Heuristics for Efficiently Calculating Jacobians by Edge Elimination in Computational Graphs

Uwe Naumann

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Heuristics for Efficiently Calculating Jacobians by Edge Elimination in Computational Graphs

Uwe Naumann

N° 3690

Mai 1999

THÈME 2

ISSN 0249-6399

INRIA
SOPHIA ANTIPOLIS

Heuristics for Efficiently Calculating Jacobians by Edge Elimination in Computational Graphs

Uwe Naumann *

Thème 2 — Génie logiciel
et calcul symbolique
Projet TROPICS

Rapport de recherche n° 3690 — Mai 1999 — 41 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 looked at this interesting subject so far and there is no generally accepted terminology for dealing with 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.

After an extensive introduction into the problem of computing Jacobians using a minimal number of arithmetic operations we will describe several heuristics for reducing the size the computational graph as well as for solving the edge elimination problem in computational graphs. We will discuss the ideas behind the heuristics and present some test results. Finally we will give an outlook on the role that edge elimination could play in the development of new AD tools.

Key-words: Computational graph, edge elimination, local heuristics

* 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

Heuristiques pour le Calcul Efficace des Jacobiennes par 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. Pour cela il faut résoudre un problème d'optimisation combinatoire. Des heuristiques diverses offrent la possibilité de résoudre ce problème.

Mots-clés : Graphe de calcul, élimination des arcs, heuristiques locales.

Contents

1 Introduction

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 given as computer programs numerically up to machine precision. Notice, that AD is completely different from the approach to computing derivatives numerically through divided differences.

1.1 Edge Elimination in Computational Graphs

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 in 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 any arbitrary order, which leads to 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.

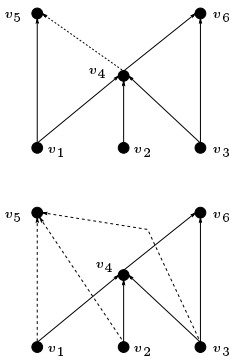

Figure 1: $(4,5)$

In our approach we expect the vector function F to be such that it can be decomposed into a sequence of 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` 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 $(\mathbb{R} \ni) v_j = \varphi_j(v_i)_{i \in P_j}$ by assigning the result of every elemental function φ_j that occurs in the program to a unique intermediate variable v_j. Here P_j is the index set of the arguments of φ_j and we denote its cardinality by $|P_j|$. 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), the elemental functions φ_j $(j = 1, 2, \ldots, q+m)$ are well defined for some fixed argument 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 $n_j \equiv |P_j|$.

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:

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

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. As above we set P_j $[S_j]$ to be the set of indices of all predecessors [successors] of a vertex v_j. According to Assumption 1.1 labels c_{ji}, representing the local partial derivatives, are attached to all edges $(i, j) \in E$.

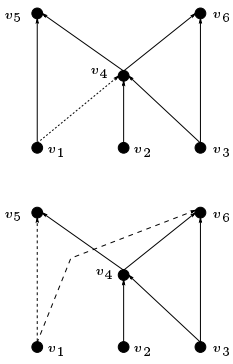

Figure 2: $(1, 4)$

We are looking for a method of transforming the 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, where 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, ... ,n-1, j=0, ... ,m-1) on the edges connecting the minimal vertices with the maximal ones, are exactly the non-zero 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 in form of c-graphs. Consequently we distinguish between two ways to eliminate an edge in CG and we will refer to them as **forward** and **backward** edge elimination.

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). This is illustrated in Figure 1 with the help of edge $(4, 5)$. 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. Thus, 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 $|P_{(i,j)}| = |P_i|$.

The graphic interpretation of the backward elimination of $(1,4)$ is shown in Figure 2. 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. 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 $|S_j|$ of successors of the target v_j.

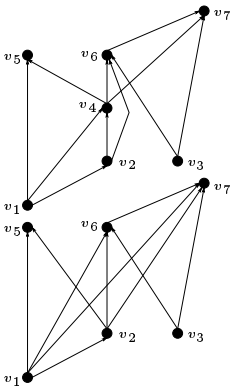

Figure 3: v_4

The process of eliminating edges will always terminate and deliver the Jacobian in form of the bipartite c-graph since the sum of the total lengths of all distinct paths in the c-graph is strictly monotonically decreasing during it (see Section 3.4). Whenever the elimination of an edge (i,j) leads to the insertion of a new edge which has not been in the c-graph before we say that fill-in was generated. More general, it is useful to define the **fill-in** as the difference between the number of edges in the c-graph after and before the elimination of (i,j). If an edge which would have to be inserted already does exist then it is said to be **absorbed**.

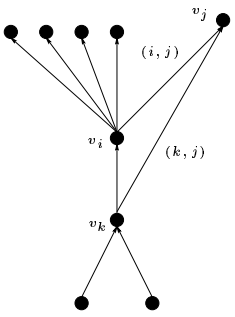

Figure 4: Surprise

The forward [backward] elimination of all out-edges [in-edges] of a vertex v_i leads to the elimination of v_i itself (Figure 3). Consequently, the elimination of an intermediate vertex v_i involves $|P_i| \cdot |S_i|$ scalar multiplications. This value is usually referred to as the **Markowitz degree** of v_i [GrRe91]. In the available literature 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]). However, 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 4. 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 $((v_i,v_k) \hat{=} 5+10 =)15$ and $((v_k,v_i) \hat{=} 4+10 =)14$ 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 13 multiplications which is obviously less than $14 = \min\{(v_i,v_k),(v_k,v_i)\}$ in the pure vertex elimination mode. Furthermore, we have shown that the **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 c-graphs with $p > 2$ intermediate

vertices. However, building on numerous test results we conjecture that the vertex-edge discrepancy will be less or equal to 2 for the general case.

1.2 Extended Jacobians

We define the **extended local Jacobians** C_i for $i = 1, \ldots, p+m$ as

$$C_i \equiv \begin{pmatrix} 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 \\ c_{i(1-n)} & \cdots & c_{i(i-n)} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{pmatrix} \in I\!R^{|V| \times |V|}$$

where the c_{ij} are the elementary partial derivatives of $v_i = \varphi_i(\{v_j : j \in P_i\})$ with respect to its predecessors occurring in the $(n+i)$-th row of C_i. Thus, we have that $c_{ij} = 0$ if $j \notin P_i$. We define the extended local Jacobian of a set of vertices with indices in $N \subseteq Z \cup Y$ as

$$C_N \equiv \sum_{i \in N} C_i.$$

Analogous, we define the **adjoint extended local Jacobians** $\bar{C}_i$ for each of the intermediate or minimal vertices $v_i \in X \cup Z$, i.e. we set for $i = 1-n, \ldots, p$

$$\bar{C}_i \equiv \begin{pmatrix} 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & c_{(i+1)i} & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & c_{(p+m)i} & 0 & \cdots & 0 \end{pmatrix} \in I\!R^{|V| \times |V|}$$

where the c_{ji} are the local partial derivatives of the successors of v_i with respect to v_i itself occurring in the $(n+i)$-th column of $\bar{C}_i$. Again, the extended adjoint local Jacobian of a vertex set $\{v_i : i \in N\}$, with $N \subseteq X \cup Z$, is defined as

$$\bar{C}_N \equiv \sum_{i \in N} \bar{C}_i.$$

In order to simplify the notation used for the further argumentation it is useful to define projections of vectors and matrices to certain components, rows or columns:

$$I\!R^{|V| \times |V|} \ni \quad I(M) \equiv \sum_{i \in M} \mathbf{e}_i \cdot \mathbf{e}_i^T$$

for a given index set $M \subseteq \{0, \ldots, |V|-1\}$ and with $\mathbf{e}_i \in I\!R^{|V|}$ denoting the i-th Cartesian basis vector. Then we have that $I(M) \cdot A$ maps A to the rows indices of which are elements of M and, analogous, $A \cdot I(M)$ extracts the corresponding columns from A. For a given vector $\mathbf{a} \in I\!R^{|V|}$ $I(M) \cdot \mathbf{a}$ sets all those components of $\mathbf{a}$ indices of which are not in M to zero.

The global extended Jacobian of a single assignment code evaluated at a given argument point is defined as $I\!R^{|V|\times|V|} \ni J_e = (c_{ji})$, where the c_{ji} are the local elementary partial derivatives labeling the edges which connect vertices v_i and v_j in the c-graph. Obviously, J_e is a square lower triangular matrix. For practical reasons we also define the diagonal elements to be zero, which gives us

$$J_e = \sum_{i=1}^{p+m} C_i = \sum_{j=1-n}^{p} \bar{C}_j$$

as the local extended Jacobian of the vertex set $\{v_i : i \in Z \cup Y\}$ or as the local extended adjoint Jacobian of $\{v_j : j \in X \cup Z\}$.

The process of calculating the complete Jacobian by a combined vertex-edge elimination procedure may be regarded as a transformation process of the extended Jacobian. There is an equivalent for every action in the c-graph in terms of a transformation of J_e:

- Forward elimination of an edge (i, j):

$$\begin{aligned} J_e :&= J_e + \mathbf{e}_j \mathbf{e}_j^T J_e \mathbf{e}_i \mathbf{e}_i^T (J_e - I) \\ &= J_e + \mathbf{e}_j \mathbf{e}_j^T \bar{C}_i C_i - \mathbf{e}_j \mathbf{e}_j^T J_e \mathbf{e}_i \mathbf{e}_i^T. \end{aligned}$$

- Backward elimination of an edge (i, j):

$$\begin{aligned} J_e :&= J_e + (J_e - I) \mathbf{e}_j \mathbf{e}_j^T J_e \mathbf{e}_i \mathbf{e}_i^T \\ &= J_e + \bar{C}_j C_j \mathbf{e}_i \mathbf{e}_i^T - \mathbf{e}_j \mathbf{e}_j^T J_e \mathbf{e}_i \mathbf{e}_i^T. \end{aligned}$$

- Elimination of a vertex v_i:

$$\begin{aligned} J_e :&= (I - \mathbf{e}_i \mathbf{e}_i^T)(J_e + \bar{C}_i C_i)(I - \mathbf{e}_i \mathbf{e}_i^T) \\ &= (I - \mathbf{e}_i \mathbf{e}_i^T) J_e (I + \mathbf{e}_i \mathbf{e}_i^T J_e)(I - \mathbf{e}_i \mathbf{e}_i^T) \\ &= (I - \mathbf{e}_i \mathbf{e}_i^T)(I + J_e \mathbf{e}_i \mathbf{e}_i^T) J_e (I - \mathbf{e}_i \mathbf{e}_i^T). \end{aligned}$$

1.3 Example

After having discussed how to calculate the entries of the complete Jacobian of a given evaluation routine of a vector function F by converting it into a directed, acyclic c-graph

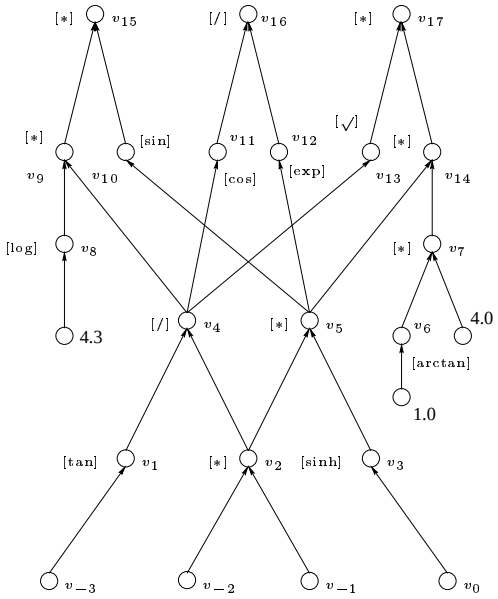

Figure 5: Complete Computational Graph

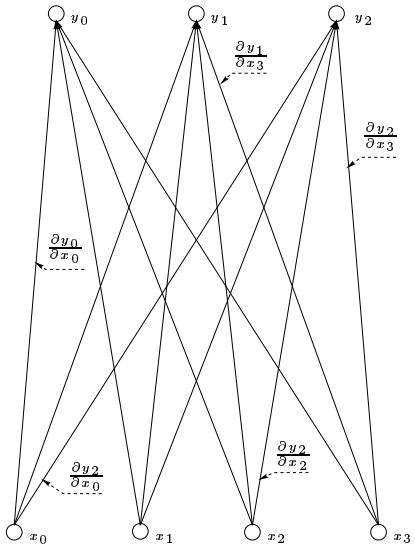

Figure 6: $K_{4,3}$

followed by the elimination of all intermediate edges [vertices] it is now time for an example to illustrate our results.

We will consider the following fragment of a C-program evaluating a vector function

$$F : I\!R^4 \supseteq D \to I\!R^3 :$$

```
double x[4], y[3];
double h1,h2,pi;
...
h1 = tan(x[0])/(x[1]*x[2]);
h2 = x[1]*x[2]*sinh(x[3]);
pi = 4.*arctan(1.);
y[0] = log(4.3)*h1*sin(h2);
y[1] = cos(h1)/exp(h2);
y[2] = sqrt(h1)*pi*h2;
...
```

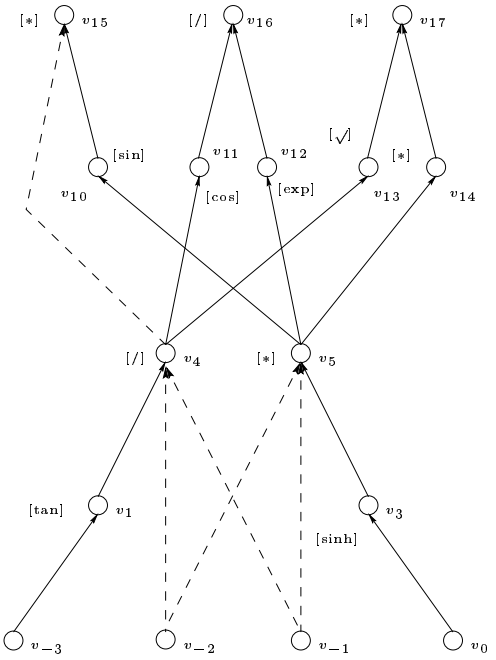

Figure 7: Transformed Computational Graph

It contains the vector $\mathbf{x}$ of the independent variables, the vector $\mathbf{y}$ the components of which represent the dependent variables, and three auxiliary variables h1, h2, and pi. Following the procedure described in Section 1.1 we can break this program down into a sequences of scalar assignment by storing the result of every elemental function in a unique intermediate variable. Thus, we get the following evaluation trace associated with the above evaluation routine:

```
v[1]=tan(v[-3]); v[2]=v[-2]*v[-1]; v[3]=sinh(v[0]); v[4]=v[1]/v[2];
v[5]=v[2]*v[3]; v[6]=arctan(1.); v[7]=4.*v[6]; v[8]=log(4.3);
v[9]=v[8]*v[4]; v[10]=sin(v[5]); v[11]=cos(v[4]); v[12]=exp(v[5]);
v[13]=sqrt(v[4]); v[14]=v[5]*v[7]; v[15]=v[9]*v[10]; v[16]=v[11]/v[12];
v[17]=v[13]*v[14];
```

Building on the single assignment code the complete c-graph CG_F^c displayed in Figure 5 can be derived. For a given argument vector $\mathbf{x}$ we are now able to calculate the function value $\mathbf{y}$ and the local partial derivatives c_{ji} labeling the edges in CG_F^c by performing the initial forward sweep. After the removal of all dead vertices, i.e. those intermediate vertices which either are not reachable from any of the minimal vertices or from which none of the maximal

vertices can be reached, we get the reduced version of the c-graph with the corresponding extended Jacobian matrix J_e. This is when the interesting part begins. By successively eliminating all intermediate vertices [edges] from the c-graph we will finally reach our goal in the shape of the complete bipartite c-graph $K_{4,3}$ shown in Figure 6, which represents the complete Jacobian.

In order to illustrate this process let us have a look at the elimination of v_2 followed by the backward elimination of $(4,9)$.

$$J_e := (I - \mathbf{e}_2\mathbf{e}_2^T)(J_e + \bar{C}_2 C_2)(I - \mathbf{e}_2\mathbf{e}_2^T)$$

describes the elimination of v_2 in terms of a transformation of J_e. Taking this new extended Jacobian and performing the equivalent to the backward elimination of $(4,9)$ results in

$$J_e := J_e + (J_e - I)\mathbf{e}_9\mathbf{e}_9^T J_e \mathbf{e}_4\mathbf{e}_4^T.$$

Here, the backward elimination of $(4,9)$ is equivalent to the elimination of v_9. The c-graph after these actions is shown in Figure 7, with the new edges highlighted. As illustrated by this example the application of the chain rule to functions, algorithms, or c-graphs is not restricted to the approach exploited in the forward or reverse modes of AD. In Section 1.5 we will deal with arbitrary application orders and we will describe the combinatorial optimization problem arising from this setup.

1.4 Classification

So far, we have assumed the derivative objects associated with the vertices in the c-graph to be scalars. 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 1.1 *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 1.2 *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 8. 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 is usually referred to as the vector mode of AD.

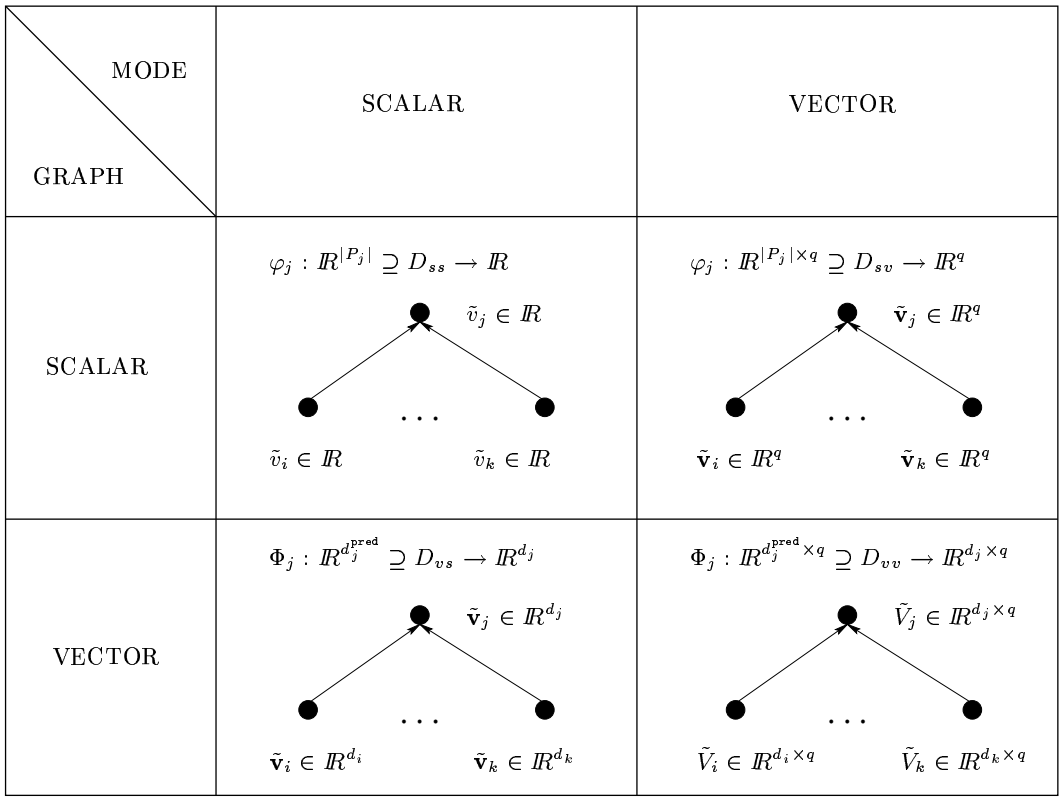

Figure 8: Classification

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 $\mathbb{R}$ 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. Here we 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 make sense even 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

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

and the local partial derivatives become local Jacobian matrices

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

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):

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

General reverse vector mode (incremental form):

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

Expressions for the two remaining, but for reasons described in [Nau99b] 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 $\mathbf{v}_i \in \mathbb{R}^{d_i}$, $\mathbf{v}_j \in \mathbb{R}^{d_j}$, and $\mathbf{v}_k \in \mathbb{R}^{d_k}$ and with both (i,j) and (j,k) in CG we get 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 $C_{ki} \mathrel{+}= C_{kj} \cdot C_{ji}$ 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:

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

As stated in Section 1, 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 1 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.

1.5 Shortest Path Problem

Given an evaluation procedure of a vector function F we are going to think about a way to compute its Jacobian at a minimal cost. For reasons described in [Nau99b] 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 $\mathbf{Cost}\{J\}$. Specifically, we will take the c-graph CG of F and we will search for an edge elimination sequence that minimizes $\mathbf{Cost}\{J\}$.

The successive elimination of edges from the c-graph defines the so-called **metagraph**

$$M = M(CG) = (V_M, E_M)$$

the vertices $w_k \in M$ of which represent all different c-graphs that could possibly be derived from CG by edge elimination. 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 and we will refer to this problem as the **general edge elimination problem**. The difficulties arise from the fact that both the number of vertices and the number of edges in the metagraph grow exponentially with the number of intermediate vertices in the original c-graph. Therefore, an exhaustive search as well as any algorithm for computing a shortest path in M are not practicable. 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 to be as small as possible as for large values c_ρ the chosen restriction ρ would not be very useful. One possibility is the restriction to eliminating vertices in the c-graph which will be exploited in this paper. This approach leads to the so-called **vertex metagraph** $M_\mathbf{V} \subseteq M$ an instance of which is shown in Figure 9 for a c-graph containing 4 intermediate vertices. For further information on the general edge elimination problem refer to [Nau99b].

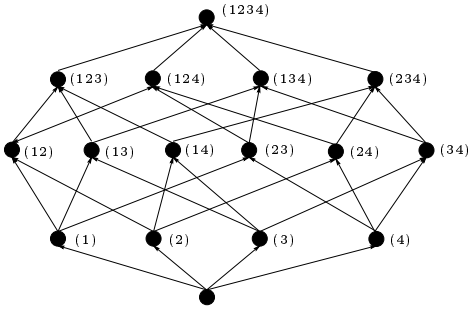

Figure 9: Vertex Metagraph $M_\mathbf{V}$

We have developed new and adapted some well-known algorithms for solving the combinatorial optimization problem. These methods contain local heuristics as well as a collection of simulated annealing schedules and dynamic programming algorithms for the computation of nearly optimal vertex elimination sequences. The former will be looked at more closely in this paper. All the approaches proposed are implemented as parts of a program named **OESCOMP** [Nau99a], which all our test results will be based on.

Considering local heuristics for edge or vertex elimination in c-graphs there are, obviously, two trivial cases: Let $v_i < v_j$ denote the situation where a vertex v_i is eliminated before v_j. The elimination order yielding $\forall w_k \in M : v_i < v_j \Leftrightarrow i < j$ will be referred to as the **forward vertex elimination mode (V_F)**. Analogous, we define the **backward vertex elimination mode (V_B)**. Both **V_F** and **V_B** are considered to be equivalent to the corresponding edge elimination modes. Other approaches to solving the general edge elimination problems are built on the ideas of dynamic programming [Nau99a] and of simulated annealing [Nau99c], respectively.

Edge elimination in c-graphs makes full use of the structural sparsity of the given problem. It is often possible to reduce the number of multiplications required to accumulate

the complete Jacobian by a factor of three and more compared to the method proposed by Newsam and Ramsdell [NeRa83]. The savings compared to the dense forward and reverse modes are even more significant. When presenting test results in Section 6 we will consider the achieved values relative to lower bounds for values of the best choice out of dense forward and reverse modes (**DM**), i.e.

$$\mathbf{Cost_{DM}}\{J\} = \min\{n(m+p), m(n+p)\},$$

and the corresponding minimum operations count achieved by a uni-directional approach of the method by Newsam and Ramsdell (**NR**) which is

$$\mathbf{Cost_{NR}}\{J\} = \min\{\hat{n}(m+p), \hat{m}(n+p)\}.$$

$\hat{n}$ and $\hat{m}$ denote the maximal number of non-zero elements per row and column of the Jacobian, respectively.

2 A Priori Reductions

The effort that has to be put into finding a method for computing the Jacobian as efficiently as possible by the solution of a shortest path problem in the corresponding metagraph grows exponentially with the number of intermediate edges [vertices]. Therefore we must be interested in maximal a priori reductions of this number without loosing the chance to find an optimal edge [vertex] elimination sequence. In this section we will discuss some of these *a priori reduction rules*. For practical reasons we will present some heuristic extensions which deliver good results in many cases and, most importantly, can be implemented efficiently. Tests will support our approach by illustrating the often remarkable decrease in the number of intermediate edges [vertices] that can be achieved.

All we need for solving the general edge elimination problem are the edges that carry non-zero elementary partial derivatives c_{ji} as labels in the c-graph. The calculation of the c_{ji} can be performed in parallel with the computation of the function value at the current argument point during one initial forward sweep.

Definition 2.1 *A restriction ρ on the c-graph which leads to the extraction of a subgraph M_ρ from the metagraph M is called* **optimality preserving** *if its computational effort is* $\mathbf{OPS}\{\rho\}$ *and*

$$\mathbf{OPS}\{\rho\} + \mathbf{Cost}_\rho\{J\} = \mathbf{Cost}\{J\}.$$

The **computational effort** *of a restriction ρ is defined as the number of multiplications to be performed in order to extract M_ρ.*

Notice that ρ is not necessarily a restriction in the sense of Section 1.5. The deletion of a passive vertex (passive in the sense that its value does not impact the Jacobian) as well as the removal of an assignment from the c-graph could be regarded as optimality preserving as their computational efforts are zero and $\mathbf{Cost}_\rho\{J\} = \mathbf{Cost}\{J\}$. On the other hand

the restriction to pure vertex elimination ($\rho = \mathbf{V}$) is certainly not optimality preserving in general although $\mathbf{OPS}\{\mathbf{V}\} = 0$ but, as shown in Section 1.1, $\mathbf{Cost_V}\{J\} \geq \mathbf{Cost}\{J\}$. For a restriction to be optimality preserving is a very strong demand. In fact, we will present only one restriction on the metagraph which satisfies all properties of being optimality preserving based on the a priori elimination of all vertices having Markowitz degree one.

It is useful to define the term *optimality preserving* with respect to different subgraphs of the metagraph. A certain restriction could be optimality preserving with respect to a vertex elimination method while not satisfying the required conditions for the general edge elimination strategy. A restriction ρ corresponding to the extraction of a subgraph M_ρ from the metagraph M is considered to lead to a reduction of the complexity of the general edge elimination problem through reasonable losses in the minimal overall Markowitz degree that can possibly be achieved if

$$\mathbf{Cost}_\rho\{J\} = \nu \cdot \mathbf{Cost}\{J\} \qquad \text{with} \quad \nu \in I\!R, \quad \nu \text{ small}.$$

So, for example, the restriction to a pure vertex elimination strategy could be regarded as reasonable in the above sense provided that our conjecture in Section 1.1 concerning the maximal *vertex-edge discrepancy* is right.

The minimal in-degree [out-degree] of any intermediate vertex in the c-graph is one with respect to the general edge elimination strategy. It is always possible to eliminate all in-edges of an intermediate vertex backward such that only one will be left. Analogous, this works with forward elimination for the minimal out-degree, too. If we restrict ourselves to forward [backward] elimination of edges only then, according to a result by Menger [Har69], the minimal in-degree [out-degree] is equal to the number of disjoint paths connecting the minimal vertices with its source [its target with the dependent variables]. For a pure vertex elimination method we have this result for both the minimal in-degree and the out-degree of any intermediate vertex.

We observe that the forward [backward] elimination of an edge (i, j) is optimality preserving if its source [target] is a hoisting vertex, i.e. if it has exactly one predecessor and one successor. Let therefore α denote the forward elimination of (i, j), which is equivalent to the elimination of its source v_i. We have that $\mathbf{OPS}\{\alpha\} = 1$ since v_i is a hoisting vertex. Obviously, there is no way to eliminate an intermediate edge [vertex] at a lower cost. Suppose that keeping (i, j) will result in the possibility to find a shorter path in M, i.e.

$$\mathbf{OPS}\{\alpha\} + \mathbf{Cost}_\alpha\{J\} > \mathbf{Cost}\{J\}.$$

The in-degree [out-degree] of vertices in the c-graph can be decreased if and only if any of their in-edges [out-edges] are involved in the absorption of other edges. In our situation this would imply that there is at least one other path connecting v_i with v_j apart from (i, j). This would lead to a contradiction to v_i being a hoisting vertex. The forward elimination of (i, j) is equivalent to the backward elimination of the only in-edge (k, i) of v_i which is optimality preserving by the similar argument.

2.1 Heuristics for a priori reductions

Heuristics turned out to deliver very promising results and will be discussed now. All of them have been implemented as part of our software package **OESCOMP** [Nau99a].

2.1.1 Forward [backward] invariant edges

Definition 2.2 *We call an edge (i,j)* **forward [backward] invariant** *if it can be eliminated at a cost of one multiplication and one addition, i.e. if the single predecessor v_p [successor v_s] of v_i [v_j] is connected with v_j [v_i] by an edge (p,j) [(i,s)].*

To make sure that the forward [backward] elimination of an invariant edge (i,j) is optimality preserving we would have to check whether the out-degree of v_i [in-degree of v_j] is minimal. For the general edge elimination approach this would again lead to the necessity for v_j [v_i] to be a hoisting vertex. However, numerous test results support the usefulness of this heuristic approach as it usually leads to a significant decrease of the number of edges in the c-graph. Moreover, it is a fast and easy to implement way for decreasing the complexity of the combinatorial optimization problem leading in most cases to only reasonable losses in the overall cost.

The forward [backward] elimination of forward [backward] invariant edges may lead to its source [target] becoming a hoisting vertex. Therefore, this heuristic should be run in parallel with the elimination of all hoisting vertices resulting in an even larger reduction of the size of the c-graph.

We have extended the above idea to what we call *forward [backward] fill-in invariant edges*. In this case we do not require the cost of eliminating an edge to be one multiplication. However, every multiplication must be followed by an addition, thus, making optimal use of absorption as all newly generated edges are immediately absorbed by an already existing edge.

Finally, we have implemented an even weaker heuristic based on the exploitation of absorption. Whenever an edge (i,j) is part of a "triangle" (i.e. if there exists a direct predecessor v_p of v_i which is connected with v_j by an edge (p,j) [if there exists a direct successor v_s of v_j such that there is an edge (i,s)]) we eliminate it forward [backward].

2.1.2 Additive parts

Consider a function of the form $y = f(x,q)$ with $f : I\!R^n \to I\!R$. In particular

$$f_b(x,q) = y = \psi\left(\sum_{j=0}^{q-1} \varphi_j\left(\frac{\sum_{i=0}^{n-1} x_i}{j+1}\right)\right)$$

which we will refer to as the so-called *belt function* indicating that its behavior with respect to the cost of the vertex elimination approach can be improved significantly by "tightening its belt". This will become clear during the following discussion. Naively, the fact that we

have additive parts in the c-graph could be a reason for eliminating them a priori because we could avoid the trivial multiplications by [minus] one which, obviously, are the corresponding partial derivatives labeling the edges that lead into the vertices representing the addition [subtraction]. The original c-graph of the problem is shown in Figure 10 on the left. We will look at two different reduced c-graphs (CG in the middle of Figure 10 and $\hat{CG}$ on the right) obtained by performing certain a priori reductions. Suppose that the vertices in $\hat{CG}$ are labeled as follows:

$$v_{i-n+1} = x_i, \quad i = 0, \ldots, n-1, \quad v_1 = \sum_{i=0}^{n-1} x_i,$$

$$v_j = \varphi_{j-2}(v_1), \quad j = 2, \ldots, q+1, \quad v_{q+2} = \sum_{i=2}^{q+1} v_i.$$

Let CG be labeled in a similar way, i.e. just with v_1 is missing. As usual, the $c_{ji} = \partial v_j / \partial v_i$ denote the local sensitivity of v_j with respect to v_i for $j = 1, \ldots, q+2$ and $i = 1-n, \ldots, q+1$. Applying the backward vertex elimination mode (**V_B**) to both CG and $\hat{CG}$ we observe a very interesting behavior of the cost of accumulating the gradient depending on the parameters n and q. We denote the cost of running **V_B** on CG [$\hat{CG}$] by $\mathbf{OPS}_{\mathbf{V\_B}}(CG)$ [$\mathbf{OPS}_{\mathbf{V\_B}}(\hat{CG})$].

Case 1 (CG):

$$\frac{\partial y}{\partial x_{i+n-1}} = \frac{\partial v_{q+2}}{\partial v_i} = \sum_{j=1}^{q} c_{ji} \cdot c_{q+2,j} \qquad \text{for } i = 0, \ldots, n-1.$$

The cost of running this elimination sequence is

$$\mathbf{OPS}_{\mathbf{V\_B}}(CG) = q(n+1).$$

Case 2 ($\hat{CG}$):

$$c_{q+2,1} = \sum_{j=2}^{q+1} c_{q+2,j} \cdot c_{j,1},$$

$$\frac{\partial y}{\partial x_{i+n-1}} = \frac{\partial v_{q+2}}{\partial v_i} = 1.0 \cdot c_{q+2,1} \qquad \text{for } i = 0, \ldots, n-1.$$

In this case the multiplications with 1.0 are actually performed and we end up with a cost of

$$\mathbf{OPS}_{\mathbf{V\_B}}(\hat{CG}) = 2q + n.$$

We have assumed that the calculation of the c_{ji} involves the multiplication with the constant factor $1/(j+1)$ which we have also omitted in the representation of the c-graph in Figure 10. Obviously,

$$\lim_{q\to\infty}\left\{\frac{\mathbf{OPS_{V\_B}}(CG)}{\mathbf{OPS_{V\_B}}(\hat{CG})}\right\} = \frac{1}{2}(n+1),$$

$$\text{and} \qquad \lim_{n\to\infty}\left\{\frac{\mathbf{OPS_{V\_B}}(CG)}{\mathbf{OPS_{V\_B}}(\hat{CG})}\right\} = q$$

which implies that depending on q and n **V_B** on CG can become arbitrarily worse than the same vertex elimination strategy applied to $\hat{CG}$. The reason for this is the repeated recalculation of *common subexpressions* as a result of the a priori removal of all additive parts from the c-graph. In terms of accumulating Jacobians by edge elimination we can associate this effect with the generation of fill-in. Based on this observation we have implemented a heuristic version of the handling of additive parts which deletes a vertex representing an addition or subtraction only if the generated fill-in is reasonable low. Specifically, given a vertex v_i which represents an additive operation we check whether at the current stage w_k in the metagraph we have

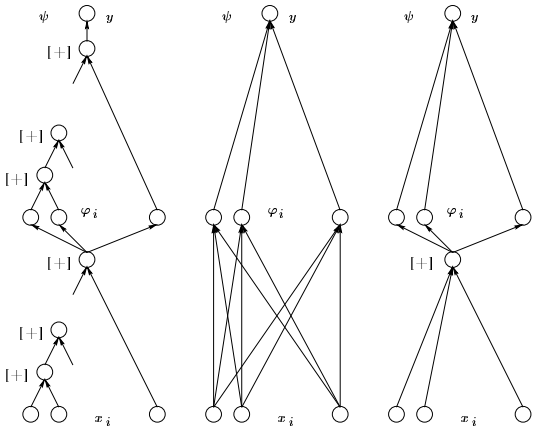

Figure 10: Belt Function

$$|P_i|_k \cdot |S_i|_k \leq \nu(|P_i|_k + |S_i|_k)$$

where $\nu \in I\!R$ is a small constant which we have set to two in the current implementation. Our test results show that this approach is very effective.

2.1.3 Sub-minimal and sub-maximal vertices

We denote the minimal [maximal] in-degree of a vertex v_i with respect to all stages in the metagraph by $\underline{|P_i|}$ [$\overline{|P_i|}$] and we use similar notations for the out-degree.

Definition 2.3 *Let $CG = (V, E)$ be a c-graph.*

1. *A vertex $v_i \in V$ is called* **sub-minimal** *if $\underline{|P_i|} = \overline{|P_i|}$.*
2. *A vertex $v_i \in V$ is called* **sub-maximal** *if $\underline{|S_i|} = \overline{|S_i|}$.*

In the general metagraph M a vertex can only be sub-minimal [sub-maximal] if it is a hoisting vertex. This is no longer true for certain restrictions on the metagraph leading to subgraphs like for instance $M_{\mathbf{V}}$.

Building on the concept of sub-minimality [sub-maximality] we have implemented more heuristics for eliminating vertices in the c-graph a priori. We have concentrated on regarding vertices that have only one minimal [maximal] predecessor [successor]. As a first heuristic we successively eliminate all these vertices regardless of their out-degrees [in-degrees]. Secondly, only those vertices with an out-degree [in-degree] which is less or equal to two are eliminated.

2.2 Case study

We will discuss three `MINPACK` test problems [ACM91].

Chebyshev Quadrature Problem (n=50, m=51): Let us have a look at two different approaches to the a priori reduction of the c-graph for $n = 50$ independent and $m = 51$ dependent variables. It consists of 7826 vertices and 15225 edges. In the first case we have removed all additive parts leading to a c-graph containing 2726 vertices and 68925 edges. Notice the large fill-in that led to an increase of the number of edges by a factor of more than 4. We have applied both the forward and the backward vertex elimination modes to this c-graph resulting of operations counts of 70100 and 622500 respectively. In a second test we have applied the degree-bound-based approach to the a priori elimination of additive parts. This time the resulting c-graph contained 3076 vertices and 22775 edges which is a reduction of the generated fill-in by a factor of 3. The operations counts for both **V_F** (23950) and **V_B** (325850) could be decreased remarkably.

By keeping 350 vertices and thus by performing the corresponding multiplications by one we avoid the large artificial fill-in and we exploit the effect of absorption leading to a significant decrease of the in-degrees in the forward and of the out-degrees in the backward vertex elimination modes.

Coating Thickness Standardization Problem (n=134, m=252): This problem is an example for the a priori reductions leading to the elimination of all intermediate vertices. Starting with a c-graph that contains 1772 vertices and 2520 edges the degree-bound elimination of all sub-maximal vertices having a maximal vertex as their only predecessor leads to the subgraph of the $K_{n,m}$ we are striving for. This is verified by the identical result obtained by running the backward vertex elimination mode.

Gaussian data fitting problem (n=11, m=65): Our third example is supposed to illustrate the efficiency of eliminating all invariant edges and hoisting vertices. For the given problem it results in a decrease of the number of intermediate vertices from 1051 to 531. The run-times of all algorithms for solving the general edge elimination problem in c-graphs are decreased significantly by the reduction.

3 Heuristics for Vertex Elimination

We will now introduce several heuristics for computing nearly optimal vertex [edge] elimination sequences. To begin with we will concentrate on the elimination of whole vertices.

Since our objective is to minimize the cost of computing the complete Jacobian it certainly makes sense to think about how cheaply a particular intermediate vertex v_i can possibly be eliminated. The logical result would be to order all intermediate vertices increasingly by their Markowitz degrees at each stage of the metagraph. At a particular stage $w_k \in M_{\mathbf{V}}$ we eliminate the vertex with the lowest Markowitz degree among all intermediate vertices. This approach is known as the **Lowest-Markowitz-Degree-First** strategy and it represents a robust and easy to implement way of calculating elimination sequences. A more formal description of this heuristic is given below. It can be combined with different tie-break criterions which could improve its performance.

Lowest-Markowitz-Degree-First heuristic (V_LM):

$$w_k \in M_{\mathbf{V}}: \quad v_i < v_j \quad \Leftrightarrow \quad |P_i|_k \cdot |S_i|_k \ \leq \ |P_j|_k \cdot |S_j|_k .$$

At each stage in the metagraph we have to compute the number of predecessors and successors of every vertex which results in a complexity of $O(|V|^2)$ for **V_LM**. As a well-known heuristic for the minimization of the generated fill-in during the solution of sparse systems of linear equations the Markowitz based approach to the vertex elimination problem in c-graphs has already been examined in several papers like for instance in [GrRe91]. However, numerous tests showed that, in general, **V_LM** does not deliver optimal elimination sequences.

This is motivation enough for us to develop different classes of new heuristics most of which result in nearly optimal elimination sequences in many cases. With every basic idea we can combine numerous versions of heuristics belonging to the same class and there is always enough room for experiments and improvements. Here, we will restrict ourselves to the introduction of the framework by presenting one representative of each class.

3.1 Dependency degree based heuristics

Definition 3.1 *Let $CG = (V, E)$ be a c-graph.*

1. *A vertex $v_j \in Y \cup Z$ has the* **input-dependency degree** $\mathtt{id}_j = k$ *if there are paths connecting a maximum of k minimal vertices with v_j.*
2. *A vertex $v_j \in X \cup Z$ has the* **output-dependency degree** $\mathtt{od}_j = k$ *if a maximum of k maximal vertices are reachable from v_j.*
3. *For vertices $v_j \in Z$ we define the* **dependency degree** *to be* $\mathtt{dd}_j \equiv \mathtt{id}_j \cdot \mathtt{od}_j$.

The dependency degree $\mathtt{dd}_j$ of an intermediate vertex v_j is equal to the Markowitz degree at which v_j would be eliminated if it was the last in the elimination sequence. The naive

approach to how cheaply a particular vertex v_j can be eliminated would lead to the earlier discussed Lowest-Markowitz-Degree-First heuristic. A different method could be to evaluate the actual Markowitz degree of v_j relative to some value that is invariant with respect to different elimination sequences. The case in which v_j is eliminated at the highest ($\overline{|P_j|} \cdot \overline{|S_j|}$) [lowest ($\underline{|P_j|} \cdot \underline{|S_j|}$)] cost is one possibility. As the problem of finding $\overline{|P_j|}$ and $\overline{|S_j|}$ [$\underline{|P_j|}$ and $\underline{|S_j|}$] is in general expensive (see [Nau99a]) we help ourselves by exploiting the fact that the dependency degree of a vertex is invariant with respect to the pure vertex elimination strategy. This property makes it ideal for the implementation of new heuristics. Once we have tabulated it for every vertex in the c-graph, which can be done at a cost of $O(|V|(|V| + |E|))$ [MeNä96], we can look it up at every stage $w_k \in M_{\mathbf{V}}$. The computation of the reachability matrix dominates the entire calculation. The dependency degree of a vertex must not be regarded as constant in the context of the general edge elimination method. For a vertex v_j reachable from at least two minimal vertices it is always possible to reduce its input-dependency degree by one (and up to one). Suppose that we want to eliminate the dependency of v_j from $v_i \in X$. We simply have to eliminate all edges belonging to paths which connect v_i with v_j forward followed by the backward elimination of the resulting edge (i, j). One possible instance of a dependency degree based heuristic could be the

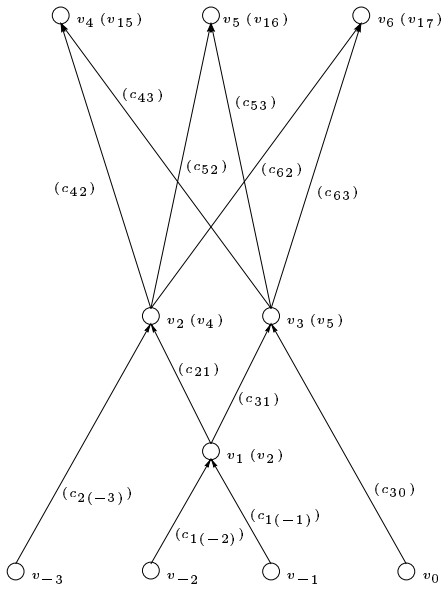

Figure 11: CG

Lowest-Relative-Markowitz-Degree-First heuristic (V_LR):

$$w_k \in M_{\mathbf{V}}: \quad v_i < v_j \quad \Leftrightarrow \quad |P_i|_k \cdot |S_i|_k - \mathtt{dd}_i \ \leq \ |P_j|_k \cdot |S_j|_k - \mathtt{dd}_j.$$

The **V_LR** strategy is slightly more expensive than **V_LM** due to the single calculation of $\mathtt{id}_j$ and $\mathtt{od}_j$ for all intermediate vertices. In many cases it delivers better elimination sequences which more than compensate the additional effort. It could be worth thinking about some tie-break criterion that could possibly even improve the performance of the generated elimination sequences. Extensive tests showed that in general **V_LR** does not give the optimal elimination sequence. But still it is a fast and practicable way of finding nearly optimal solutions in most cases. It turned out to be the most consistent strategy in comparison with **V_F**, **V_B**, and **V_LM**.

3.2 Example

We will go on with the example from Section 1.3. The application of the a priori reduction rules to the complete c-graph CG_F^c displayed in Figure 5 leads to its reduced version which is shown in Figure 11 and which contains as little intermediate vertices as possible.

Obviously, it is easy to solve the resulting vertex elimination problem in three intermediate variables. It is even possible to check all different orders. Notice that **V_LM** does not deliver the minimal operations count in this very simple case. It behaves like **V_F** resulting in 22 multiplications. Considering **V_LR** we have $\mathtt{dd}_1 = 6$, $\mathtt{dd}_2 = \mathtt{dd}_3 = 9$ and since $6 - 9 > 4 - 6$ the vertex v_2 is eliminated first. At the next stage we get $6 - 9 > 8 - 6$ and therefore **V_LR** would act similar to **V_B** delivering the minimal operations count which is 18 for this example.

3.3 Fill-in based heuristics

As a result of the elimination rule the elimination of a vertex v_i at a stage w_k in the metagraph causes the insertion of

$$f_i^k = |P_i|_k \cdot |S_i|_k - (|P_i|_k + |S_i|_k) - \zeta_i^k \tag{2}$$

new edges into the c-graph where ζ_i^k is the number of edges connecting predecessors of v_i with its successors. Every edge labeled with an elementary partial derivative is a potential factor in the elimination process which represents the computational process of accumulating the complete Jacobian. A logical consequence of these observations is to keep the number of these factors as low as possible which implies the minimization of the locally produced fill-in. This idea is exploited in the

Lowest-Fill-in-First heuristic (V_LF)

$$w_k \in M_{\mathbf{V}}: \quad v_i < v_j \quad \Leftrightarrow \quad f_i^k \leq f_j^k.$$

The relatively high time complexity ($O(|V|^3)$) of this heuristic is caused by the repeated computation of ζ_i^k. Therefore, we have also implemented a reduced version where we simply consider $|P_i|_k \cdot |S_i|_k - (|P_i|_k + |S_i|_k)$ which leads to a $O(|V|^2)$ algorithm.

3.4 Exploiting information on paths through vertices

The repeated application of the chain rule yields a general expression for the partial derivative c_{kl} in the scalar mode on scalar c-graphs:

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

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 $c_{kl} = f_k'(v_l)$. Then $c_{kl} = c_{kl}^{2^d - 1}$, i.e. all edges lying on paths connecting v_l with v_k have to be eliminated in order to calculate the unique final value of c_{kl}.

Speaking about paths in the c-graph we make an important observation: 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 $c_{j_1 i_1} c_{j_2 i_2} \ldots c_{j_k i_k}$ 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.

The number of distinct paths $n(v_i)$ through one vertex in CG can be calculated by performing one forward or one reverse sweep. Let $n^b(v_i)$ $[n^f(v_i)]$ denote the number of different paths leading into [emanating from] a vertex $v_i \in CG$. Initializing $n^b(v_i) = 1$ for $i \in X$ and $n^f(v_i) = 1$ for $i \in Y$ we get

$$n^b(v_i) = \sum_{j \in P_i} n^b(v_j) \qquad \text{and} \qquad n^f(v_i) = \sum_{j \in S_i} n^f(v_j). \tag{3}$$

It follows that

$$n(v_i) = \begin{cases} n^b(v_i), & i \in Y \\ n^f(v_i), & i \in X \\ n^b(v_i) n^f(v_i), & i \in Z \end{cases}$$

and we get

$$\mathcal{N} = \sum_{j \in X} n(v_j) = \sum_{j \in Y} n(v_j). \tag{4}$$

Using the information about the number of different paths through a given vertex in the c-graph we can calculate the total length of all paths in CG again by either one forward or one reverse sweep. 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

$$l^b(v_i) \quad = \quad \sum_{j \in P_i} \left[l^b(v_j) + n^b(v_j) \right] \quad = \quad n^b(v_i) + \sum_{j \in P_i} l^b(v_j)$$

and

$$l^f(v_i) \quad = \quad \sum_{j \in S_i} \left[l^f(v_j) + n^f(v_j) \right] \quad = \quad n^f(v_i) + \sum_{j \in S_i} l^f(v_j)$$

for the total length of all paths in CG:

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

We have initialized $l^b(v_i) = 0$ for $i \in X$ and $l^f(v_i) = 0$ for $i \in Y$. Consider the behavior of both $\mathcal{N}$ and $\mathcal{L}$ with respect to the elimination of edges in the c-graph. For the forward elimination of (i,j) we have

$$n^b(v_j) := n^b(v_j) \; - \; n^b(v_i) \; + \sum_{k \in P_i} n^b(v_k) \; - \sum_{k \in P_j \cap P_i} n^b(v_k)$$

while $n^f(v_j)$ remains unchanged. From $n^b(v_i) = \sum_{k \in P_i} n^b(v_k)$ follows

$$n^b(v_j) \mathrel{-}= \sum_{k \in P_j \cap P_i} n^b(v_k)$$

which implies that the number of paths leading into v_j decreases if some path containing (i,j) coalesces with another one due to the absorption of (i,j) by an edge leading from $P_j \cap P_i$ to v_j. Otherwise, $n^b(v_j)$ is not changed by the elimination of (i,j). The forward elimination of (i,j) decreases the out-degree of v_i, thus, also reducing the number of different paths starting in v_i. Its in-degree is not effected at all giving

$$n^f(v_i) \mathrel{-}= n^f(v_j) \quad \Rightarrow \quad n(v_i) \mathrel{-}= n^f(v_j) \cdot n^b(v_i).$$

Let Y_j denote the set of indices such that for all $k \in Y_j$ v_k is a maximal vertex in CG reachable from v_j. From Equation (3) follows immediately that the number of paths leading into each of the v_k ($k \in Y_j$) depends linearly on the values of $n^b(v_l)$ for all v_l that are connected to v_k by a path. We have shown that the forward elimination of (i,j) may either result in a decrease of the number of paths leading into v_j, while not changing this value for any other vertex in CG, or this number may remain constant. An interesting consequence is that although the forward elimination of (i,j) always decreases $n^f(v_i)$ this does not necessarily lead to a decrease of the number of paths emanating from the minimal vertices in CG (see Equation (4)).

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).

Finally, we can state that $\mathcal{N}$ is monotonically decreasing whereas $\mathcal{L}$ is strictly monotonically decreasing during the general edge elimination process. 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.

Nothing changes substantially with respect to $\mathcal{N}$ and $\mathcal{L}$ when we go over to vector mode according to Section 1.4. The structure of the c-graph is invariant with respect to the different modes and even the local partial derivatives are still scalars. It follows that an upper bound for the number of products involved in the accumulation of the Jacobian by the naive component-wise calculation of its entries is still $\mathcal{N}$. However, the propagation of tangent [gradient] bundles of length $q \geq 1$ results in an upper bound for the total count of scalar multiplications which is equal to $q \cdot \mathcal{L}$.

As a direct consequence of the local partial derivatives being matrices we have a different interpretation of $\mathcal{N}$ and $\mathcal{L}$ in general vector mode. Let their meaning in a graph-theoretical context remain unchanged. $\mathcal{N}$ still denotes the number of distinct paths in the c-graph while $\mathcal{L}$ stands for the sum of the total lengths of these paths. However, the products of the local partial derivatives along the paths are now chained matrix products. It follows that we could use a dynamic programming algorithm [HoSa78] for minimizing the number of scalar multiplications involved in the evaluation of each single matrix chain. The corresponding maximum would again give us an upper bound for the number of scalar multiplications required for the computation of the complete Jacobian which we intent to minimize.

3.4.1 $\mathcal{L}$-based heuristics

We have developed two heuristics that exploit the above observations and we will present them both. Notice that $\mathcal{L}$ is not a static quantity. It depends on the stage in the metagraph of a given elimination problem. At a stage w_k in the vertex metagraph $M_{\mathbf{V}}$ we denote the total length of all paths in CG_k by $\mathcal{L}_k$. For any vertex path in $M_{\mathbf{V}}$ $(w_{k_1}, \ldots, w_{k_p})$ (each of them has length $p \equiv |Z|$) we have $\mathcal{L}_{k_1} > \ldots > \mathcal{L}_{k_p}$ where $\mathcal{L}_{k_p}$ is equal to the number of non-zero entries in the Jacobian. $\mathcal{L}_{k_p}$ can also be regarded as the sum of the input-dependency degrees of all maximal vertices which is precisely the same as the sum of the output-dependency degrees of all minimal vertices.

The idea behind our first $\mathcal{L}$-based heuristic is to find a path in $M_{\mathbf{V}}$ such that the difference $\mathcal{L}_{k_i} - \mathcal{L}_{k_{i+1}}$ is maximized by eliminating a vertex which causes the maximum decrease of the upper bound for the number of scalar multiplications required for accumulating the Jacobian.

Maximum-Total-Path-Length-Reduction-First heuristic (V_RPL):

Suppose that starting at a stage w_k in the metagraph the elimination of a vertex $v_i \in CG_k$ leads to stage $w_{k_i} \in M_{\mathbf{V}}$ and the elimination of $v_j \in CG_k$ is equivalent to move to stage $w_{k_j} \in M_{\mathbf{V}}$. Then

$$w_k \in M_{\mathbf{V}}: \quad v_i < v_j \quad \Leftrightarrow \quad \mathcal{L}_k - \mathcal{L}_{k_i} \geq \mathcal{L}_k - \mathcal{L}_{k_j}$$

which is the same as

$$w_k \in M_{\mathbf{V}}: \quad v_i < v_j \quad \Leftrightarrow \quad \mathcal{L}_{k_i} \leq \mathcal{L}_{k_j}$$

and it can be implemented at a cost of $O(|V|^3)$. The above approach does not take into account the actual Markowitz degrees of the vertices which are the costs of eliminating them from the c-graph. Since our objective is to minimize the overall Markowitz degree by finding a shortest path in the vertex metagraph it certainly would make sense to include this information into our heuristic. Therefore, we have implemented a second version where we regard the new total path length caused by the elimination of a vertex $v_i \in CG_k$ weighted by its actual Markowitz degree $|P_i|_k \cdot |S_i|_k$ at the current stage $w_k \in M_{\mathbf{V}}$.

Minimum-Relative-Total-Path-Length-First heuristic (V_RLR):

$$w_k \in M_{\mathbf{V}}: \quad v_i < v_j \quad \Leftrightarrow \quad (|P_i|_k \cdot |S_i|_k) \cdot \mathcal{L}_{k_i} \leq (|P_j|_k \cdot |S_j|_k) \cdot \mathcal{L}_{k_j}.$$

The implementation of the $\mathcal{L}$-based heuristics is rather expensive as at each stage w_k in the metagraph and for all vertices in CG_k one has to perform one sweep through the c-graph in order to compute the new total path length. Thus, we end up with an algorithm of order $O(|V|^3)$ for **V_RLR**.

3.4.2 $\mathcal{N}$-based heuristics

Let us now consider the actual Markowitz degree of $v_i \in CG$ relative to the number of paths leading through it. The idea is that if $|P_i|_k \cdot |S_i|_k$ is small at the current stage w_k in the metagraph and if there are many paths leading through v_i then the Markowitz degree of v_i is likely to be increased at a later stage of the elimination process. Similarly to the *Lowest-Relative-Markowitz-Degree-First* heuristic this approach can be regarded as a *Lowest-Markowitz-Degree-First* strategy extended by some global information represented by the number of paths leading through a vertex.

Extended-Lowest-Markowitz-Degree-First heuristic (V_ELM):

$$w_k \in M_{\mathbf{V}}: \quad v_i < v_j \quad \Leftrightarrow \quad \frac{|P_i|_k \cdot |S_i|_k}{n_k(v_i)} \leq \frac{|P_j|_k \cdot |S_j|_k}{n_k(v_j)}.$$

Since we have to compute both the number of paths through a vertex and its Markowitz degree at each stage in $M_{\mathbf{V}}$ we end up with an algorithm the complexity of which is $O(|V|^3)$.

4 Heuristics for Edge Elimination

The ideas behind the heuristics for finding nearly optimal edge elimination orders are similar to the ones exploited in the vertex case. Undoubtedly, there are equivalents for both the forward and backward elimination modes provided that an appropriate re-numbering scheme which will be necessary due to the generated fill-in is used.

Fill-in based heuristics

Remember that the fill-in caused by the elimination of an intermediate edge (i,j) is defined as the number of edges in the c-graph after minus the number of edges before the elimination. The fill-in generated by the forward elimination of an edge (i,j) is given by

$$\begin{aligned} & \texttt{ffi}\{(i,j)\} = |S_j| - \overline{\kappa}_{ij} - 1 \qquad \text{if } |P_j| > 1 \\ \text{or} \quad & \texttt{ffi}\{(i,j)\} = -(\overline{\kappa}_{ij} + 1) \qquad \text{if } |P_j| = 1 \end{aligned}$$

where $\overline{\kappa}_{ij}$ denotes the number of common successors of v_i and v_j. Analogous, the fill-in generated by the backward elimination of the same edge is given by

$$\begin{aligned} & \texttt{bfi}\{(i,j)\} = |P_i| - \underline{\kappa}_{ij} - 1 \qquad \text{if } |S_i| > 1 \\ \text{or} \quad & \texttt{bfi}\{(i,j)\} = -(\underline{\kappa}_{ij} + 1) \qquad \text{if } |S_i| = 1 \end{aligned}$$

where $\underline{\kappa}_{ij}$ denotes the number of common predecessors of v_i and v_j. We have implemented a procedure which calculates both values $\texttt{ffi}\{(i,j)\}$ and $\texttt{bfi}\{(i,j)\}$. Depending on which of them is lower we eliminate (i,j) either forward or backward:

$$w_k \in M: \quad (i_1,j_1) <_{[f,b]} (i_2,j_2)$$

$$\Leftrightarrow$$

$$\min\{\texttt{ffi}\{(i_1,j_1)\}, \texttt{bfi}\{(i_1,j_1)\}\} \leq \min\{\texttt{ffi}\{(i_2,j_2)\}, \texttt{bfi}\{(i_2,j_2)\}\}.$$

We will refer to this heuristic as the **Lowest-Fill-in-First** edge elimination heuristic (**E_LF**). The notation $(i_1,j_1) <_{[f,b]} (i_2,j_2)$ is supposed to indicate that (i_1,j_1) is eliminated either forward or backward depending on the condition

$$\texttt{ffi}\{(i_1,j_1)\} \leq \texttt{bfi}\{(i_1,j_1)\}.$$

Operations count based heuristics

The minimal number of multiplications required for the elimination of an edge (i,j) is the minimum out of the number of predecessors of its source and the number of successors of its target. This leads to a heuristic which could be regarded as equivalent to **V_LM**. The **Lowest-Markowitz-Degree-First** edge elimination heuristic will be denoted by **E_LM**:

$$w_k \in M: \quad (i_1,j_1) <_{[f,b]} (i_2,j_2)$$

$$\Leftrightarrow$$

$$\min\{|P_{i_1}|_k, |S_{j_1}|_k\} \leq \min\{|P_{i_2}|_k, |S_{j_2}|_k\}.$$

(i_1,j_1) is eliminated either forward or backward depending on the condition $|P_{i_1}|_k \leq |S_{j_1}|_k$.

Dependency degree based heuristics

As in the case of vertex elimination we can consider the minimal number of multiplications required for the elimination of an edge (i,j) relative to the actual input-[output-]dependency degree of v_i $[v_j]$, resulting in the **Lowest-Relative-Markowitz-Degree-First** edge elimination heuristic (**E_LR**):

$$w_k \in M: \quad (i_1,j_1) <_{[f,b]} (i_2,j_2)$$

$$\Leftrightarrow$$

$$\max\{\mathtt{id}_{i_1} - |P_{i_1}|_k, \mathtt{od}_{j_1} - |S_{j_1}|_k\} \geq \max\{\mathtt{id}_{i_2} - |P_{i_2}|_k, \mathtt{od}_{j_2} - |S_{j_2}|_k\}.$$

Notice that the dependency degree of a vertex is not invariant with respect to the general edge elimination method and has therefore to be recalculated at every stage in the metagraph. However, as a simplification, we could perform the calculation just once and consider the vertex degrees relative to these initial dependency degrees.

As for the vertex elimination heuristics we have presented one representative of each class. Building on the above observations we can construct numerous variations by introducing different tie-break criterions or by altering certain parameters. It remains to mention that none of the presented heuristics for edge elimination ever lead to a lower operations count for the accumulation of the Jacobian then the best known vertex elimination heuristic when applied to our selection of test problems.

5 Case Study

5.1 Absorption function

Consider a function of the form $y = f(x)$ with $f: \mathbb{R}^n \to \mathbb{R}$ and

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

A special case is given by

$$y = f_a(x) = \prod_{i=0}^{n-1} \left(\cos\left(x_i + i(2) \cdot \frac{\pi}{2}\right) \cdot \prod_{j=1}^{2(i+1)} x_i \right)$$

where $i(2) \equiv i \mod 2$ and which we will refer to as the *absorption function*. This name is motivated by the shape of the c-graph which is displayed in Figure 12 and the behavior of f_a with respect to different vertex elimination strategies. It represents an example where as a result of the absorption of potentially new edges the backward elimination mode is not necessarily optimal in terms of the number of multiplications required for the computation

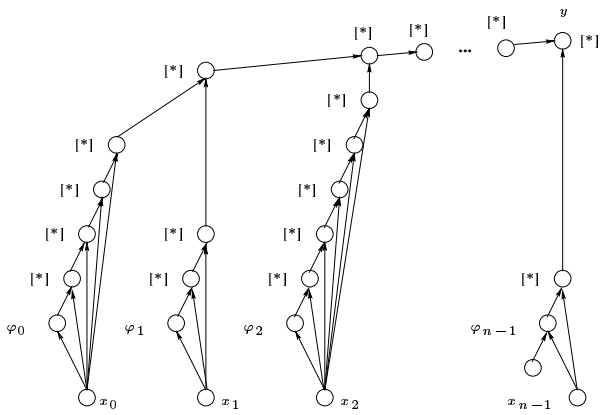

Figure 12: Absorption Function

of a gradient. Looking at the c-graph of f_a we observe the following: The backward vertex elimination mode starts with the elimination of the $n-2$ multiplications performed in the outer loop at a Markowitz degree of two each. It goes on with the $2(i+1)$ multiplications from the inner loop, again, each of them at a cost of two followed by the n univariate scalar functions φ_i. Hence,

$$\mathbf{OPS_{V\_B}}\{CG_{f_a}\} \quad = \quad 2(n-2) + 2\sum_{i=0}^{n-1} e_i + n.$$

Using the forward vertex elimination strategy we would start with the φ_i followed by the elimination of all multiplications of the inner loop at a Markowitz degree of one due to the exploitation of absorption. Finally, the outer loop multiplications are eliminated successively with their costs incremented by one after each step:

$$\mathbf{OPS_{V\_F}}\{CG_{f_a}\} \quad = \quad \binom{n}{2} - 1 + \sum_{i=0}^{n-1}(e_i+1).$$

Obviously, we would prefer some combination of the forward and backward elimination modes in order to be able to determine the minimal cost for accumulating the gradient of f_a. This is what our **V_LR** heuristic does. It makes full use of absorption as the forward mode does while avoiding the incrementation of the vertex degrees by running through the outer loop backward:

$$\mathbf{OPS_{V\_LR}}\{CG_{f_a}\} \quad = \quad 2(n-2) + \sum_{i=0}^{n-1}(e_i+1).$$

For our special case we have with $e_i = 2(i+1)$:

$$\begin{aligned} \mathbf{OPS_{V\_B}}\{CG_{f_a}\} &= 2n^2 + 5n - 4, \\ \mathbf{OPS_{V\_F}}\{CG_{f_a}\} &= \frac{3}{2}n^2 + \frac{3}{2}n - 1, \\ \mathbf{OPS_{V\_LR}}\{CG_{f_a}\} &= n^2 + 4n - 4. \end{aligned}$$

We observe that with increasing values of n **V_B** becomes better compared to **V_F** while never reaching the same low cost. Moreover, neither forward nor backward vertex elimination are optimal with respect to the number of multiplications required for the accumulation of the gradient of f_a. As n increases their costs converge to a fixed multiple of the value of the optimum.

Note: As for the *belt function* we have applied the verified global optimization algorithm which is described in [Nau94] to f_a. Both functions seem to be suitable as test problems for algorithms for unconstrained optimization.

5.2 Discretization of evolutionary equations

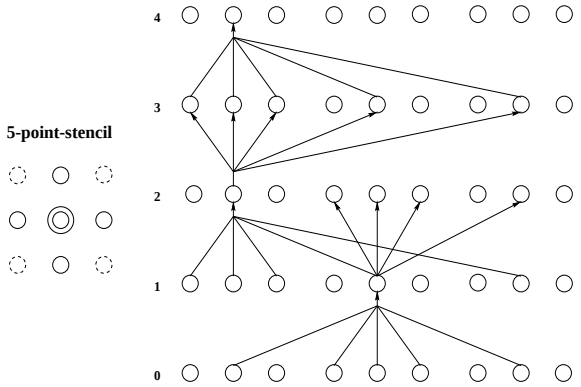

Figure 13: Evolutionary Equations

Suppose we have a local evolution process on a circle so that after discretization

$$z_{t+1} = C_t'(z_t) \quad \text{for} \quad t = 0, 1, \ldots, l-1,$$

where $z_t \in I\!R^\eta$ describes the state at time t at η points on the circle. Assuming differentiability and locality we find that the local Jacobians $C'(z_t) \equiv C_t'$ exist and are bounded.

Without loss of generality we assume that the C'_t are tridiagonal with a non-zero $(1,\eta)$ and $(\eta,1)$ element, i.e. cyclic. Let us consider the task of computing

$$\frac{\partial z_l}{\partial z_0} = C'_{l-1}C'_{l-2}\cdots C'_1C'_0$$

with a minimal number of operations. The forward and backward methods would calculate according to the parenthesizations

$$C'_{l-1}(C'_{l-2}\cdots(C'_2(C'_1C'_0))\cdots)$$

and

$$(\cdots((C'_{l-1}C'_{l-2})C'_{l-3})\cdots C'_1)C'_0,$$

respectively. Now let us examine the question whether $\partial z_l/\partial z_0$ can be obtained cheaper (or more expensive) by other methods, i.e. parenthesization schemes. A good contender would seem to be the Markowitz compatible scheme

$$((C'_{l-1}C'_{l-2})(C'_{l-3}C'_{l-4}))\cdots((C'_3C'_2)(C'_1C'_0)). \tag{5}$$

We can show that in this case Markowitz is in fact optimal and slightly more efficient than forward or backward in terms of the operations count.

Now, suppose we consider a regular cube of $(1+\eta)^d$ points in $\mathbb{R}^d$. Assuming periodicity of the nodal states we only have to consider η^d points. If the next state in each point depends only on the nearest neighbors in the $l_\infty-$norm (as in the one dimensional case considered first) the Jacobians C'_t have $\eta^d\cdot 3^d=(3\eta)^d$ non-zero entries. The product $C'_{t+1}C'_t$ has $(5\eta)^d$ non-zero entries which represents a cube of half width 2 rather than 1.

It is possible to show that for the multidimensional situations Markowitz is definitely not the right way to go. This is supported by our test results. We have applied several heuristics to c-graphs resulting from the application of a five-point stencil to a two-dimensional problem. Figure 13 shows the corresponding dependency scheme with the vertices drawn with dashed lines not depending on the locally dependent vertex in the center. On the right hand side we have displayed a c-graph for a two-dimensional problem with $\eta=3$ and $l=5$.

While both **V_F** and **V_B** eliminate the vertices in CG level by level **V_LM** performs several sweeps through the graph while subsequently eliminating every second level. This behavior leads to a strong increase of the Markowitz degrees of the remaining vertices with every traversal. Referring to the example graph in Figure 13 we observe that there are 3 intermediate levels to eliminate. The optimal strategies (e.g. **V_F** and **V_B**) would eliminate one level at a cost of 25 multiplications per vertex followed by the remaining two levels at $9\cdot 45$ scalar multiplications resulting in a total cost of 1035. **V_LM** with a simple forward approach as primary tie-breaker would start with the first level $(\overline{9}\cdot 25)$ followed by the third at the same cost and, finally, it would have to eliminate the vertices at the second level at a cost of $9\cdot 81$. This process would take 1179 scalar multiplications. The following table lists the values for a selection of heuristics applied to the simple graph in Figure 13. **V_HM** stands

for a **Highest-Markowitz-Degree-First** approach which represents an upper bound for all our heuristics in most cases. It is supposed to give an impression of "how bad" it may get.

V_B	**V_LM**	**V_HM**	**V_RPL**	**V_LR**	**E_LF**	**E_LM**
1035	1179	1869	1035	1035	1421	1190

The above example is supposed to illustrate the problem and give a first idea on how different heuristics would behave. For **V_RPL** we observe that since the reduction in $\mathcal{L}$ remains constant the tie-breaker **V_F** applies which makes this heuristic succeed. **V_LR** is still optimal on this small graph but it will "fail" on evolutions in general as illustrated below. Here we have listed the values for a larger problem with $\eta = 8$ and $l = 50$ resulting in a c-graph which contains 3136 intermediate vertices and 16000 edges. While, obviously, **V_RPL** remains optimal this is not true for **V_LR** anymore. In general, we observe that **V_RPL** is optimal on discretizations of evolutionary equations whereas all Markowitz related heuristics are not.

V_F	**V_LM**	**V_HM**	**V_LR**	**V_RPL**
944 960	2 150 144	4 399 445	2 019 584	944 960

6 Numerical Results

6.1 Unsaturated flow problem

The first test problem which we will discuss more in detail is a two-dimensional unsaturated flow problem in a porous medium. An extensive description of it can be found in [CGRW92]. Its Jacobian J is an extremely sparse 1989×1989 square matrix yielding a very regular sparsity structure arising from the underlying discretization grid. Notice that our edge elimination algorithms do not make use of this knowledge. We have chosen to optimize the computation of the sub-matrix $\hat{D} = J(0..935, 936..1286)$ of the Jacobian. The corresponding c-graph consists of 351 minimal, 936 maximal, and 21177 intermediate vertices. According to what we have introduced in Section 1.5 the approximate operations counts for the dense forward (**DF**) and for the dense reverse modes (**DR**) of AD are then equal to 350 thousand and 20 million multiplications, respectively. However, T. Robey [CGRW92] recognized that the 351 rows of $\hat{D}$ could be computed in only 6 passes. This would result in an remarkably decreased operations count of approximately 133 thousand in sparse forward mode, yielding an improvement by a factor of nearly 3.

By fully exploiting the sparsity of the problem in forward vertex elimination mode the cost could be decreased even further (73 thousand multiplications). The application of the cross-country vertex elimination method in form of the **V_LR** heuristic delivered the lowest known number of multiplications required for accumulating $\hat{D}$ (29 thousand), so far. A priori elimination of all hoisting vertices (see Section 2) led to a reduction of the number of intermediate vertices by 17433.

To summarize the above, we observe that the full exploitation of the sparsity leads to savings by a factor of 6. By solving the combinatorial problem of finding an optimal elimination sequence we could decrease this number by another factor of 2.5.

6.2 Further results

We have applied the cross-country elimination method to more than a hundred test problems out of which we will present a small but representative subset without a detailed discussion for each of them below. The numbers will speak for themselves. Consider the following eleven test problems:

PDJ:	Pressure distribution in a journal bearing problem;
SSC:	Steady state combustion problem;
FDC:	Flow in a driven cavity problem;
FCH:	Flow in a channel problem;
WAT:	Watson function;
DIE:	Discrete integral equation function;
VDI:	Variably dimensioned function;
PEN:	Penalty function II;
EXP:	Extended Powell function;
SPE:	Speelpenning function;
GDF:	Gaussian data fitting problem.

Apart from the Speelpenning function [Spe80] the above examples are taken from the MINPACK test problem suite [ACM91]. In the following table we will compare the values delivered by the **V_LR** vertex elimination heuristic with the theoretical operations counts that can be achieved using state-of-the-art AD-technology (see Section 1.5). Here, $[\hat{n}, \hat{m}]$ denotes the minimum out of the maximal numbers of non-zero elements per row and per column in the Jacobian.

	n	p	m	$[\hat{n}, \hat{m}]$	**DF**	**DR**	**NR**	**V_LR**	**NR / V_LR**
PDJ	64	1447	1	1	92672	1511	1511	1278	**1.18**
SSC	16	655	1	1	10496	671	671	452	**1.49**
FDC	16	984	16	11	16000	16000	11000	930	**11.83**
FCH	32	1209	32	9	39712	39712	11169	845	**13.22**
WAT	7	1683	7	7	11830	11830	11830	4240	**2.79**
DIE	20	2499	20	20	50380	50380	50380	1659	**30.34**
VDI	100	504	100	100	60400	60400	60400	10301	**5.86**
PEN	200	2399	1	1	480000	2599	2599	1798	**1.45**
EXP	96	479	1	1	4680	575	575	504	**1.14**
SPE	50	48	1	1	2450	98	98	96	**1.02**
GDF	11	1625	65	11	18590	106340	18590	1430	**13.0**

Considering the ratio between the optimal one-sided Newsam-Ramsdell approach and the **V_LR** vertex elimination mode we observe that savings within the range from nearly 3 up to 30 can be achieved. Obviously, this cannot be the case for the computation of single gradi-

ents. However, for $\{n, m\} \gg 1$ we get a significant decrease of the number of multiplications involved in the accumulation of the Jacobian for virtually all problems.

The next table shows a comparison between the different approaches to the solution of the shortest path problem. We have also listed the values achieved by the better choice out of the forward and the backward vertex elimination modes, denoted by [**V_F**,**V_B**]. Notice that these results imply that we know which of the two methods is actually "the better choice". The differences between the results delivered by **V_F** and **V_B** can be very large in some cases.

	[V_F,V_B]	V_LM	V_LR	V_LF	V_RPL	V_ELM	E_LF	worst/best
PDJ	1376	1278	1278	1409	1368	1376	1307	1.1
SSC	516	452	452	452	496	558	452	1.2
FDC	930	1338	930	1030	1002	1616	1106	1.7
FCH	941	845	845	925	845	941	941	1.1
WAT	6473	4240	4240	4240	7488	7838	4240	1.9
DIE	2059	1659	1659	1864	1659	2001	2001	1.2
VDI	20400	10401	10301	10301	20201	15791	10301	1.9
PEN	1996	1798	1798	1857	1996	2011	1798	1.1
EXP	576	504	504	566	552	536	571	1.1
SPE	96	96	96	273	96	1224	273	12.8
GDF	1430	1430	1430	1528	1430	1625	1625	1.1

For most real-world problems the generation of optimized derivative code based on either forward or backward vertex elimination sequences combined with the pre-elimination of all hoisting vertices would result in remarkable savings in the overall operations count. The fact that it is not clear a priori whether we should prefer the forward or the backward approach is one of the problems. At this point it could be useful to exploit the strength of heuristics for determining nearly optimal vertex elimination sequences for almost all sorts of c-graphs. **V_LR** turned out to be very consistent, while being only slightly more expensive than the pure uni-directional methods. It could be worth to analyze selected evaluation routines deeper by using more costly optimization methods like dynamic programming or simulated annealing. However, the additional effort is justified only if the resulting derivative code is generated once and is then used over and over again.

7 Summary, Conclusion and Outlook

The goal of this paper was to present an approach to the efficient accumulation of Jacobian matrices using elements from graph theory and 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 non-zero 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

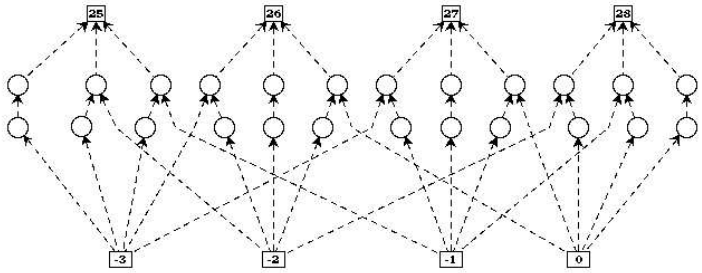

Figure 14: Solid Fuel Ignition Problem

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. We conjecture that the vertex-edge discrepancy is less or equal to 2. Therefore, we have put the focus on vertex elimination for the development of heuristics for solving the shortest path problem in the metagraph.

We have presented several heuristics for reducing the size of the c-graph a priori as well as heuristics for computing nearly optimal edge and vertex elimination sequences. Their complexity is polynomial in the number of intermediate vertices in the c-graph. Most of our theoretical results were implemented and tested on numerous example problems.

The exploitation of the results of the cross-country elimination method should lead to the automatic generation of optimized derivative code. Since memory accesses have a strong impact on the run-time of the code we should concentrate on locally bounded parts of the c-graph in order to make full use of the reduction in the operations count. Hierarchical vertex elimination could be one way to ensure this.

We are not able to generate optimized derivative code automatically, so far. Therefore, we have decided to compare the run-times of a **V_LR**-based hand-written code (`Code 2`) with a scalar tangent code without exploitation of sparsity (`Code 1`) for the Solid Fuel Ignition problem [ACM91]. Figure 14 shows the corresponding c-graph. Regarding the number of multiplications involved in the computation of the (4×4)-Jacobian it is possible to achieve savings by a factor of 4:

n	p	m	$[\hat{n}, \hat{m}]$	**DM**	**NR**	**V_LR**	**DM / V_LR**
4	24	4	3	128	84	32	**4**

Using the corresponding derivative codes presented in [Nau99a] we have checked whether these savings can be transfered to decreases in the run-time by a comparably high factor. We have considered the elapsed CPU times for 10^5 successive calculations of the complete Jacobian on a SUN Sparc20:

	`Code 1`	`Code 2`	`Code 3`	`Code 4`	`Code 1 / Code 2`
t_{CPU} (in sec)	9.8	2.8	2.6	3.15	**3.5**

In addition to codes 1 and 2 we have included the optimal hand-coded Jacobian (`Code 3`) and the derivative code supplied by the `MINPACK` package. Obviously, for this small problem the memory accesses will not have any negative impact on the run-time of cross-country elimination sequences. This results in the factor of 3.5 by which `Code 2` ran faster than `Code 1`.

To summarize the above, it appears to be useful to work on the automatic generation of optimized adjoint code based on different elimination sequences. In order to be able to handle large-scale evaluation programs the hierarchical approach has to be implemented efficiently. Therefore, we require tools for analyzing the code which generate some (ideally standardized) intermediate form which contains all the necessary information [Bro98], [BRM96].

From the theoretical point of view, the proof of the NP-completeness of the general edge elimination problem is still not given. Our conjecture about the small constant vertex-edge discrepancy has a more practical relevance. To support the search for its proof the implementation of a simulated annealing algorithm for optimizing edge elimination sequences could be useful. The principle is the same as for vertex elimination sequences. We simply have to adapt the annealing schedule such that rearrangements including both forward and backward elimination of edges become possible.

The current version of our software package **OESCOMP** is to be regarded as experimental. Since it represents a useful tool for supporting both research and teaching in the field of AD we would like it to be subject to further development.

We believe that the efficient implementation of different edge elimination algorithms that lead to the automatic generation of optimized derivative code will take AD a large step forward.

References

[ACM91] B. Averik, R. Carter, and J. More, *The Minpack-2 test problem collection (preliminary version)*, Technical Memorandum No. 150, Mathematical and Computer Science Division, Argonne National Laboratory, 1991.

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

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

[BRM96] C. Bischof, L. Roh, and A. Mauer, *ADIC: An extensible automatic differentiation tool for ANSI-C*, Preprint ANL/MCS-P626-1196, Argonne National Laboratory, March 1997.

[Bro98] S. Brown, *Models for automatic differentiation: A conceptual framework for exploiting program transformation*, PhD thesis, Computer Science, University of Hertfordshire, Hatfield, England, February 1998.

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

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

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

[Har69] F. Harary, *Graph Theory*, Addison-Wesley, 1969.

[HoSa78] E. Horowitz and S. Sahni, *Fundamentals of computer algorithms*, Computer Science Press, Inc., 1978.

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

[Nau94] U. Naumann, *Globale Optimierung mit Ergebnisverifikation auf Multiprozessorsystemen*, Diplomarbeit, TU Dresden, 1994.

[Nau99a] 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.

[Nau99b] U. NAUMANN, *Optimizing the Accumulation of Jacobians by Edge Elimination in Computational Graphs* INRIA Rapport de Recherche Nr. 3659, Institut National de Recherche en Informatique et Automatique, Sophia-Antipolis, France, 1999.

[Nau99c] U. NAUMANN, *SAVE - Simulated Annealing applied to the Vertex Elimination Problem in Computational Graphs* INRIA Rapport de Recherche Nr. 3660, Institut National de Recherche en Informatique et Automatique, Sophia-Antipolis, France, 1999.

[NeRa83] G. NEWSAM AND J. RAMSDELL, *Estimation of sparse Jacobian matrices*, SIAM J. Alg. Discr. Meth., 4 (1983), pp. 404-417.

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

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

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