $d_k \sum_{i \in P_j} d_i$ additions this approach seems to be appropriate as the latter are surely dominated by the former in terms of the time required to run them.

The backward elimination of (i,j) is performed correspondingly, i.e. $C_{ki} += C_{kj} \cdot C_{ji}$ for all $k \in S_j$. Again we connect v_i with all successors of v_j and perform the corresponding matrix multiplications possibly followed by additions. We get the following number of multiplications needed for the backward elimination of an edge (i,j):

$$\mathbf{OPS_B}\{(i,j)\} = d_i d_j \sum_{k \in S_j} d_k.$$

In consistency with Section 2.1 and Section 2.2 we call $\mathbf{OPS_F}\{(i,j)\}$ [$\mathbf{OPS_B}\{(i,j)\}$] the **forward [backward] Markowitz degree** of an edge (i,j) in general vector mode.

Vertex elimination: The elimination of a vertex v_j can be regarded as the forward elimination of all of its out-edges or, equivalently, as the backward elimination of its in-edges. In either case we get for the number of multiplications involved

$$\mathbf{OPS}\{v_j\} = d_j \left(\sum_{i \in P_j} d_i \right) \left(\sum_{k \in S_j} d_k \right)$$

which reduces to the **Markowitz degree** of v_j in the scalar mode if $d_j = 1$ for all j.

3 A Shortest Path Problem in the Metagraph

The Jacobian matrix of a vector function defines a linear transformation on real vectors. We will pick one particular representative J associated with a given vector function F and

think about a way to compute all its entries at a minimal cost. We have decided to regard the number of multiplications required to compute the complete Jacobian as the cost and we will denote this objective function by **Cost**$\{J\}$. Specifically, we will take the c-graph CG of F and we will search for an edge elimination sequence that minimizes **Cost**$\{J\}$. Apart from a large number of multiplications the calculation of the complete Jacobian involves a varying number of additions, too. Modern computers are able to perform additions and multiplications in the same time. So, we could think about not only minimizing the number of multiplications but also the number of additions or even the sum of both. We have the impression that the effort would not pay off since the number of additions is in general small compared to the number of multiplications. Therefore, we will concentrate on the problem of minimizing the number of multiplications by determining nearly optimal edge and vertex elimination orders. Undoubtedly, it will not be sufficient to think about operations counts while trying to make the computation of the complete Jacobian as fast as possible – and this is what we actually want. Memory accesses will play an important role as they usually dominate the overall execution time on the computer. However, this problem should be examined as part of a different project and therefore we will not go into detail with it. The final objective must be to merge the results from both projects in order to generate optimized derivative code as a compromise between the minimal number of arithmetic operations and an optimal memory access pattern.

Building on work of Rose and Tarjan [RoTa78] Herley showed in an unpublished paper that the computation of a vertex elimination sequence that minimizes the fill-in in the c-graph of a vector function is an NP-hard combinatorial optimization problem. We expect the same property to hold for the closely related problem of minimizing the number of multiplications required to accumulate the complete Jacobian matrix although there is no proof for it, so far.

3.1 Minimizing the operations count

The successive elimination of edges from the c-graph defines the so-called **metagraph** $M = M(CG) = (V_M, E_M)$ the vertices of which represent all different c-graphs that could possibly be derived from CG by edge elimination. Sometimes we will call the vertices $w_k \in V_M$ of the metagraph **stages**, thus indicating that they represent certain intermediate forms on our way from the original c-graph to a subgraph of the complete bipartite graph $K_{n,m}$ which represents the complete Jacobian. If there are $p \gg \max\{m,n\}$ intermediate vertices in CG, which can be connected by at most $\binom{p}{2}$ edges each of them either being in the graph or not, then an upper bound for the number of vertices in M is given by $2^{\binom{p}{2}}$. This value which is not very useful for practical situations can be improved if we take into account that certainly there are vertices in CG which are mutually unreachable. These vertices will never be connected by an edge regardless of the chosen edge elimination sequence. By denoting the number of edges in the transitive closure of CG by E^* we get $2^{|E^*|}$ as a tighter upper bound for the number of vertices in the metagraph. Labeling the edges in M with the cost of getting from the graph represented by their source to the one associated with their target

we end up with a shortest path problem on the metagraph. The edge labels are considered to be the distances. This **general edge elimination problem** could be solved using, for example, the algorithm by Bellman and Ford. Its run-time is proportional to $O(|V_M| \cdot |E_M|)$. The difficulties arise from the fact that both the number of vertices and the number of edges in the metagraph depend on the number of intermediate vertices in the original c-graph exponentially. Therefore, an exhaustive search as well as any algorithm for computing a shortest path in M are not practicable.

In Section 2.5 we have introduced the [forward, backward] Markowitz degree of [edges] vertices in CG. Notice that [$\mathbf{OPS_F}\{(i,j)\}$, $\mathbf{OPS_B}\{(i,j)\}$] $\mathbf{OPS}\{v_j\}$ is not static. Its value changes with the ongoing elimination of vertices and edges, which makes our objective to minimize $\mathbf{Cost}\{J\}$ a computationally hard combinatorial optimization problem. In order to be able to give an exact description of the shortest path problem which we intent to solve it is necessary to give precise definitions of what we mean by the [forward, backward] Markowitz degree [edges] vertices at a given stage in the metagraph.

Definition 3.1 *Let $w_k \in V_M$ be a stage in the metagraph $M = (V_M, E_M)$ representing the search space of the general edge elimination problem in a given c-graph $CG = (V, E)$. Let $CG_k = (V_k, E_k)$ be the c-graph associated with w_k.*

We call the number of multiplications required to eliminate an edge $(i,j) \in E_k$ forward [backward] the **forward [backward] Markowitz degree of** (i,j) **at the stage** $w_k \in V_M$ *and we denote it by* $\mathbf{OPS_F}^k\{(i,j)\}$ $\left[\mathbf{OPS_B}^k\{(i,j)\}\right]$. *Furthermore, we introduce*

$$\mathbf{OPS_{[F,B]}}^k\{(i,j)\} \equiv \min\left\{\mathbf{OPS_F}^k\{(i,j)\}, \mathbf{OPS_B}^k\{(i,j)\}\right\}.$$

For a vertex $v_j \in V_k$ we define its **Markowitz degree at the stage** $w_k \in V_M$ *as the number of multiplications involved in its elimination from CG_k and denote it by $\mathbf{OPS}^k\{v_j\}$. The minimal number of multiplications required to convert the original c-graph CG_0 into the graph CG_k by the [forward, backward] elimination of [edges] vertices will be referred to as the* **overall Markowitz degree** *at stage w_k.*

For the scalar mode on a scalar c-graph we get

$$\mathbf{OPS_F}^k\{(i,j)\} = |P_i|_k, \quad \mathbf{OPS_B}^k\{(i,j)\} = |S_j|_k, \quad \text{and} \quad \mathbf{OPS}^k\{v_j\} = |P_j|_k \cdot |S_j|_k$$

where $|P_j|_k$ [$|S_j|_k$] denotes the number of predecessors [successors] of v_i [v_j] in CG_k. Every stage w_k in the metagraph can be associated with a pair (m_k, ES_k) consisting of the overall Markowitz degree which is equal to the shortest path from the source w_0 of the metagraph to w_k and a sequence ES_k of labeled edges. The latter implies the order in which the edges were eliminated such that the number of multiplications involved in this process is equal to m_k, which is locally minimal. The labels indicate whether an edge $(i,j) \in E$ was eliminated forward or backward, i.e.

$$(i,j)_{[\mathbf{F},\mathbf{B}]} \equiv \begin{cases} (i,j)_{\mathbf{F}} & \text{for forward elimination of } (i,j) \\ (i,j)_{\mathbf{B}} & \text{for backward elimination of } (i,j) \end{cases}.$$

The initial stage $w_0 \in V_M$ represents the original c-graph with none of its intermediate edges eliminated. It is uniquely defined by $(m_0 = 0, ES_0 = ())$. An edge connects a stage $w_k \in V_M$ with another stage $w_l \in V_M$ if there is an edge $(i,j) \in CG_k$ the [forward, backward] elimination of which leads to CG_l, i.e.

$$(k,l) \in E_M \quad \Leftrightarrow \quad \Big[\exists (i,j) \in E_k : \;\; ES_l = (ES_k, (i,j)_{[\mathbf{F},\mathbf{B}]}),\; m_l = m_k + \mathbf{OPS}_{[\mathbf{F},\mathbf{B}]}{}^k\{(i,j)\}\Big].$$

With this notation we are now able to give the following formal description of the general edge elimination problem: *For a given c-graph $CG = (V.E)$ with the corresponding metagraph $M(CG) = (V_M, E_M)$ find an edge elimination sequence $ES_{|V_M|-1} = ((i_1,j_1),\ldots,(i_k,j_k))$ such that*

$$\mathbf{Cost}\{J\} = \sum_{l=1}^{k} m_{l-1} + \mathbf{OPS}^l_{[\mathbf{F},\mathbf{B}]}\{(i_l,j_l)\} \quad \rightarrow \quad \mathbf{MIN}.$$

In order to decrease the complexity of the problem to solve we have to make certain restrictions, thus reducing both size of the metagraph defined by the number of its stages and the number of different paths to check. This approach will lead to subgraphs which are then subject to an analogous shortest path problem. With every restriction ρ we can associate a corresponding minimal cost $\mathbf{Cost}_\rho\{J\}$ of computing the Jacobian by solving the shortest path problem on the induced subgraph $M_\rho \subseteq M$ of the metagraph, i.e. $\mathbf{Cost}_\rho\{J\} = c_\rho \cdot \mathbf{Cost}\{J\}$ for some $c_\rho \geq 1$. Obviously, we would like the factor c_ρ to be as small as possible as for large values the chosen restriction ρ would not be very useful. Some practicable methods which lead to significant reductions of the size of M are described in [Nau99]. Here we will concentrate on the three basic possibilities .

1. Forward edge elimination $(\rho = \mathbf{F})$

Suppose that we permit edges to be eliminated forward exclusively. Having only one elemental action for edge elimination this leads to a subgraph $M_\mathbf{F} \subseteq M$ the size of which has been reduced compared to M. For each stage $w_k \in M$ there are stages adjacent from w_k which can be reached only by the backward elimination of some edge in CG_k. For example, this is the case for those stages $w_s \in M$ where CG_s is derived from CG_k through the elimination of an edge $(i,j) \in E_k$ for which $v_i \in X$ is a minimal vertex, making the forward elimination of (i,j) impossible. The length of a shortest path in $M_\mathbf{F}$ will be denoted by

$$\mathbf{Cost}_\mathbf{F}\{J\} = \sum_{l=1}^{k_\mathbf{F}} m_{l-1} + \mathbf{OPS}^l_\mathbf{F}\{(i_l,j_l)\}$$

for a given optimal elimination sequence $ES_{|V_{M_\mathbf{F}}|-1} = ((i_1,j_1),\ldots,(i_{k_\mathbf{F}},j_{k_\mathbf{F}}))$. Obviously, we can construct special cases where the restriction to pure forward elimination does not lead to a reduction of the size of M. Since these situations are not interesting from the point of view of the optimization of cross-country elimination we ignore them.

In [Nau99] we have dealt with the representations of the edge elimination problem as a scalar chained matrix product and as a chained matrix product on the so-called *quotient graph*. Both approaches will lead to further reductions of the size of the metagraph leaving us with optimization problems which can be solved in a time that is polynomial in the number of intermediate vertices in the original c-graph. There are versions of the two problems that are based on products of local extended Jacobians such that the corresponding edge eliminations can be regarded as forward.

2. Backward edge elimination ($\rho = \mathbf{B}$)

Analogous statements as for the restriction to forward edge elimination can be made for the backward edge elimination case. The solution of the corresponding shortest path problem in $M_{\mathbf{B}}$ will be denote by $\mathbf{Cost_B}\{J\}$ and it is given as the solution of the combinatorial optimization problem

$$\sum_{l=1}^{k_{\mathbf{B}}} m_{l-1} + \mathbf{OPS}^{l}_{\mathbf{B}}\{(i_l, j_l)\} \quad \rightarrow \quad \mathbf{MIN}.$$

We will also consider the so-called *adjoint* versions of the corresponding chained matrix products. Thus, we get further subgraphs of the metagraph on which the shortest path problem can be solved in a time which is polynomial in the number of vertices in the c-graph. They have been regarded in a dynamic programming context in [Nau99].

3. Vertex elimination ($\rho = \mathbf{V}$)

The restriction to successively eliminating vertices from the c-graph has been exploited in several papers dealing with the minimization of the overall operations count for computing Jacobians ([GrRe91], [Bis96]). There are exactly $p!$ different elimination orders for a problem in p intermediate variables. Vertex elimination leads to the so-called **vertex metagraph** $M_{\mathbf{V}} \subseteq M$ an instance of which is shown in Figure 7. The minimal cost of computing the Jacobian using an optimal vertex elimination order is equal to the sum of all Markowitz degrees of the intermediate vertices v_j at the stages $w_{i(j)} \in M_{\mathbf{V}}$ of their elimination, i.e.

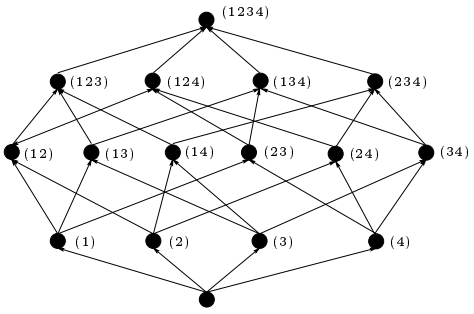


Figure 7: Vertex Metagraph $M_{\mathbf{V}}$

$$\mathbf{Cost_V}\{J\} = \sum_{1<j\leq p} \mathbf{OPS}^{i(j)}\{v_j\}. \tag{8}$$

The notation $i(j)$ is supposed to emphasize that the stage at which a particular vertex v_j is eliminated depends on the chosen method for determining the elimination order. A cost $\mathbf{OPS}^{i(j)}\{v_j\}$ corresponds to every vertex in CG depending on the stage $w_{i(j)} \in M_{\mathbf{V}}$ at which it is eliminated under the selected method. The problem of minimizing the cost defined by Equation (8) by determining an optimal vertex elimination order is called a **vertex elimination problem**. It is equivalent to the solution of a shortest path problem on the subgraph of the metagraph which is induced by the restriction to the elimination of vertices, i.e. the vertex metagraph. For a problem leading to a c-graph that contains p intermediate vertices there are $1+\sum_{i=0}^{p-1}\prod_{j=0}^{i}(p-j)$ vertices [stages] in the corresponding vertex metagraph. This makes a naive exhaustive search method impracticable. Some considerable improvement can be achieved by using a dynamic programming approach based on the interpretation of the vertex metagraph as a multistage graph as proposed in [Bis96]. We have to tabulate the solution to the 2^p subproblems corresponding to the vertices. Obviously, the complexity of this algorithm is still exponential in p.

3.2 Example

With the help of the very simple graph shown in Figure 10 in the box number 0 we will now have a closer look at the problem of minimizing the number of multiplications required for the accumulation of the Jacobian via the solution of a shortest path problem in the corresponding metagraph. Our example c-graph represents a scalar valued function $F : I\!R^2 \supseteq D \to I\!R$ like for instance $y = F(x_0, x_1) = x_0 \sin(x_1)\sin(x_1)$. But this is not as relevant as the structure of the c-graph. We have one dependent vertex v_3 depending on two independent vertices v_{-1} and v_0 and the complete c-graph consists of five vertices which are connected by five edges. Notice that there is just one intermediate edge that can be eliminated both forward and backward ($(1,2)$). Furthermore, we have two edges which are allowed to be eliminated forward only ($(2,3)$ and $(1,3)$) and the same number of edges which have to be eliminated backward ($(-1,2)$ and $(0,1)$).

M_ρ	V_{M_ρ}	symbol
$M_{\mathbf{F}}$	$\{0,\mathbf{1},\mathbf{2},7,\mathbf{12},3,13\}$	$\square$
$M_{\mathbf{B}}$	$\{0,3,\mathbf{4},\mathbf{5},\mathbf{9},\mathbf{11},7,13\}$	$\diamond$
$M_{\mathbf{V}}$	$\{0,3,7,13\}$	$\bullet$

Figure 8: M_ρ

The objective is to find a sequence of (forward and backward) edge eliminations which will give us the complete Jacobian as the complete bipartite graph $K_{2,1}$ at a minimal cost. We start by listing all intermediate forms of the c-graph that can be constructed using forward and/or backward edge elimination. For this small example we end up with the fourteen different c-graphs displayed in Figure 10 including both the original c-graph (0) and the $K_{2,1}$ (13), which represents the complete Jacobian (or in our case the gradient) of the underlying vector function F. These fourteen c-graphs are the vertices of the metagraph $M = (V_M, E_M)$ which is shown in Figure 9. Two vertices $w_i, w_j \in V_M$ are connected by an edge $(i,j) \in E_M$ if and only if it is possible to get from the intermediate c-graph CG_i represented by w_i to the one associated with w_j by either forward or backward elimination of

an edge in CG_i. We have indicated this fact by labeling the edges (i,j) in M with `[f]` [`[b]`] if we can construct CG_j from CG_i by forward [backward] elimination of an edge in CG_i only. Whenever there are both ways to get from CG_i to CG_j the edges are marked with `[f,b]`. In this case we could regard the corresponding edges as multiple consisting of two parallel edges – one for the forward elimination and the other representing the reverse elimination of an edge in CG_i. This will only be the case for the in- and out-edges of hoisting vertices. Here, both forward elimination of the out-edge as well as the backward elimination of the in-edge take exactly one multiplication. Having in mind that we are about to solve a shortest path problem on the metagraph with the edges representing distances corresponding to the costs of the associated edge eliminations we can consider all parallel edges in the metagraph as equivalent. Thus, we get that the metagraph is a directed, connected, and acyclic graph which does not contain any multiple edges.

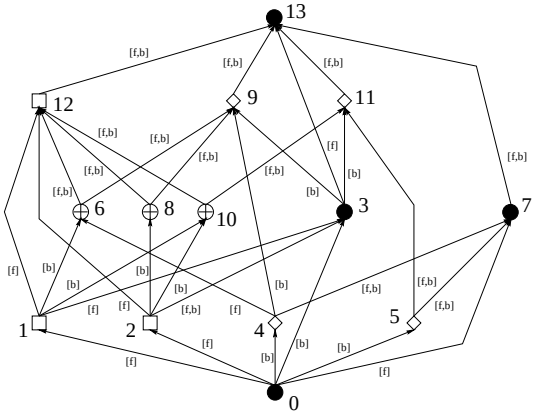


Figure 9: Metagraph

We will consider three strategies apart from the general edge elimination method: Forward edge elimination ($\rho = \mathbf{F}$), backward edge elimination ($\rho = \mathbf{B}$), and vertex elimination ($\rho = \mathbf{V}$). With every restriction ρ we can associate a certain subgraph $M_\rho \subseteq M$ on which we are going to solve the shortest path problem. Obviously, the source (0) and the sink (13) will be contained within any of the different subgraphs as we always start with the original c-graph and convert it into a subgraph of the complete bipartite graph. Figure 8 lists the vertices contained within the subgraphs M_ρ. The vertices corresponding to graphs which can be constructed by using one particular method exclusively are highlighted. In the metagraph we have visualized this fact by different ver-

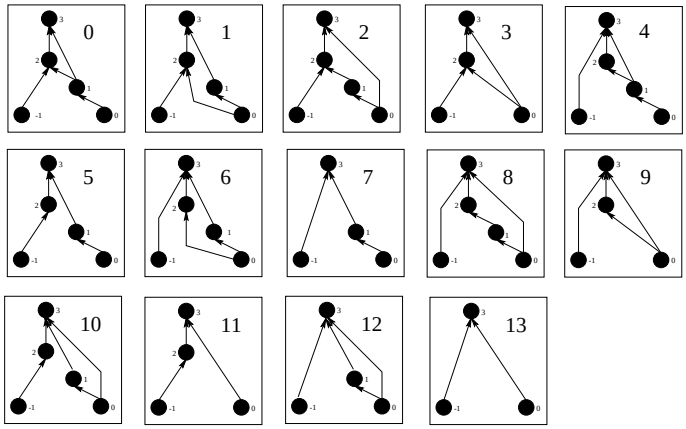


Figure 10: Metanodes

tex styles. Notice that three vertices (6,8,and 10) cannot be reached using any of the restricted methods and will therefore be contained only within the metagraph representing the general edge elimination strategy. From the numerical point of view the metagraph could contain even more vertices as there may occur situations where two intermediate forms of the c-graph are identical with respect to their structure while carrying different edge labels. Consider, for example, the graph associated with vertex 10 in the metagraph. There are two ways to transform CG_0 by eliminating edges such that we get to a graph with the structure of CG_{10}. We could forward eliminate $(1,3)$ ($\Rightarrow CG_2$) followed by the backward elimination of $(1,2)$. Alternatively, the forward elimination of $(1,2)$ takes us to CG_1 where we would have eliminate $(0,2)$ backward. Consider the local partial derivative labeling edge $(0,3)$ in CG_{10}. Choosing the way via CG_1 we have that $c^1_{30} = c_{32}c_{21}c_{10}$. In contrast to that the sequence which derives CG_{10} from CG_2 leads to $c^2_{30} = c_{31}c_{10}$. Assuming that $y = x_0 \sin(x_1)\sin(x_1)$ we would get for the further case $c^1_{30} = \sin(x_1)\cos(x_1)\sin(x_1)$ and for the latter $c^2_{30} = x_0 \sin(x_1)\cos(x_1)$. Naturally, the values of the remaining labels are such that in the end we will get the unique gradient.

4 Summary and Outlook

The goal of this paper was to present a formalism for describing the problem to calculate Jacobian matrices efficiently. Methods for solving this computationally hard problem will include elements from graph theory as well as from combinatorial optimization. The application of the chain rule to c-graphs is interpreted as the elimination of edges. By successively eliminating all intermediate edges we get to a stage where the c-graph represents a subgraph of a complete bipartite graph $K_{n,m}$, with a bipartition that corresponds to the n independent and the m dependent variables. The labels on the remaining edges are then exactly the nonzero entries of the complete Jacobian. Depending on whether the chain rule is applied as usual or to the adjoints, one distinguishes between two equal ways of eliminating edges which are referred to as forward and backward. The number of multiplications involved in the forward [backward] elimination of an edge is called its forward [backward] Markowitz degree. The sum of the forward [backward] Markowitz degrees of all edges at the time of their elimination from the c-graph (overall Markowitz degree) is considered to be the cost of calculating the complete Jacobian. In order to minimize this cost we have to solve a shortest path problem on the metagraph the vertices (stages) of which stand for all different graphs that can be constructed starting with the original c-graph by applying an arbitrary sequence of both forward and backward edge eliminations. Since the number of stages in the metagraph grows exponentially with the number of intermediate vertices in the original c-graph we have to put certain restrictions on the metagraph in order to reduce the size of the search space of the shortest path problem. This approach leads to different subgraphs of the metagraph which are then subject to the same shortest path problem, while having the number of distinct paths to be checked reduced.

The restriction to the elimination of vertices is one practicable way to reduce the size of the metagraph. However, the number of stages in the resulting subgraph still grows

exponentially with the number of intermediate vertices in the original c-graph. Furthermore, we have shown that the optimal vertex elimination sequence does in general not minimize the number of multiplications involved in the computation of the Jacobian. We conjecture that the vertex-edge discrepancy is less or equal to 2.

In [Nau99] we have presented numerous heuristics for eliminating both edges and vertices from c-graphs. Furthermore, dynamic programming algorithms are presented for optimizing the chained local extended [adjoint] Jacobians product the complexity of which is polynomial in the number of intermediate vertices in the c-graph. Simulated annealing methods turned out to be a very robust but expensive way to approach the vertex elimination problem. Most of our theoretical results were implemented and tested on numerous example problems. Reductions of the operations count by an average factor of 3-5 could be achieved. In order to exploit these results we are going to work on the automatic generation of optimized derivative code.

References

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

[BKBC96] C. Bischof, P. Khademi, A. Bouaricha, and A. Carle, *Computation of gradients and Jacobians by transparent exploitation of sparsity in automatic differentiation*, Tech. Report 1, Argonne National Laboratory, 1996.

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

[Chri93] B. Christianson, *Reverse accumulation of functions containing gradients*, Tech. Report 278, NOC, University of Hertfordshire, Hatfield, UK, 1993.

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

[GUW97] A. Griewank, J. Utke, and A. Walter, *Evaluating higher derivative tensors through univariate Taylor polynomials*, Preprint, IOKOMO-09-1997, TUD, 1997.

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

[Iri91] M. Iri, *History of automatic differentiation and rounding error estimation* in [CoGr91], pp. 3-16.

[Kub91] K. Kubota and M. Iri, *Estimates of rounding errors with fast automatic differentiation and interval analysis*, Journal of Information Processing, 14 (1991), pp. 508-515.

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

[RaCo96] L. Rall and G. Corliss, *An introduction to automatic differentiation*, in [BBCG96], pp. 1-18.

[RoTa78] D. J. Rose and R. E. Tarjan, *Algorithmic aspects of vertex elimination on directed graphs.* SIAM Journal of Applied Mathematics, Vol. 34, no. 1, January 1978, 176-197.

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

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399