Optimization Methods and Software
Vol. 00, No. 00, January 2009, 1–11

RESEARCH ARTICLE

Randomized Heuristics for Exploiting Jacobian Scarcity

Andrew Lyons[ab*] , Ilya Safro[b] and Jean Utke[ab]

[a] *Computation Institute, The University of Chicago*;
[b] *Mathematics and Computer Science Division, Argonne National Laboratory*

(*v4.4 released October 2008*)

We describe a code transformation technique that, given code for a vector function F, produces code suitable for computing collections of Jacobian-vector products $F'(\mathbf{x})\dot{\mathbf{x}}$ or Jacobian-transpose-vector products $F'(\mathbf{x})^T\bar{\mathbf{y}}$. Exploitation of scarcity - a measure of the degrees of freedom in the Jacobian matrix - means solving a combinatorial optimization problem that is believed to be hard. Our heuristics transform the computational graph for F, producing, in the form of a transformed graph G', a representation of the Jacobian $F'(\mathbf{x})$ that is both concise and suitable for evaluating large collections of Jacobian-vector products or Jacobian-transpose-vector products. Our heuristics are randomized and compare favorably in all cases with the best known heuristics.

Keywords: automatic differentiation; scarcity; preaccumulation; edge elimination

AMS Subject Classification: 90C59; 68W20; 05C81; 68N20

1. Introduction

The computation of Jacobian-vector products is a fundamental step in the context of science and engineering applications. Without loss of generality, suppose a vector function $F : \mathbb{R}^n \to \mathbb{R}^m$ is given as a straight-line evaluation procedure; real-life application codes often comprise such straight-line procedures. Thus, F may not represent the entire function of interest but rather a small part that is executed many times. We are interested in algorithmically applying the chain rule, a technique known as automatic differentiation (AD), in order to obtain a new program that evaluates $\mathbf{y} = F(\mathbf{x})$ along with some derivative information for F. Suppose, in particular, that we are interested in computing either a collection of p Jacobian-vector products $\left(F'(\mathbf{x})\dot{\mathbf{x}}^i\right)_{i=1,\dots,p}$ or a collection of p Jacobian-transpose-vector products $\left(F'(\mathbf{x})^T\bar{\mathbf{y}}^i\right)_{i=1,\dots,p}$, where p is assumed to be sufficiently large. In this context, the vectors $\dot{\mathbf{x}}^i$ are directions in the domain and the vectors $\bar{\mathbf{y}}^i$ may be interpreted as weights. Because of symmetry, we can restrict our attention to the former without loss of generality. Our goal, therefore, will be to approximate the most efficient program for computing collections of Jacobian-vector products. The notion of Jacobian scarcity [1–3] generalizes the properties of sparsity and rank to capture a deficiency in the degrees of freedom of the Jacobian matrix. We describe new randomized heuristics that exploit scarcity for the optimized evaluation of collections of Jacobian-vector or Jacobian-transpose-vector products.

[*] Corresponding author. Email: lyonsam@gmail.com

ISSN: 1055-6788 print/ISSN 1029-4937 online

DOI: 10.1080/1055678xxxxxxxxxxxxx
http://www.informaworld.com

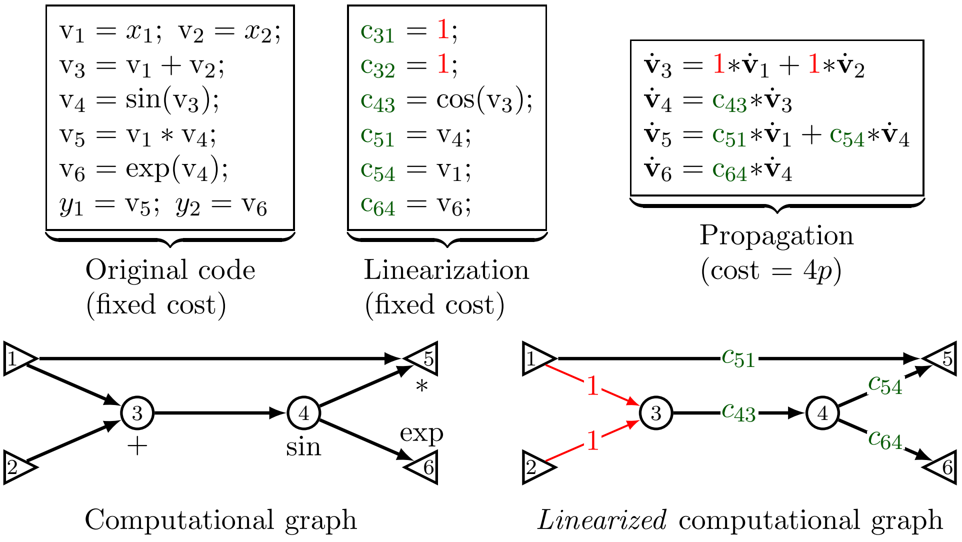
In the remainder of this section, we introduce the necessary definitions and concepts. Section 2 contains a description of our heuristics and their implementation. The results of our computatioal experiments are discussed in Section 3. In Section 4 we offer some conclusions and suggest possible directions for future work.

1.1. *Propagating derivatives*

We consider an implementation of F to consist of a sequence of assignments to program variables of the form

$$\mathrm{v}_j = \phi(\mathrm{v}_i)_{i \prec j} \;,$$

where the relation $i \prec j$ indicates a direct dependence of variable v_j on variable v_i and induces a directed acyclic graph (DAG) G called the *computational graph* of F. Furthermore, it is assumed that each function ϕ is differentiable with respect to each of its arguments so that we may *linearize* F (and thus G) by adding code that evaluates the *local partial derivatives* $c_{ji} \equiv \partial \mathrm{v}_j / \partial \mathrm{v}_i$. This process is implemented as a fully mechanical procedure in such a way that the local partial derivatives are evaluated at a fixed cost that is typically a small constant. These concepts are illustrated by example in Figure 1.



$\mathrm{v}_1 = x_1;\ \mathrm{v}_2 = x_2;$
$\mathrm{v}_3 = \mathrm{v}_1 + \mathrm{v}_2;$
$\mathrm{v}_4 = \sin(\mathrm{v}_3);$
$\mathrm{v}_5 = \mathrm{v}_1 * \mathrm{v}_4;$
$\mathrm{v}_6 = \exp(\mathrm{v}_4);$
$y_1 = \mathrm{v}_5;\ y_2 = \mathrm{v}_6$

Original code
(fixed cost)

$\mathrm{c}_{31} = 1;$
$\mathrm{c}_{32} = 1;$
$\mathrm{c}_{43} = \cos(\mathrm{v}_3);$
$\mathrm{c}_{51} = \mathrm{v}_4;$
$\mathrm{c}_{54} = \mathrm{v}_1;$
$\mathrm{c}_{64} = \mathrm{v}_6;$

Linearization
(fixed cost)

$\dot{\mathbf{v}}_3 = 1*\dot{\mathbf{v}}_1 + 1*\dot{\mathbf{v}}_2$
$\dot{\mathbf{v}}_4 = \mathrm{c}_{43}*\dot{\mathbf{v}}_3$
$\dot{\mathbf{v}}_5 = \mathrm{c}_{51}*\dot{\mathbf{v}}_1 + \mathrm{c}_{54}*\dot{\mathbf{v}}_4$
$\dot{\mathbf{v}}_6 = \mathrm{c}_{64}*\dot{\mathbf{v}}_4$

Propagation
(cost = $4p$)

Computational graph

Linearized computational graph

Figure 1. Suppose we are given a straight-line program (top left) that evaluates the vector function $F\colon \mathbf{x} \mapsto \mathbf{y}$ defined as $y_1 = x_1 \sin(x_1 + x_2)$, $y_2 = e^{\sin(x_1+x_2)}$. The process of *linearization*, fundamental to AD, automatically produces code for evaluating the local partial derivatives c_{ji} at a small fixed cost.

We henceforth assume that the linearized computational graph G is given, where every edge (i, j) is associated with either a unique variable representing the local partial derivative c_{ji} or a value in $\{1, -1\}$. Traditional AD [3] prescribes the *forward mode* for evaluating Jacobian-vector products, whereby derivative values $\dot{\mathbf{v}}_j$ are *propagated* through G from the sources to the sinks by traversing the vertices in topological order. When a vertex j is visited, we compute

$$\dot{\mathbf{v}}_j = \sum_{i \prec j} c_{ji} * \dot{\mathbf{v}}_i \;.$$

$F'(\mathbf{x})\dot{X}$ can be evaluated either by propagating p directions $\dot{\mathbf{x}}^1, \ldots, \dot{\mathbf{x}}^p$ separately (scalar mode), or by propagating $\dot{X} \in \mathbb{R}^{n \times p}$ in a single pass (vector mode).

In terms of scalar multiplications, the total computational cost associated with each edge (i, j) is p if $c_{ji} \notin \{1, -1\}$ and 0 otherwise; this highlights the special significance of unit edges. Thus, in both scalar and vector modes, the cost of evaluating p Jacobian-vector products is $p|E^+(G)|$ scalar multiplications, where $E^+(G) \equiv \{(i,j) \in E \mid c_{ji} \notin \{1,-1\}\}$ denotes the set of *nonunit edges* in G.

1.2. *Graph transformations and preaccumulation*

The essential property of G is that the entries of the Jacobian $F'(\mathbf{x})$ can be expressed as *Baur's formula*

$$\frac{\partial y_j}{\partial x_i} = \sum_{P \in [x_i \rightsquigarrow y_j]} \prod_{(k,\ell) \in P} c_{\ell k} \quad ,$$

where $[x_i \rightsquigarrow y_j]$ denotes the set of all paths from x_i to y_j in G. A sequence ρ of local graph transformations called *edge eliminations* allow us to transform G into the *remainder graph* $G'(\rho)$, which retains this property. Note that we do not consider other known types of transformations (normalizations, reroutings, etc. [7]) because of the caveats associated with them, see also Section 4.1.

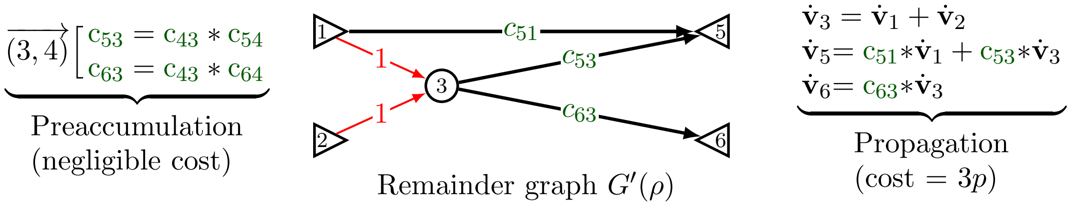

$\overrightarrow{(3,4)} \begin{bmatrix} c_{53} = c_{43} * c_{54} \\ c_{63} = c_{43} * c_{64} \end{bmatrix}$
Preaccumulation
(negligible cost)
1
2
3
5
6
c_{51}
c_{53}
c_{63}
1
1
Remainder graph $G'(\rho)$
$\dot{\mathbf{v}}_3 = \dot{\mathbf{v}}_1 + \dot{\mathbf{v}}_2$
$\dot{\mathbf{v}}_5 = c_{51} * \dot{\mathbf{v}}_1 + c_{53} * \dot{\mathbf{v}}_3$
$\dot{\mathbf{v}}_6 = c_{63} * \dot{\mathbf{v}}_3$
Propagation
(cost $= 3p$)

Figure 2. Result of applying the (partial) edge elimination sequence $\rho = \left(\overrightarrow{(3,4)}\right)$ to the example shown in Figure 1. The cost of the preaccumulation is assumed to be negligible, as it is independent of p.

Front elimination of an edge (i, j), denoted $\overrightarrow{(i,j)}$, entails updating $c_{\ell i} {+}{=} c_{ji} * c_{\ell j}$ for all successors ℓ of j. Similarly, *back elimination* of an edge (i, j), denoted $\overleftarrow{(i,j)}$, entails updating $c_{jk} {+}{=} c_{ik} * c_{ji}$ for all predecessors k of i. If the elimination of an edge leaves an intermediate vertex with either no inedges or no outedges, then the vertex and all incident edges are removed from the graph. An edge elimination sequence ρ is *full* if $G'(\rho)$ contains no intermediate vertices. A full edge elimination sequence corresponds to fully accumulating $F'(\mathbf{x})$ as a matrix. Any edge elimination sequence that is not full is called *partial*.

For functions with scarce Jacobians, judicious choice of an edge elimination sequence can yield a remainder graph that has significantly fewer nonunit edges than G. Proof of this concept is given in Figure 2. Propagation can then be performed at a cost of $p|E^+(G'(\rho))| < p|E^+(G)|$ scalar multiplications. We distinguish between the propagation phase and the *preaccumulation* phase, which consists solely of edge eliminations and has a cost independent of p. When p is sufficiently large, the cost of the propagation phase dominates the computation. With this as our motivation, we focus on finding a sequence ρ of edge eliminations such that $|E^+(G'(\rho))|$ is minimized.

2. Randomized heuristics

A simple variant of the greedy heuristics described in [7] (hereafter called H_g) exploits scarcity by greedily choosing the best possible edge elimination at each step, while maintaining a record of the best nontrivial edgecount that has been obtained. Here, we describe our experiments with two basic types of randomized local search methods, *Metropolis* [8] and *simulated annealing* (SA) [5]. Randomized local search methods are especially useful for hard combinatorial optimization problems about which little is known; successful use cases include problems such as TSP [13] VLSI design [15] and vertex elimination in AD [11, 12]. Our new randomized heuristics are compared with H_g when applied to the same set of examples, which include both sample codes derived from real-world applications and artificially genearted DAGs. Among the heuristics we tested, a hybrid version of Metropolis produced the best results.

2.1. *The edge elimination metagraph*

Consider a directed, Markov chain-based dynamic *metagraph* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ of all possible states G attains as it undergoes sequences of edge eliminations and their backtrackings along with a random walk process on $\mathbf{G}$. Each node $i \in \mathbf{V}$ corresponds to some state of G after a sequence of edge eliminations. The set $\mathbf{E}$ of directed edges is partitioned into sets $\mathbf{E}_s$ and $\mathbf{E}_d$ for static and dynamic directed edges, respectively. A static directed edge $ij \in \mathbf{E}_s$ corresponds to the legal edge elimination that produces state j from i. A dynamic directed edge $ij \in \mathbf{E}_d$ represents a backward step (or backtracking) that is not an edge elimination. At any moment $\mathbf{E}_d$ will contain only one edge. In other words, if at the kth step of a random walk, elimination ij was accepted, then at the $(k+1)$th step $ji \in \mathbf{E}_d$ will appear but the previous backward edge will be removed. Note that, theoretically, at any state j many backward edges could be created since j can be reachable from more than one state. However, introduction of all backward edges can significantly increase the complexity of traversing the metagraph, creating significant implementation difficulties. Thus, at any state (except the initial one) there will be only one backward edge. We denote by b_i the predecessor of i in the random walk over $\mathbf{G}$.

Let G_i denote the DAG corresponding to state i, $c(i)$ denote $|E^+(G_i)|$, and $N_i = N_i^+ \cup N_i^- \cup N_i^0$ denote the set of neighbors of i in $\mathbf{G}$, where

$$\begin{aligned} N_i^+ &= \{j \in \mathbf{V} \mid ij \in \mathbf{E} \text{ and } c(j) > c(i)\} \ ; \\ N_i^- &= \{j \in \mathbf{V} \mid ij \in \mathbf{E} \text{ and } c(j) < c(i)\} \ ; \\ N_i^0 &= \{j \in \mathbf{V} \mid ij \in \mathbf{E} \text{ and } c(j) = c(i)\} \ . \end{aligned}$$

2.2. *The heuristics*

We describe the classical version of Metropolis heuristic H_M in Algorithm 1; we began our computational experiments with this heuristic.

The difference between classical Metropolis and SA lies in the choice of a temperature factor T. Instead of choosing a fixed T, a graduate cooling scheme for T is employed in SA. Carefully chosen (fixed and varying) schemes for T is a central issue of these algorithms. We refer the reader to [5] for a comprehensive background on these methods and to [11, 12] for example of using SA in automatic differentiation elimination problems. As discussed in [4], a Metropolis algorithm with the best temperature can outperform SA. The third heuristic H_h that we used, described in Algorithm 2, is a hybrid of Metropolis and a regular random walk.

Algorithm 1: Classical Metropolis algorithm (H_M)

```
input: initial graph $G_0$
$i \leftarrow 0$
for $k = 1, 2, \ldots$ do
    while $c(i)$ is sufficiently large do
        choose a random edge elimination $ij$
        if $c(j) \leq c(i)$ then
            accept $ij$
        else
            accept $ij$ with probability $e^{-(c(j)-c(i))/T}$ for fixed $T$
        if $ij$ is accepted then
            $i \leftarrow j$
```

Algorithm 2: Hybrid algorithm (H_h)

```
input: initial graph $G_0$, maximum number of steps $K$
$i \leftarrow 0$
for $k = 1, 2, \ldots, K$ do
    if $k$ is sufficiently large then
        $i \leftarrow 0$
    list all possible eliminations $ij$
    if $|N_i^- \setminus b_i| > 0$ then
        accept $j \in N_i^+$ with normalized probability $p_{ij} \propto e^{-(c(j)-c(i))/T}$
    else
        accept $j \in N_i^+ \cup N_i^0 \cup \{b_i\}$ with normalized probability
        $p_{ij} \propto e^{(-(c(j)-c(i))+1)/T}$
    if $ij$ is accepted then
        $i \leftarrow j$
```

3. Computational results

In this section, we describe numerical results obtained using Algorithm 2, which has been implemented as part of OpenAD [14]. We designed our numerical experiments with three types of computational graphs: (a) examples derived from applications; (b) a set of artificially generated single-expression-use (SEU) graphs [9]; and (c) random DAGs. We began with a series of aggressive random walks over $\mathbf{G}$ on SEU graph instances and randomly generated instances. After sufficiently many steps, the random walk is restarted, while keeping track of the best result achieved so far. Surprisingly, this trivial algorithm resulted in an improvement of 5-10% over H_g; This provided the first indication that randomized local search can improve on H_g. In the real-life examples, however, this strategy was not able to beat H_g. The next stage of experiments consisted of designing the classical versions of Metropolis and SA. Independent of the aggressiveness of the gradual cooling scheme for T, both methods provided an improvement of up to 20% on the real-life instances and 10-15% on the artificial instances. The distribution of the results was proportional to the Gaussian which concentrated the most likely improvement on 12% over H_g on real-life instances. This series of experiments gave us an important observation regarding the steps that improve the current state i: *if there exist two eliminations ij and ik such that $ij, jk \in N_i^-$ and $c(j)$ and $c(k)$*

are almost equal, the better of the two should not necessarily be accepted. This observation guided the design of H_h which is the most successful of the heuristics. Note this issue cannot be addressed by any classical gradual cooling scheme, as such schemes work only on the elements of N_i^+. Thus, no significant difference was observed between Metropolis and SA when different schemes were employed.

Our best computational results were obtained using Algorithm 2 (H_h). The observed improvement on the real-life graphs was up to 35%. Examples of two experimental series for real-life graphs are presented in Figure 3. For every graph we ran 800 experiments. The results of H_g are 185 and 186 for the first and second graphs, respectively. The most interesting example can be observed in Figure 3(a), in which one can see that H_h almost separates two clusters of the solution quality. The maximum number of steps (K) (see Algorithm 2) was $20|V|$ with 5 restarts (line 3) when k was reaching $4|V|$.

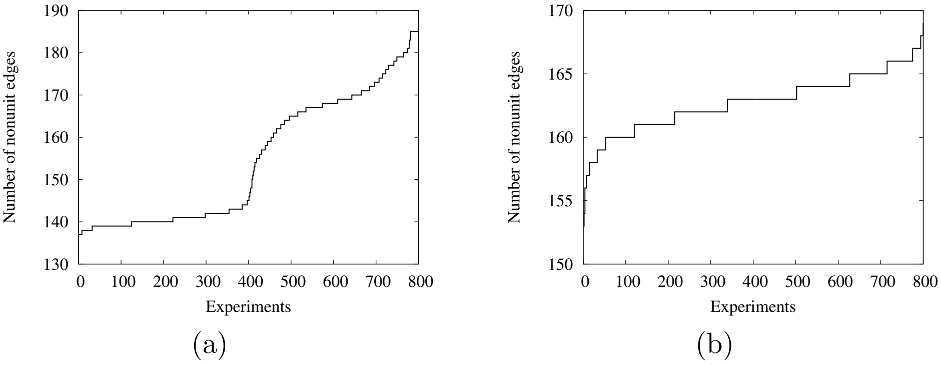


Figure 3. Results for H_h on two real-life instances: (a) example derived from a code for fluid dynamics; and (b) example from a complicated finite elements code.

3.1. *Interpretation of the results*

As with other randomized heuristics, one is curious whether some structure in the problem is essential for the improvement in the cost function. If such a structure could be found and characterized so that it may be recognized with relatively low computational complexity, then the heuristic could be modified to specifically search for and exploit such structure, in turn improving the chances of the thus modified randomized heuristic to find a good solution. The results of our computational experiments indicate that (1) there can be a substantial improvement over H_g and (2) there can be a separation of the cost function values. We followed two paths to analyze the heuristic results.

One approach is to look at the *energy difference* δ_t of the tth step of H_h relative to the step H_g would have taken. In other words, if i is the current state at the tth step, ij is a transition accepted by H_h and ij^* is a transition that should minimize $c(k)$ over all possible transitions ik,

$$\delta_t = c(j) - c(j^*) \ .$$

There are 2 cases:

(1) If there are targets with an improvement to the cost function, then we look among those targets for the energy δ_t of the randomized step vs. the deterministic step:
   a) If $\delta_t = 0$, then we randomize over the artificial order within the graph representation;

b) If $\delta_t > 0$, then we randomize with respect to the actual cost function.

(2) If there are no targets with an improvement to the cost function, then we look among those targets with the analogous subcases (a) and (b) as described in *(i)*.

For each randomized elimination sequence ρ we can then observe how many steps fall into one of the above four categories and also consider the quantity

$$\delta_\rho \equiv \frac{\sum\limits_{\delta_t \in \rho} \delta_t}{|\rho|}$$

as a (rough) measure of the distance of the given randomized heuristic from the deterministic heuristic as far as the actual cost function is concerned, where $|\rho|$ denotes the length of ρ. If we can find ρ with a substantial improvement of the cost function but all steps fall into categories 1(a) and 2(a), then the conclusion to be drawn is that the artificial order in the graph determines most of the cost, the suggested randomization over the energy would not be worth the effort, and one should break ties randomly. On the other hand, if there are no such sequences or even if there are only a few steps with a nonzero δ_t, then the suggested heuristic is justified.

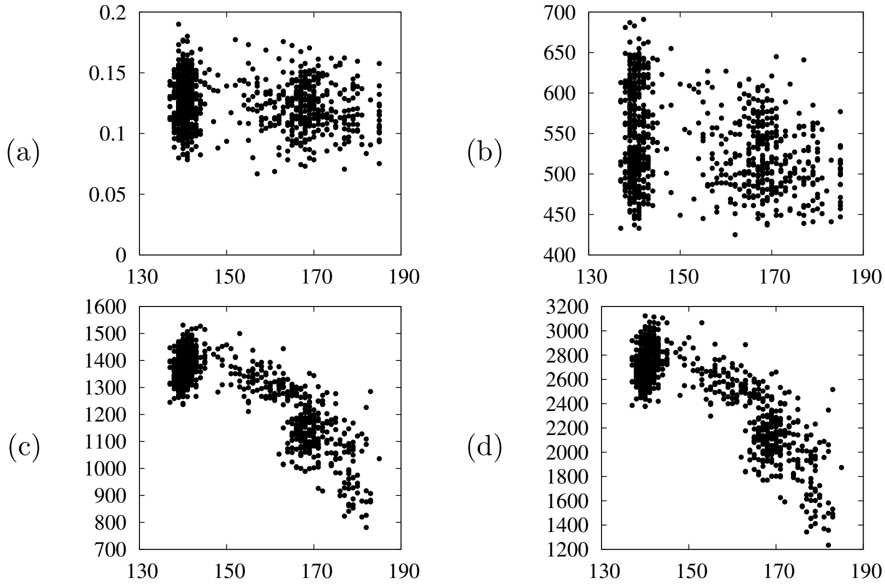


Figure 4. Plotted over the cost function values are δ_ρ in (a), $|\rho|$ in (b), vertex counts in compressed accumulation circuits in (c), edge counts in compressed accumulation circuits in (d).

The δ_ρ values shown in Figure 4(a) indicate that the cases 1(b) and 2(b) play a substantial role and therefore the suggested heuristic is effective. To gain insight into any structural properties, one will eventually have to look at the elimination sequences and compare them. This comparison is nontrivial because, as shown in Figure 4(b), the length of the sequences varies greatly. One might suspect that more elimination steps would be required to drive the cost down and therefore the best sequences would likely be longer than the others. We point out that contrary to this plausible assumption, among the sequences with the lowest cost is also one that is the shortest; see the lower left datapoint in Figure 4(b). This gives another indication that a structural property lies at the heart of the improvement.

Comparing of elimination sequences in order to detect structural properties is difficult for at least two reasons. First, the random sequences are of different length;

and, second, the sequences still embody a substantial artificial order that stems from tie breaking and is not at all relevant to the question of structural properties.

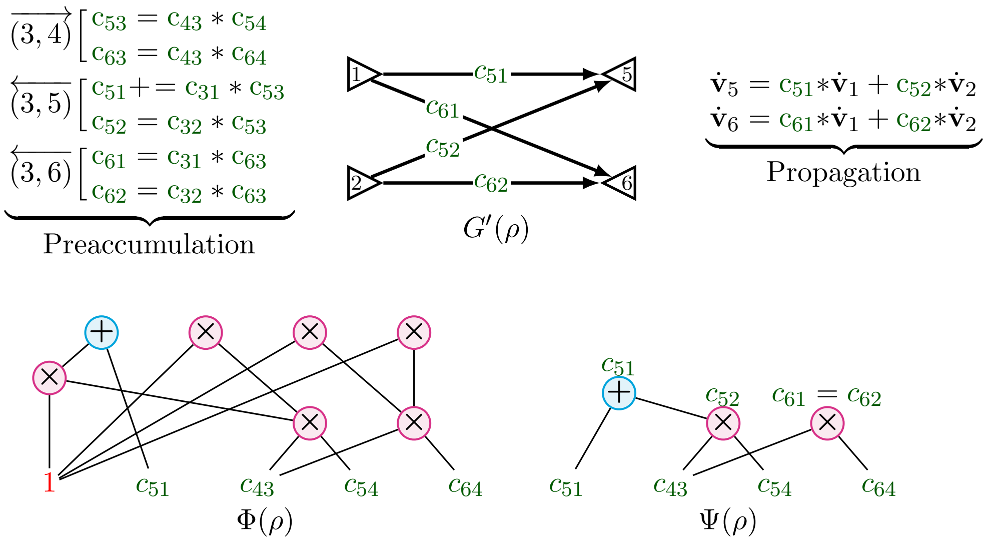

$\overrightarrow{(3,4)}$ $c_{53} = c_{43} * c_{54}$
$c_{63} = c_{43} * c_{64}$
$\overleftarrow{(3,5)}$ $c_{51} += c_{31} * c_{53}$
$c_{52} = c_{32} * c_{53}$
$\overleftarrow{(3,6)}$ $c_{61} = c_{31} * c_{63}$
$c_{62} = c_{32} * c_{63}$
Preaccumulation
1
2
5
6
c_{51}
c_{61}
c_{52}
c_{62}
$G'(\rho)$
$\dot{\mathbf{v}}_5 = c_{51} * \dot{\mathbf{v}}_1 + c_{52} * \dot{\mathbf{v}}_2$
$\dot{\mathbf{v}}_6 = c_{61} * \dot{\mathbf{v}}_1 + c_{62} * \dot{\mathbf{v}}_2$
Propagation
1
c_{51}
c_{43}
c_{54}
c_{64}
$\Phi(\rho)$
c_{51}
c_{52}
$c_{61} = c_{62}$
c_{51}
c_{43}
c_{54}
c_{64}
$\Psi(\rho)$

Figure 5. A full edge elimination sequence $\rho = \left(\overrightarrow{(3,4)}, \overleftarrow{(3,5)}, \overleftarrow{(3,6)}\right)$ is shown along with the corresponding remainder graph $G'(\rho)$, accumulation circuit $\Phi(\rho)$, and compressed accumulation circuit $\Psi(\rho)$. Note the redundancy in the fully preaccumulated Jacobian. Any full edge elimination sequence will result in the same remainder graph G', though different sequences generally imply different computational costs for the preaccumulation phase. However, sequences that are functionally identical may look quite different: For $\tilde{\rho} = \left(\overleftarrow{(4,5)}, \overrightarrow{(3,4)}, \overrightarrow{(1,2)}, \overrightarrow{(2,3)}\right)$, we have $G'(\rho) = G'(\tilde{\rho}), \Phi(\rho) = \Phi(\tilde{\rho})$, and $\Psi(\rho) = \Psi(\tilde{\rho})$.

3.2. *Comparing accumulation circuits*

For a given edge elimination sequence ρ, the essential structure of the computation performed by the corresponding accumulation code is exhibited in the corresponding *accumulation circuit* $\Phi(\rho)$. $\Phi(\rho)$ is an arithmetic circuit whose leaves - the inputs to the circuit - correspond to the variables labeling the edges $E(G)$. The internal nodes of $\Phi(\rho)$ all have exactly two predecessors and are labeled either + (sum gates) or × (product gates), where the label for a gate $\alpha \in \Phi$ is denoted o_α. In general, an accumulation circuit will have many outputs, each of which computes a value carried by an edge in the remainder graph G'. (Note that, for edge elimination, this is also true of some nodes in the accumulation circuit that are not maximal.) In general, the number of edge elimination sequences (partial or full) is much bigger than the number of accumulation circuits that can result from an edge elimination sequence. This gives us an equivalence relation where two edge elimination sequences ρ_1 and ρ_2 may satisfy $\Phi(\rho_1) = \Phi(\rho_2)$ in addition to satisfying $G'(\rho_1) = G'(\rho_2)$. Note that having $G'(\rho_1)$ equal to $G'(\rho_2)$ is necessary but not sufficient for two edge elimination sequences to be considered equivalent. *Compression* of the accumulation circuit, which establishes a kind of canonical form, allows for an even coarser equivalence relation. An example is shown in Figure 5.

The hope is that one may find a pair of sequences that has a substantial difference in the cost function yet the accumulation circuits are similar. The remaining difference then might point to a particular structure that triggers the difference in the cost. The following properties of the accumulation circuit guide the compression. All nonconstant minimal vertices are distinct; all constant minimal vertices either have identical values or are considered distinct; there can be nonmaximal vertices referenced by the remainder graph; and all non-minimal vertices are either multiplication or addition operations. The circuit compression consists of the following steps: (1) collapse all vertices that are minimal, constant, and have identical values to a single representer vertex; (2) replace any constant valued subgraphs S that evaluate to exactly `1.0` and have a single outedge (S, j) by a new edge $(\texttt{1.0}, j)$ from the constant `1.0` representer vertex; (3) contract any edge (i, j) such that i is nonminimal, j is the only successor of i, $o_i = o_j$, and i is not referenced by the remainder graph; (4) collapse to i all nonminimal vertices j, if i and j have identical predecessor multisets[1] and $o_i = o_j$.

The numbers in Figure 4(d) show that compression yields reductions between 611 and 1130. On the compressed accumulation circuits we can recursively build a signature $s_v = (o_v, c_v, \mathcal{V}_v, \mathcal{C}_v)$ for each vertex v by considering its optional operation o_v or constant value c_v, a multiset $\mathcal{V}_v$ of its dependencies on variable minimal vertices, and a multiset $\mathcal{C}_v$ of elements $(c, o, \mathcal{C}^*)$ representing operations with constant values.

Algorithm 3: Construct signatures for accumulation circuit vertices

```
input: initial Accumulation circuit $\Phi$
forall minimal $v$ with constant value $c$ do
    set $s(v) = (., c, \emptyset, \emptyset)$
forall variable minimal $v$ with edge label $c_{ji}$ do
    set $s(v) = (., ., \{c_{ji}\}, \emptyset)$
forall non-minimal $v$ with direct predecessors $\mathcal{P}$ do
    $\mathcal{V}' = \bigcup_{p \in \mathcal{P}} \mathcal{V}_p$;   $\mathcal{C}' = \bigcup_{p \in \mathcal{P}} \mathcal{C}_p$
    if $\mathcal{V}' = \mathcal{C}' = \emptyset$ then
        compute new constant $c_v$ value from all predecessors
        set $s(v) = (., c_v, \emptyset, \emptyset)$
    if $\exists p \in \mathcal{P}$ such that $\mathcal{V}_p = \mathcal{C}_p = \emptyset$ then
        compute new constant $c_v$ value from the constants of those $p$
        set $s(v) = (o_v, ., \mathcal{V}', \{(o_v, c_v, \mathcal{C}')\})$
    else
        set $s(v) = (o_v, ., \mathcal{V}', \mathcal{C}')$
```

The signatures can be built bottom up in the accumulation circuit and, with the multisets lexicographically ordered and suitably represented as a string, can be used to compare two vertices by string comparison. This feature has been implemented to enable comparisons between sequences from the cluster of good solutions and the cluster of solutions closer to the H_g result. Unfortunately, even in circuits of similar size we could match only less than half of the vertices. The alternative search for some vertices occurring only in circuits from the preferred cluster of solutions indicating a crucial step in the elimination has so far not produced tangible results and is subject to further research.

4. Conclusions and Future Work

The design of AD tools that can effectively exploit Jacobian scarcity in a practical setting remains a challenging problem. Through the use of a randomized strat-

[1] The accumulation circuit can have parallel paths; we must determine how often a given vertex is a predecessor.

egy implemented in OpenAD, we were able to significantly reduce the attainable nontrivial edge count in a number of real-world examples. The heuristics and corresponding computational results presented here indicate that a greedy approach is not sufficient to produce optimal partial edge elimination sequences.

4.1. *Rerouting and normalization*

We excluded both reroutings and normalizations from the graph modifications in this paper. In [7] it was shown that normalizations can be done in a postprocessing step which implies that the randomized heuristic is not affected. With respect to reroutings operations the situation is more complicated. The experiments with greedy heuristics in [7] show that some improvement in the final cost function was obtained when rerouting operations were allowed. However, these heuristics yielded an acceptable operations count for the ensuing elimination sequence only when rerouting operations were restricted to combinations of reroutings and edge eliminations, and the respective combined operations also reduced the nontrivial edge count. One can view a rerouting as an inverse edge elimination and that poses a problem for discriminating between simple backtracking in the elimination metagraph and a rerouting step. In fact, for the randomized heuristics proposed here the metagraph is a tree. Because in our experiments the randomized heuristics have always undercut the rerouting-enabled greedy heuristics, we did not see enough justification to introduce the added complexity of rerouting operations to the metagraph.

4.2. *Face elimination*

Thus far, our approach has been to treat edge elimination operations as elemental. Future work on the minimum edge count problem might focus on an alternative elimination framework known as *face elimination*, which involves transformations in the line graph of G [10]. (Note that, in the line graph, the labels representing partial derivatives are associated with the vertices rather than the edges; hence, the corresponding problem would involve minimizing the number of nontrivial *vertex labels* in the line graph.) Face elimination represents a more fine-grained approach and is strictly more general than edge elimination in that any accumulation circuit produced by an edge elimination sequence can also result from some face elimination sequence (the converse does not always hold).

Exploiting scarcity in this framework would require a redefinition of propagation through the remainder line graph that results from a partial face elimination sequence. Such a redefinition might involve some method for reversing the line graph operation, which, in general, is possible only by adding *dummy edges* (in our context these edges would receive the unit label 1); minimizing the number of dummy edges is a nontrivial problem in its own right [6]. While this approach shows some promise, the efficacy of face elimination for minimizing edge counts has not yet been demonstrated.

4.3. *Outlook*

The results we have obtained indicate that the potential for exploiting Jacobian scarcity may be far greater than what the current heuristics - those described in this paper included - can achieve. A more thorough investigation of these possibilities requires a theoretical foundation for the minimum edge count problem. Such a foundation ought to be independent of any particular elimination framework.

Nevertheless, edge elimination has proven to be a sound algorithmic framework for this problem. It would be of some interest to determine whether there is a polynomial-time algorithm that, given a linearized computational graph G, produces a remainder graph with a number of nontrivial edges that coincides with the minimum over all possible edge elimination sequences. While the search space for this problem is truly enormous, the discussion of compressed accumulation circuits in Section 3.2 represents a first step toward identifying of those properties that truly distinguish one elimination sequence from another. Future work in this area should include further investigation of the structure of accumulation circuits.

Acknowledgments

This work was supported in part by the U.S. Department of Energy, under Contract DE-AC02-06CH11357.

References

[1] Andreas Griewank. A mathematical view of automatic differentiation. In *Acta Numerica*, volume 12, pages 321–398. Cambridge University Press, 2003. .

[2] Andreas Griewank and Olaf Vogel. Analysis and exploitation of Jacobian scarcity. In Hans Georg Bock, Ekaterina Kostina, Hoang Xuan Phu, and Rolf Rannacher, editors, *Modeling, Simulation and Optimization of Complex Processes*, pages 149–164, Berlin, 2005. Springer. ISBN 978-3-540-27170-3. .

[3] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.* Number 105 in Other Titles in Applied Mathematics. SIAM, Philadelphia, PA, 2nd edition, 2008. ISBN 978–0–898716–59–7.

[4] Mark Jerrum and Alistair Sinclair. *The Markov chain Monte Carlo method: an approach to approximate counting and integration*, pages 482–520. PWS Publishing Co., Boston, MA, 1997. ISBN 0-534-94968-1.

[5] S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[6] M. S. Krishnamoorthy and N. Deo. Complexity of the minimum-dummy-activities problem in a pert network. *Networks*, 9:189–194, 1979. .

[7] Andrew Lyons and Jean Utke. On the practical exploitation of scarsity. In Christian H. Bischof, H. Martin Bücker, Paul D. Hovland, Uwe Naumann, and J. Utke, editors, *Advances in Automatic Differentiation*, volume 64 of *Lecture Notes in Computational Science and Engineering*, pages 103–114. Springer, Berlin, 2008. ISBN 978-3-540-68935-5. .

[8] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953. .

[9] U. Naumann and Y. Hu. Optimal vertex elimination in single-expression-use graphs. *ACM Transactions on Mathematical Software*, 35(1):1–20, 2008. ISSN 0098-3500. .

[10] Uwe Naumann. Optimal accumulation of Jacobian matrices by elimination methods on the dual computational graph. *Mathematical Programming, Ser. A*, 99(3):399–421, 2004. .

[11] Uwe Naumann and Peter Gottschling. Prospects for simulated annealing in automatic differentiation. In Kathleen Steinhöfel, editor, *Stochastic Algorithms: Foundations and Applications*, number 2264 in LNCS, pages 355–359, Berlin, 2001. Springer. ISBN 3-540-43025-3. .

[12] Uwe Naumann and Peter Gottschling. Simulated annealing for optimal pivot selection in Jacobian accumulation. In Andreas Albrecht and Kathleen Steinhöfel, editors, *Stochastic Algorithms: Foundations and Applications*, number 2827 in Lecture Notes in Computer Science, pages 83–97. Springer, 2003. .

[13] Johannes J. Schneider and Scott Kirkpatrick. *Stochastic Optimization (Scientific Computation).* Springer-Verlag New York, Inc., 2006. ISBN 3540345590.

[14] Jean Utke, Uwe Naumann, Mike Fagan, Nathan Tallent, Michelle Strout, Patrick Heimbach, Chris Hill, and Carl Wunsch. OpenAD/F: A modular, open-source tool for automatic differentiation of Fortran codes. *ACM Transactions on Mathematical Software*, 34(4):18:1–18:36, 2008. .

[15] D. F. Wong, H. W. Leong, and C. L. Liu. *Simulated annealing for VLSI design.* Kluwer Academic Publishers, Norwell, MA, USA, 1988. ISBN 0-89838-256-4.