Markowitz-Type Heuristics for Computing Jacobian Matrices Efficiently

Andreas Albrecht, Peter Gottschling[1], and Uwe Naumann[2]

[1] Department of Computer Science, University of Hertfordshire, College Lane, Hatfield AL10 9AB, UK
{A.Albrecht, P.1.Gottschling}@herts.ac.uk

[2] Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA
naumann@mcs.anl.gov

Abstract. We consider the problem of accumulating the Jacobian matrix of a nonlinear vector function by using a minimal number of arithmetic operations. Two new Markowitz-type heuristics are proposed for vertex elimination in linearized computational graphs, and their superiority over existing approaches is shown by several tests. Similar ideas are applied to derive new heuristics for edge elimination techniques. The well known superiority of edge over vertex elimination can be observed only partially for the heuristics discussed in this paper. Nevertheless, significant improvements can be achieved by the new heuristics both in terms of the quality of the results and their robustness with respect to different tiebreaking criteria.

1 Introduction

Consider the nonlinear vector function $F : I\!R^2 \to I\!R^2$ that is given by the following sequence of scalar assignments:

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

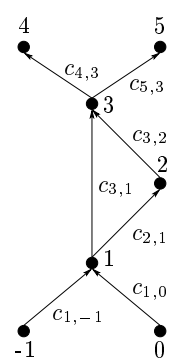

Fig. 1. F

For simplicity, all variables carry a unique index. There are $n = 2$ independent, $p = 3$ intermediate, and $m = 2$ dependent variables. Mathematical functions that are implemented as computer programs written in an imperative programming language such as C or Fortran can always be decomposed to meet this requirement. The structure of such computations can be visualized by a directed acyclic graph (dag) $G = (V, E)$ as shown in Figure 1. If one assumes that jointly continuous local partial derivatives

$$c_{j,i} \equiv \frac{\partial}{\partial v_i}\varphi_j(v_k)_{k \prec j}, \quad j = 1, \ldots, p+m \quad ,$$

of the elemental functions (e.g., $*, \sin, \cos, \exp$) with respect to its arguments exist in some neighborhood of the current point, the corresponding numerical values, computed as

$$c_{1,-1} = v_0;\ c_{1,0} = v_{-1};\ c_{2,1} = \cos(v_1);\ c_{3,1} = v_2;\ c_{3,2} = v_1;\ c_{4,3} = -\sin(v_3);\ c_{5,3} = v_5 \quad ,$$

can be attached to the edges in the dag. The notation $k \prec j$ is used to indicate that v_k is an argument of φ_j. Thus one gets the linearized computational graph (or c-graph) $G = (V, E)$ of F as displayed in Figure 1. Its vertices $V = X \cup Z \cup Y$, where $X = \{1-n, \ldots, 0\}$, $Z = \{1, \ldots, p\}$, and $Y = \{p+1, \ldots, p+m\}$, are numbered consistently with respect to dependence, that is, $i \prec^+ j \Rightarrow i < j$. Here, $\prec^+$ denotes the transitive closure of the dependence relation $\prec$. The vertices in X (Z; Y) are referred to as minimal or independent (intermediate; maximal or dependent) vertices.

The objective is to transform the program that implements F into one that computes the Jacobian matrix (or Jacobian)

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

of F with respect to the n inputs for a given argument $\mathbf{x}_0$ such that a minimal number of scalar fused multiply-add floating-point operations (`fmas`) are performed. Once numerical values have been computed for all local partial derivatives, scalar floating-point multiplications and additions are the only arithmetic operations required to accumulate F'. The accumulation of F' can be regarded as an elimination procedure in the c-graph G of F, as introduced in [1]. The original c-graph is transformed into a subgraph of the directed complete bipartite graph $K_{n,m}$ such that the labels on the remaining edges are exactly the nonzero elements of the Jacobian. The result of this transformation applied to the c-graph from Figure 1 is displayed in Figure 2.

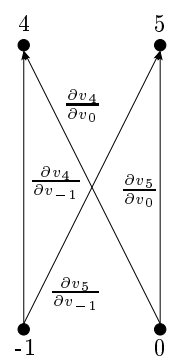

Fig. 2. F'

In this paper we develop heuristics for eliminating vertices and edges such that the overall cost of computing F' is minimized. A detailed discussion of the corresponding theory can be found in [2]. Here, we introduce only a minimal subset of the framework, in order to focus on new heuristics for vertex and edge elimination in c-graphs. The structure of the paper is as follows. In Section 2 we introduce vertex and edge elimination in c-graphs. Various well known and new Markowitz-type heuristics for both vertex and edge elimination are presented in Section 3. Their properties and performance are discussed in Section 4, and numerical results are presented. Conclusions are drawn in Section 5.

2 Jacobians by Vertex and Edge Elimination

The origins of both vertex and edge elimination in c-graphs are in *automatic differentiation* (AD) [3–6]. This technique modifies the semantics of numeri-

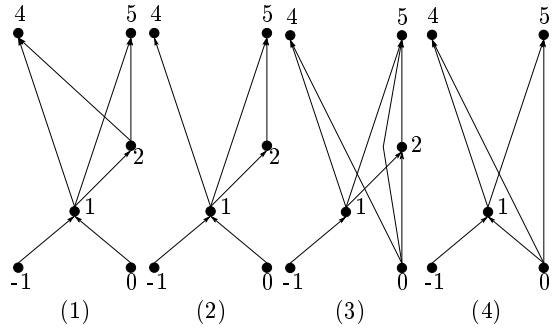

Fig. 3. Vertex and edge elimination

cal programs such that derivatives of the underlying vector function can be computed efficiently with machine accuracy. In contrast with divided difference approximations, AD exploits the chain rule to compute Jacobian times vector products $\dot{\mathbf{y}} = F'\dot{\mathbf{x}}$ in forward mode and transposed Jacobian times vector products $\bar{\mathbf{x}} = (F')^T\bar{\mathbf{y}}$ in reverse mode (see [6, Chapter 3] for details). In particular, the Jacobian itself can be obtained at a cost of $n|E|$ in forward mode and $m|E|$ in reverse mode by letting the vectors $\dot{\mathbf{x}}$ and $\bar{\mathbf{y}}$ range over the Cartesian basis vectors in $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively. The number of edges in the c-graph of $G = (V, E)$ is denoted by $|E|$.

Alternatively, F' can be computed by eliminating all intermediate vertices or edges in G as follows. When eliminating an intermediate vertex j, new edges are introduced connecting the predecessors of j with its successors. A new edge (i, k) is labeled with the product of the labels of (j, k) and (i, j). Parallel edges are merged, and the corresponding edge labels are added. Finally, j is removed together with its incident edges. The elimination of vertex 3 from the c-graph shown in Figure 1 leads to graph (1) in Figure 3. For example, the new label of $(1, 4) \in E$ is equal to $c_{4,1} = c_{4,3}c_{3,1}$. The elimination of vertex 2 in graph (3) leads to graph (4) and, for example, $c_{5,0} = c_{5,0} + c_{5,2}c_{2,0}$. The correctness of the vertex elimination rule is shown in [1]. The number of `fmas` involved in the elimination of j is referred to as the *Markowitz degree* of j, and it is equal to $\mu_j = |\{i : i \prec j\}||\{k : j \prec k\}|$.

The elimination of a vertex j is equivalent to the simultaneous *front elimination* of all edges leading into it. Similarly, the elimination of j is equivalent to the simultaneous *back elimination* of all edges emanating from it. An edge (i, j) is front eliminated by connecting i with all successors of j. For all successors k of j, the new edges (i, k) are labeled with $c_{k,i} = c_{k,j}c_{j,i}$. If (i, k) existed before, then $c_{k,i} = c_{k,i} + c_{k,j}c_{j,i}$. The edge (i, j) is removed after this. The number of `fmas` required to front eliminate (i, j) is equal to $|\{k : j \prec k\}|$. The front elimination of $(0, 1)$ transforms

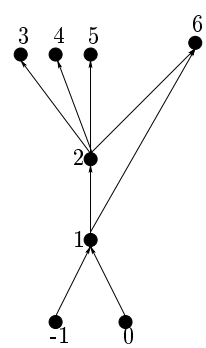

Fig. 4. Lion

graph (2) into graph (3) in Figure 3. Analogously, an edge (j,k) is back eliminated by connecting all predecessors of j with k. For all predecessors i of j the new edge (i,k) is labeled with $c_{k,i} = c_{k,j}c_{j,i}$. Again, the label becomes $c_{k,i} = c_{k,i} + c_{k,j}c_{j,i}$ if (i,k) existed before. Finally, (j,k) is removed. The number of `fmas` required to back eliminate (j,k) is equal to $|\{i : i \prec j\}|$. In Figure 3, graph (2) can be obtained from graph (1) by back elimination of $(2,4)$. Newly generated edges are referred to as fill-in. Absorption takes place whenever two parallel edges are merged.

The set of all valid vertex eliminations is contained within the set of all valid edge eliminations. Hence, the optimal vertex elimination sequence is contained within the set of all edge elimination sequences, and thus the optimal edge elimination sequence performs at most the same number of arithmetic operations as the optimal vertex elimination sequence. The *lion graph* [7] displayed in Figure 4 represents one example where the optimal edge elimination sequence involves fewer operations than does the optimal vertex elimination sequence. Both elimination sequences require $4 + 8 = 12$ `fmas`. The back elimination of $(2,6)$ followed by the elimination of 1 and 2 reduces this number by one. Refer to [2] for a more detailed investigation of this *vertex-edge discrepancy*. It is the motivation for introducing edge elimination in addition to the conceptually much easier vertex elimination method.

We assume the chain rule to be associative over the floating point numbers. In other words, the numerical values of the entries in F' do not depend on the order in which the intermediate vertices or edges are eliminated. However, the computational cost varies. In Figure 1, for example, the vertex elimination sequence $[1,2,3]$ performs $4 + 2 + 4 = 10$ `fmas`. The reader may wish to verify that $[2,1,3]$ takes only $1 + 2 + 4 = 7$ `fmas`. The cost of computing F' by using either the forward or the reverse mode of AD is $2 \cdot 7 = 14$. Even on this small example, the operations count can be reduced by a factor of two. This is the primary motivation for investigating heuristics for vertex and edge elimination in Section 3.

3 Heuristics

Both the vertex and edge elimination problems in c-graphs are conjectured to be NP-complete [8, 1, 7]. No polynomial algorithm is known for solving them exactly. The problem of minimizing the fill-in under vertex elimination was shown to be NP-complete by Herley [9] in an unpublished adaption of a note by Gilbert [10] on a result by Rose and Tarjan [11] about vertex elimination techniques for solving sparse linear systems. So far, it remains unclear whether the same is true for edge and *face* [2] elimination.

In the following we write $H(G) = i$ whenever the application of a heuristic H to a c-graph G gives the vertex i as a result. An analogous notation is used for edge elimination heuristics. All heuristics H are defined such that $|H(G)| = 1$, that is, the result of applying H to G should contain a single vertex or edge.

3.1 Forward Vertex and Edge Elimination

The forward vertex elimination mode FM_v eliminates the intermediate vertices in ascending order with respect to their indices, that is, $\mathrm{FM}_v(G) = j \Leftrightarrow \forall i \in V\colon j \leq i$. The same idea can be applied to edge elimination. For reasons of consistency, we require the forward edge elimination mode FM_e to have the same computational cost as FM_v. This can be achieved by pure back edge elimination sequences in lexicographical order or by pure front edge elimination sequences in switched lexicographical order. For example, the latter can be written as

$$\mathrm{FM}_e(G) = (i,j)_f \quad \Leftrightarrow \quad (i,j) \in E \cap (V \times Z) \ \wedge \ \forall (k,l) \in E \cap (V \times Z)\colon j \leq l \vee j = l \wedge i \leq k.$$

The fact that an edge (i,j) is front (back) eliminated is denoted by $(i,j)_f$ ($(i,j)_b$). Note that the set of edges that can be front eliminated is restricted to those having an intermediate vertex as target.

3.2 Reverse Vertex and Edge Elimination

In reverse vertex elimination mode the intermediate vertices are eliminated in descending order starting with p, that is, $\mathrm{RM}_v(G) = j \Leftrightarrow \forall i \in V\colon j \geq i$. The extension to edge elimination sequences is similar to FM, for example,

$$\mathrm{RM}_e(G) = (i,j)_f \quad \Leftrightarrow \quad (i,j) \in E \cap (Z \times V) \ \wedge \ \forall (k,l) \in E \cap (Z \times V)\colon j \geq l \vee j = l \wedge i \geq k.$$

To ensure uniqueness of the result, one must combine all heuristics with a tiebreaking criterion or even a hierarchy of tiebreakers. We use either FM or RM (for vertices or edges, depending on the context) as the "bottom line," since we always have $|\mathrm{FM}(G)| = 1$ as well as $|\mathrm{RM}(G)| = 1$.

3.3 Lowest Markowitz

The lowest Markowitz (LM) degree-first heuristic was introduced in [1]. It was motivated by a similar idea from the theory of direct methods for solving sparse linear systems.

Heuristic 1 *$LM_v(G) = j$ if*

$$\forall i \in V : \ \mu_j \leq \mu_i \ \ \wedge \ \ \forall i \in V', V' = \{k \in V : \mu_j = \mu_k\}\colon j = RM_v(G') \quad ,$$

where G' is the subgraph of G that is induced by V'.

$G' = (V', E')$ is said to be induced by a vertex set $V' \subseteq V$ if $E' = \{(i,j) : i \in V' \wedge j \in V'\}$. In general, we always have the choice to use either FM_v or RM_v as the ultimate tiebreaker to make the result of a heuristic unique. This choice, however, significantly affects the performance of the heuristic, as illustrated by the c-graphs in Figure 5. As shown in [12], forward vertex elimination is optimal for the graph on the left, whereas reverse vertex elimination minimizes the number of `fmas` required to accumulate the gradient for the graph on the right. The

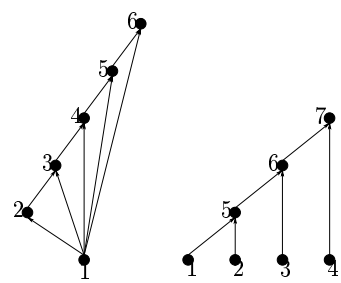

Fig. 5. Example: LM_v tiebreakers

LM_v heuristic finds the optimal vertex elimination sequence not depending on the tiebreaker for the graph on the left. To solve the problem for the graph on the right, however, one must combine it with RM_v.

As for FM_v and RM_v, the computational effort of LM_v is constant per vertex and per iteration. The number of iterations is equal to the number of intermediate vertices p, and the vertices taken into account are the intermediate vertices that have not been eliminated yet. Hence, the maximal overall effort is $p(p+1)/2 \in \mathcal{O}(|V|^2)$.

The Markowitz degree of $(i,j) \in E$ is defined as $\min(|\{k\colon j \prec k\}|, |\{k\colon k \prec i\}|)$. The edge (i,j) is front eliminated if $|\{k\colon j \prec k\}| \leq |\{k\colon k \prec i\}|$. Otherwise, it is back eliminated. The lowest-Markowitz heuristic for edge elimination LM_e is defined analogous to Heuristic 1. Its complexity, however, also depends on the fill-in that is generated. Although, the termination of edge elimination was shown in [7], it remains unclear whether the cost of edge elimination sequences is still polynomial in the worst case. This question is the subject of ongoing research.

3.4 Lowest Relative Markowitz

The lowest relative Markowitz (LRM) degree-first heuristic is an extension of LM. It was introduced in [13]. Let $\iota_j = |\{k\colon k \in X \wedge k \prec^+ j\}|$, $\delta_j = |\{k\colon k \in Y \wedge j \prec^+ k\}|$, and $\hat{\mu}_j = \mu_j - \iota_j \cdot \delta_j$. The relative Markowitz degree is defined as the difference between the Markowitz degree and the *dependence degree* $\iota_j \cdot \delta_j$ of the vertex j. The idea is to maximize the dependence degree while, at the same time, minimizing μ_j. This heuristic is usually combined with LM_v as tiebreaker.

Heuristic 2 *$LRM_v(G) = j$ if*

$$\forall i \in V \;:\; \hat{\mu}_j \leq \hat{\mu}_i \;\;\wedge\;\; \forall i \in V', V' = \{k \in V\colon \hat{\mu}_j = \hat{\mu}_k\}\colon j = LM_v(G') \quad ,$$

where G' is the subgraph of G that is induced by V'.

See [13] for further details.

To formulate LRM for edge elimination, we set

$$\hat{\mu}((i,j)) = \min(|\{k : j \prec k\}| - \delta_j, |\{k : k \prec i\}| - \iota_i).$$

As for LM_e, an edge (i,j) is front eliminated if $|\{k : j \prec k\}| - \delta_j \leq |\{k : k \prec i\}| - \iota_i$ and else back eliminated. Thus, LRM_e can be derived immediately from Heuristic 2.

3.5 Maximal Overall Markowitz Degree Reduction

The maximal overall markowitz degree reduction (MOMR) heuristic represents the global pendant to LM. It considers the effect of the elimination of a vertex on the Markowitz degrees of its neighbors instead of its own Markowitz degree. Therefore we define the overall Markowitz degree of a c-graph $G = (V, E)$ as $M = M(G) = \sum_{i \in V} \mu_i$. The overall Markowitz degree reduction of a vertex $i \in V$ is defined as $\mu_j^- = M(G) - M(G - j)$, where $G - j$ is the c-graph after the elimination of j.

Heuristic 3 *$MOMR_v(G) = j$ if*

$$\forall i \in V : \ \mu_i^- \leq \mu_j^- \quad \wedge \quad \forall i \in V', V' = \{k \in V : \mu_j^- = \mu_k^-\} : j = LM_v(G') \quad ,$$

where G' is the subgraph of G that is induced by V'.

Initially, MOMR_v attempts to reduce the Markowitz degrees of its neighbors as much as possible, thus making use of absorption while trying to avoid excessive fill-in. Only as a second step does it take the vertex's own Markowitz degree into account. MOMR_v reduces the cost of eliminating the remaining vertices as much as possible.

To calculate the Markowitz degree reduction of k caused by the elimination of some $j \prec k$, one must compute the difference of their respective predecessor sets. Similarly, the successor sets must be considered if $k \prec j$. Hence, MOMR_v involves at most $\mathcal{O}(a^2 \, |V|^2)$ operations, where a is the average number of predecessors (or successors) per vertex over all elimination steps.

The formulation of MOMR for edge elimination is straightforward by comparing the overall Markowitz degree of G before and after the front and back elimination of an edge.

3.6 Lowest Markowitz Minimal Damage

This lowest Markowitz minimal damage (LMMD) heuristic combines LM and MOMR. The effect of MOMR is twofold. On one hand, it enforces the elimination of vertices that reduce the Markowitz degrees of their neighbors maximally (or at least do not increase them too much). On the other hand, it implicitly prefers vertices with high Markowitz degrees, and hence incurs a high elimination cost.

The idea behind LMMD is to look for vertices with low Markowitz degrees that cause a minimal increase of the Markowitz degree on other vertices (minimal damage). If the overall Markowitz degree without considering $j \in V$ of a c-graph $G = (V, E)$ is defined as $\hat{M}_j(G) = \sum_{i \neq j} \mu_i$, then the damage caused by the elimination of j is set to $d_j = M(G - j) - \hat{M}_j(G)$. In order to increase the flexibility of LMMD, the damage is scaled with a weight w.

Heuristic 4 *$LMMD_v(w,G) = j$ if*

$$\forall i \in V : \ \mu_j + wd_j \leq \mu_i + wd_i \ \wedge \ \forall i \in V', V' = \{k \in V : \mu_j + wd_j = \mu_k + wd_k\} : j = RM_v(V').$$

W.l.o.g., we choose RM_v as a tiebreaker. Other criteria can be used as well; however, especially LM_v as tiebreaker would be somewhat redundant because of the already present implicit orientation toward the lowest Markowitz degree. Choosing a small factor w focuses on the Markowitz degree of the vertex and reduces the value of the damage to a tiebreaker. Large values, on the other hand, emphasize the Markowitz degree reduction of the neighboring vertices and degrade the current degree to a secondary criterion. The computational effort is identical to that of MOMR_v.

Again, the formulation of LMMD_e is analogous to Heuristic 4, when considering the damage caused by front and back edge elimination, respectively.

4 Numerical Results

The generation of c-graphs from real-world application programs requires a fully functional compiler front-end to transform the program into an abstract syntax tree. We are using the tool EliAD [14] to perform this task.

Table 1. Elimination Costs for Graphs of Real-World Programs

	n	p	m	FM	RM	LM1	LM2	LRM1	LRM2	MOMR1	MOMR2	LMMD
ROE	10	208	5	1977	1341	1035	962	1073	1036	1215	1130	1023
				1977	1341	1134	1021	1800	1260	1396	1384	1324
CTS	134	1386	252	3402	2268	2898	2268	3390	3385	2268	2268	2268
				3402	2268	3402	2268	2307	2306	2268	2268	2268
CPF	12	56	11	168	98	120	98	120	119	96	96	96
				168	98	168	98	97	99	96	96	96
HHD	8	82	8	308	220	258	212	286	281	222	222	212
				308	220	326	224	223	223	216	216	216
FIC	128	1230	34	4003	1860	2571	1860	3220	2982	1860	1860	1860
				4003	1860	4003	1860	1866	1862	1860	1860	1860

Table 1 lists the results obtained by applying various heuristics to graphs of the following applications: Roe's numerical flux (ROE) [15], coating thickness standardization (CTS), combustion of propane, full formulation, (CPF), human heart dipole (HHD), and Flow in a Channel (FIC). CTS, HHD, and FIC have been generated from the corresponding MINPACK-2 test problems [16] by unrolling all loops for a given input. We list the number of minimal (n), intermediate (p), and maximal (m) vertices. The two lines per c-graph list the results for vertex (top) and edge elimination (bottom). The two columns for LM, LRM,

and MOMR refer to FM and RM as tie-breakers, respectively. For MOMR, FM or RM are secondary criteria for the primary tie-breaker LM. LMMD is used with $w = 1$ and tie-breaker RM.

Ideally, one expects good heuristics for edge elimination to solve the reference problems for which edge elimination is known to be superior compared with vertex elimination. Examples are the lion and the bat graphs whose Jacobians can be accumulated by using 11 and 23 multiplications, respectively [2]. The straight-forward application of ideas from vertex elimination heuristics (Markowitz-degree-related in our case) does not exhibit the desired effect, as shown in Table 2. For the larger problems in Table 1 the values obtained by this method turn out to be even worse than the ones resulting from vertex elimination.

Table 2. Elimination Costs for Lion and Bat Graphs

	n	p	m	FM	RM	LM1	LM2	LRM1	LRM2	MOMR1	MOMR2	LMMD
LION	2	2	4	12	12	12	12	12	12	12	12	12
				12	12	12	12	12	12	14	14	12
BAT	4	3	4	24	24	24	24	24	24	24	24	24
				24	24	24	24	24	24	26	27	26

At the statement and basic block level, c-graphs can be generated at compile time. These graphs are usually small, an advantage for studying the effect that heuristics for vertex or edge elimination have on the runtime of the compilation. However, small graphs are not very useful for studying the effect of heuristics on the number of `fmas` that are required to accumulate the corresponding Jacobian. Therefore, we have developed a random c-graph generator. It allows us to check the behavior of various heuristics on a large number of c-graphs of varying sizes.

4.1 Random Structured C-Graph Generator

Since c-graphs are not arbitrary but are structured in some sense, our c-graph generator works hierarchically in two or three phases. First, a set of random statements is built. These statements are *single expression use* programs as discussed in [12]. They consist of a given number of inputs, a given number of intermediate vertices that represents subexpressions, and a single output. Inputs may have several successors, and all intermediate vertices have exactly one successor. In most high-level programming languages, the elementary functions are either unary or binary. In our c-graph generator, their distribution can be specified. For all intermediate vertices and for the output, the number of predecessors is chosen with respect to this distribution followed by the actual predecessors themselves. If unused inputs remain, then the number of binary operations is increased automatically.

An ordered sequence of statements can be composed into a basic block with a given number of inputs and outputs. The composition guarantees that for every input there exists at least one path to some output and vice versa. Furthermore, the resulting c-graph is acyclic.

Blocks (or statements) can be wrapped into loops by concatenation of multiple copies. The outputs of one iteration become the inputs of the next. If the number of inputs n is greater than the number of outputs m, then the latter get mapped onto the first m inputs for the next iteration. The remaining inputs become global inputs. A similar strategy is employed for $n < m$, and the number of global outputs is increased.

4.2 Discussion of Further Results

In Table 3 we have listed the results obtained by applying the heuristics discussed in this paper to a number of randomly generated c-graphs. Here, `iters` is the number of executions of an assumed loop around the code leading to the original (`iters` = 1) c-graph.

We observe that MOMR and LMMD show the most consistent behavior, while LMMD delivers the best results in most cases. RM outperforms FM if there are fewer outputs than inputs. FM delivers better results if the number of outputs exceeds the number of inputs. If there are as many outputs as there are inputs, then RM seems to perform better. A likely reason is that for the intermediate vertices the number of successors is arbitrary, whereas the number of predecessors is limited by two. LM appears to be a reasonable compromise for a large number of c-graphs. Typically, it performs almost as well as the best heuristic.

An important goal of the research that led to this paper was the investigation of Markowitz degree-based heuristics for edge elimination. We expected our results to be improved by simply reformulating the ideas behind the vertex elimination heuristics in the context of edge elimination. The opposite appears to be the case. Apart from the last two examples, none of the edge elimination heuristics delivers better results than the best vertex elimination heuristic. Apparently, the heuristics are influenced negatively by the considerably widened search space. We conclude that the implicit locality of eliminating all edges incident to a vertex is usually more beneficial than the higher degree of freedom when considering the heuristics that are discussed here. A good example is the seventh c-graph, where LRM_e combined with forward mode as a tiebreaker is six times as expensive as the best heuristic (LMMD_e).

In Table 4 we have applied different damage weights and tiebreaking criteria to LMMD for both vertex and edge elimination. For vertex elimination, only the results obtained with RM_v as tiebreaker are listed; however, the results obtained with FM_v are similar. The five examined weights w were 1.0, 0.8, 0.6, 2.0, and 4.0. LMMD behaves more like LM with $w < 1$ and more like MOMR with $w > 1$. The application of LMMD to edge elimination was influenced strongly by the tiebreaking criterion. Therefore, the forward mode (LMMD1, LMMD3, and LMMD5) was compared with the reverse mode (LMMD2, LMMD4, and

Table 3. Elimination Costs for Graphs from Structured Random C-Graph Generator

n	p	m	iters	FM	RM	LM1	LM2	LRM1	LRM2	MOMR1	MOMR2	LMMD
3	64	2	1	108	87	89	77	93	86	83	81	77
				108	87	104	87	106	86	83	81	78
3	206	3	1	638	418	374	316	392	389	337	342	312
				638	418	553	351	448	417	333	336	334
10	236	10	1	1216	609	480	425	520	499	434	441	417
				1216	609	666	462	507	467	448	454	447
10	482	10	2	3945	2486	1156	1069	1385	1345	1186	1109	1051
				3945	2486	1613	1293	2255	1701	1258	1230	1199
10	1220	10	5	13050	10653	3513	3591	3685	3573	3904	3705	3340
				13050	10653	4983	5220	8141	6515	5001	4509	4582
10	243	3	1	1567	435	468	369	448	430	411	406	367
				1567	435	837	392	813	418	428	398	380
38	1227	3	5	31182	4620	2736	2285	2708	2540	2614	2430	2129
				31182	4620	4806	2422	12610	3902	2942	2483	2780
10	245	1	1	1605	349	454	326	461	430	356	357	325
				1605	349	801	349	614	352	339	339	332
46	1229	1	5	26761	1749	2346	1634	2330	1866	1784	1789	1629
				26761	1749	4549	1749	8188	1764	1699	1699	1664
3	236	10	1	718	649	463	420	526	519	435	462	415
				718	649	646	460	669	488	440	459	453
3	1192	38	5	3906	14233	2662	2525	2786	2778	2581	2648	2294
				3906	14233	3436	3272	4988	3910	2571	3316	2865
1	236	10	1	337	651	423	373	444	432	342	394	356
				337	651	337	404	742	426	334	368	352
1	1184	46	5	1725	4249	2155	1941	2227	2172	1742	2006	1840
				1725	4249	1725	2564	1818	2099	1702	1864	1788

LMMD6). On the other hand, only three weights were used: 1.0 (LMMD1, LMMD2), 0.8 (LMMD3, LMMD4), and 2.0 (LMMD5, LMMD6).

Table 4. Parameter Study of LMMD

n	p	m	iters	LMMD1	LMMD2	LMMD3	LMMD4	LMMD5	LMMD6
3	206	3	1	312	312	312	313	314	
				340	334	552	344	341	323
10	236	10	1	417	417	418	421	421	
				447	447	665	460	446	447
10	1220	10	5	3340	3323	3315	3347	3384	
				5386	4582	6149	5541	4710	4341
38	1227	3	5	2129	2131	2193	2147	2196	
				3139	2780	4872	2912	3125	2926
3	1192	38	5	2294	2309	2400	2354	2388	
				2934	2865	3748	3358	2719	2901
1	1184	46	5	1840	1847	1866	1818	1803	
				1636	1788	1636	2534	1636	1788

One observes that LMMD_v is not very sensitive with respect to w. The performance of LMMD_e, however, can vary strongly for different weights. There does not seem to be a preferable value for w. Furthermore, the tiebreaking criterion influences the elimination cost even more. RM_e is usually the better choice. Again, the most interesting observation is that in most cases LMMD_v yields better results than LMMD_e. The discrepancy is often quite large.

5 Conclusion

Both MOMR and LMMD represent good choices for Markowitz-based vertex elimination heuristics. On a representative test set (a subset of which was presented here) they exhibit a consistent behavior in terms of the quality of the elimination sequences that were generated. In particular, they exhibit an increased robustness with respect to the choice of different tiebreaking criteria. Similar ideas were applied to edge elimination with no noticeable improvement in the cost of the elimination sequences generated. We conclude that different criteria must be developed in order to exploit the power of edge elimination for accumulating Jacobians. How edge elimination sequences can reduce the cost of vertex elimination sequences is the subject of ongoing research.

Similar to the approach taken in [17], we are implementing logarithmic simulated annealing algorithms [18] for edge elimination. We hope to observe discrepancies between the optimal vertex and edge elimination sequences for real-world applications. From the structure of such problems we expect to learn more about

suitable criteria for edge elimination heuristics. Furthermore, a detailed investigation of the energy landscape of the various combinatorial optimization problems arising in Jacobian computation will allow us to gain insight into potential improvements. Finally, we conclude that, in view of our promising results, robust heuristics for accumulating Jacobians efficiently should become a key feature of software tools for automatic differentiation.

Acknowledgments

This research is supported by the UK's Engineering and Physical Sciences Research Council under grant GR/R/38101/01.

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

References

1. Griewank, A., Reese, S.: On the calculation of Jacobian matrices by the Markovitz rule. In: [5]. (1991) 126–135
2. Naumann, U.: Optimal accumulation of Jacobians by elimination methods on the dual computational graph. Mathematical Programming (2002) to appear.
3. Berz, M., Bischof, C., Corliss, G., Griewank, A., eds.: Computational Differentiation: Techniques, Applications, and Tools, Philadelphia, SIAM (1996)
4. Corliss, G., Faure, C., Griewank, A., Hascoet, L., Naumann, U., eds.: Automatic Differentiation of Algorithms – from Simulation to Optimization, New York, Springer (2002)
5. Corliss, G., Griewank, A., eds.: Automatic Differentiation: Theory, Implementation, and Application, Philadelphia, SIAM (1991)
6. Griewank, A.: Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation. Number 19 in Frontiers in Applied Mathematics. SIAM, Philadelphia (2000)
7. Naumann, U.: Efficient Calculation of Jacobian Matrices by Optimized Application of the Chain Rule to Computational Graphs. PhD thesis, Technical University Dresden (1999)
8. Bischof, C., Haghighat, M.: Hierarchical approaches to automatic differentiation. In: [3]. (1996) 82–94
9. Herley, K.: A note on the NP-completeness of optimum Jacobian accumulation by vertex elimination. Presentation at: Theory Institute on Combinatorial Challenges in Computational Differentiation (1993)
10. Gilbert, J.: A note on the NP-completeness of vertex elimination on directed graphs. J. Alg. Disc. Meth. **1** (1980) 292–294
11. Rose, D., Tarjan, R.: Algorithmic aspects of vertex elimination on directed graphs. J. Appl. Math. **34** (1978) 176–197
12. Naumann, U.: On optimal Jacobian accumulation for single expression use programs. Preprint ANL-MCS/P944-0402, Argonne National Laboratory (2002)
13. Naumann, U.: An enhanced Markowitz rule for accumulating Jacobians efficiently. In Mikula, K., ed.: ALGORITHMY'2000 Conference on Scientific Computing, Slovak University of Technology, Bratislava, Slovakia (2000) 320–329

14. Tadjouddine, M., Forth, S., Pryce, J., Reid, J.: Performance issues for vertex elimination methods in computing Jacobians using Automatic Differentiation. In: Proceedings of the ICCS 2000 Conference. Volume 2330 of Springer LNCS. (2002) 1077–1086
15. Roe, P.: Approximating Riemann solvers, parameter vectors, and difference schemes. J. Comp. Physics (1981) 357–372
16. Averik, B., Carter, R., Moré, J.: The MINPACK-2 test problem collection (preliminary version). Technical Report 150, Mathematical and Computer Science Division, Argonne National Laboratory (1991)
17. Naumann, U., Gottschling, P.: Prospects for simulated annealing in automatic differentiation. In Steinhöfel, K., ed.: SAGA 2002 - Stochastic Algorithms, Foundations and Applications. Volume 2264 of LNCS., Springer, Berlin (2001) 131–144
18. Albrecht, A., Wong, C.: On logarithmic simulated annealing. In van Leeuwen, J., Watanabe, O., Hagiya, M., Mosses, P., eds.: Proc. IFIP International Conference on Theoretical Computer Science. LNCS, Springer (2000)