On the Practical Exploitation of Scarsity

Andrew Lyons and Jean Utke

Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, USA,
{lyonsam,utke}@mcs.anl.gov

Summary. Scarsity is the notion that the Jacobian $\boldsymbol{J}$ for a given function $\boldsymbol{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$ may have fewer than $n * m$ degrees of freedom. A scarse $\boldsymbol{J}$ may be represented by a graph with a minimal edge count. So far, scarsity has been recognized only from a high-level application point of view, and no automatic exploitation has been attempted. We introduce an approach to recognize and use scarsity in computational graphs in a source transformation context. The goal is to approximate the minimal graph representation through a sequence of transformations including eliminations, reroutings, and normalizations, with a secondary goal of minimizing the transformation cost. The method requires no application-level insight and is implemented as a fully automatic transformation in OpenAD. This paper introduces the problem and a set of heuristics to approximate the minimal graph representation. We also present results on a set of test problems.

Key words: reverse mode, scarsity, source transformation

1 Introduction

While automatic differentiation (AD) is established as key technology for computing derivatives of numerical programs, reducing the computational complexity and the memory requirements remains a major challenge. For a given numerical program many different high-level approaches exist for obtaining the desired derivative information; see [2, 1]. In this paper we concentrate on the transformation of the underlying computational graph, defined following the notation established in [2]. Consider a code that implements a numerical function

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) : \mathbb{R}^n \mapsto \mathbb{R}^m \tag{1}$$

in the context of AD. We assume $\boldsymbol{f}$ can be represented by a directed acyclic computational graph $G = (V, E)$. The set $V = X \cup Z \cup Y$ comprises vertices for the n independents X, the m dependents Y, and the p intermediate values Z occurring in the computation of $\boldsymbol{f}$. The edges $(i, j) \in E$ represent the direct dependencies of the $j \in Z \cup Y$ computed with elemental functions $j = \phi(\ldots, i, \ldots)$ on the arguments $i \in X \cup Z$. The computations imply a dependency relation $i \prec j$ and its transitive closure $\prec^*$. The ϕ are the elemental functions (sin, cos, etc.) and operators (+,-,*, etc.) built into the given programming language. All edges $(i, j) \in E$ are labeled with the local partial derivatives $c_{ji} = \frac{\partial j}{\partial i}$.

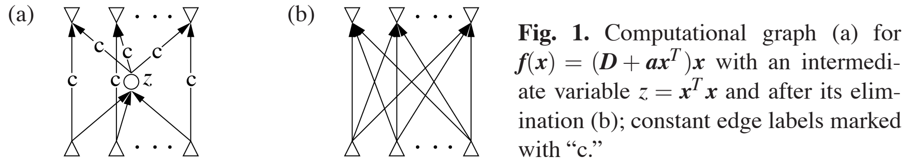

Fig. 1. Computational graph (a) for $\boldsymbol{f}(\boldsymbol{x}) = (\boldsymbol{D} + \boldsymbol{a}\boldsymbol{x}^T)\boldsymbol{x}$ with an intermediate variable $z = \boldsymbol{x}^T\boldsymbol{x}$ and after its elimination (b); constant edge labels marked with "c."

Generally the code for $\boldsymbol{f}$ contains control flow (loops, branches) that precludes its representation as a single G. One could construct G for a particular $\boldsymbol{x} = \boldsymbol{x}_0$ at runtime, for instance with operator overloading. Therefore, any automatic scarsity detection and exploitation would have to take place at runtime, too. Disregarding the prohibitive size of such a G for large-scale problems, there is little hope of amortizing the overhead of graph construction and manipulation at runtime with efficiency gains stemming from scarsity.

In the source transformation context, we can construct local computational graphs at compile time, for instance within a basic block [7]. Because the construction and manipulation of the local graphs happen at compile time, any advantage stemming from scarsity directly benefits the performance, since there is no runtime overhead. We will consider $\boldsymbol{f}$ to be computed by a sequence of basic blocks $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_l$. Each basic block $\boldsymbol{f}_i$ has its corresponding computational graph G_i. Using a variety of elimination techniques [5], one can preaccumulate local Jacobians $\boldsymbol{J}_i$ and then perform propagation

$$\text{forward } \dot{\boldsymbol{y}}_j = J_j\dot{\boldsymbol{x}}_j;\ j = 1, \ldots, l \text{ or reverse } \bar{\boldsymbol{x}}_j = (J_j)^T\bar{\boldsymbol{y}}_j;\ j = l, \ldots, 1\ , \tag{2}$$

where $x_j = (x_i^j \in V : i = 1, \ldots, n_j)$ and $y_j = (y_i^j \in V : i = 1, \ldots, m_j)$ are the inputs and outputs of F_j, respectively. From here on we will consider a single basic block, its computational graph G, and Jacobian $\boldsymbol{J}$ without explicitly denoting the index.

Griewank [3, 4] characterizes scarsity using the notion of degrees of freedom of a mapping from an argument $\boldsymbol{x}$ to the Jacobian $\boldsymbol{J}(\boldsymbol{x}) \in \mathbb{R}^{m\times n}$. $\boldsymbol{J}$ is said to be scarse when there are fewer than $n \cdot m$ degrees of freedom. Consequently, $\boldsymbol{J}$ that are sparse or have constant entries will also be scarse. Likewise, rank deficiency can lead to scarsity, for example $\boldsymbol{f}(\boldsymbol{x}) = (\boldsymbol{D} + \boldsymbol{a}\boldsymbol{x}^T)\boldsymbol{x}$. Here we see that the Jacobian is dense but is the sum of a diagonal matrix $\boldsymbol{D}$ and a rank 1 matrix $\boldsymbol{a}\boldsymbol{x}^T$. Generally, scarsity can be attributed to a combination of sparsity, linear operations, and rank deficiency; see [4] for more details. The origin of scarsity is not always as obvious as in our example but is represented in the structure of the underlying computational graph. The graph G for our example is shown in Fig. 1. Clearly, it has only $3n$ edges, n of which are constant. If we eliminate the intermediate vertex z (see also Sec. 2.1), we end up with the n^2 nonconstant edges. When the reverse mode implementation relies on storing the preaccumulated local Jacobians for use in (2), a reduction in the number of elements to be stored translates into a longer section between checkpoints, which in turn may lead to less recomputation in the checkpointing scheme.[1] Often the edge labels not only are constant but happen to be ± 1. Aiming at the minimal number of nonunit edges will greatly benefit propagation with (2), especially in vector mode; this was the major motivation for investigating scarsity in [4]. Given these potential benefits of scarsity, we address a number of questions in this paper.

- Can we automatically detect scarsity in a source transformation context given that we see only local Jacobians and assume no prior knowledge of scarsity from the application context?

[1] One can, of course, also consider the accumulation of the scarse Jacobian directly in the reverse sweep, but this is beyond the scope of this paper.

- What is a reasonable heuristic that can approximate a minimal representation of the Jacobian?
- How can this be implemented in an AD tool? Are there practical scenarios where it matters?

Section 2 covers the practical detection of scarsity and the propagation, Sec. 3 the results on some test examples, and Sec. 4 a summary and outlook.

2 Scarsity

The notion of scarsity for the local Jacobians introduced in Sec. 1 can already be beneficial for single assignment statements. For instance consider $s = \sin(x_1 + x_2 + x_3 + x_4)$, whose computational graph G is shown in Fig. 2. The initial G has one variable and six unit edges. After eliminating the vertices (see Sec. 2.1) whose in and out edges are all unit labeled, we still have one variable and four unit edges; see Fig. 2(b). Complete elimination gives four variable edges. A forward propagation implementing (2) on a graph $G = (V,E)$ is simply the chain rule $\dot{k} += c_{kl}\dot{l}$ for all (l,k) in topological order. In vector mode this means each element $\dot{x}^i_j$ of $\dot{\boldsymbol{x}}_j$ and with it each $\dot{v} \in V$ is itself a vector $\in \mathbb{R}^p$ for some $0 < p$. Likewise, a reverse propagation is implemented by $\bar{l} += c_{kl}\bar{k}$ for all (l,k) in reverse topological order. Consequently, in our example the propagation through (b) entails $3p$ scalar additions and p scalar multiplications, while with (c) we would have the same number of additions but $4p$ scalar multiplications. Clearly, in (c) all the edge labels are numerically the same; however, this information is no longer recognizable in the structure.

Flattening of assignments into larger computational graphs [7] will provide more opportunity for exploiting these structural properties beyond what can be achieved by considering only single assignments. This graph-based framework, on the other hand, limits the scope of our investigation to structural properties. For example, the fact that in a graph for the expression $\sum x_i^2$ all edge labels emanating from the x_i have a common factor 2 is not recognizable in the structure. Such algebraic dependencies between edge labels can in theory lead to further reductions in the minimal representation. However, their investigation is beyond the scope of this paper.

For a given graph G we want an approximation G^* to the corresponding *structurally minimal graph*, which is a graph with the minimal count of nonconstant edge labels. Alternatively, we can aim at an approximation G^+ to the *structurally minimal unit graph*, which is a graph with the minimal count of nonunit edge labels.

2.1 Transformation Methods

Following the principal approach in [4] we consider a combination of edge elimination, rerouting, and normalization to transform the input graph G.

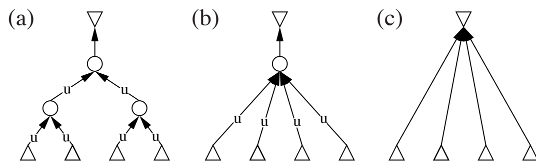

Fig. 2. Computational graph for $s = \sin(x_1 + x_2 + x_3 + x_4)$ (a), minimal representation after partial elimination (b), and complete elimination (c); unit edge labels are marked with "u."

Elimination: An edge (i,j) can be *front eliminated* by reassigning the labels
$c_{ki} = c_{ki} + c_{kj} \cdot c_{ji} \forall k \succ j$, followed by the removal of (i,j).
An edge (j,k) can be *back eliminated* by reassigning the labels
$c_{ki} = c_{ki} + c_{kj} \cdot c_{ji} \forall i \prec j$, followed by the removal of (j,k).

Grouping the elimination of all in or out edges of a given vertex amounts to vertex elimination. We do not consider face elimination [5], a more general technique, because it entails transformations of the line graph of G that can result in intermediate states for which we cannot easily find an optimal propagation (2).

Rerouting was introduced in [6] to perform a factorization and then refined in [4] as follows.

Rerouting: An edge (j,l) is *prerouted* via pivot edge (k,l) (see Fig. 5) by setting
$c_{kj} = c_{kj} + \gamma$ with $\gamma \equiv c_{lj}/c_{lk}$ for the *increment edge* (j,k)
$c_{hj} = c_{hj} - c_{hk} \cdot \gamma$ for $h \succ k, h \neq l$ for the *decrement edges* (j,h)
followed by the removal of (j,l) from G.
An edge (i,k) is *postrouted* via pivot edge (i,j) by setting
$c_{kj} = c_{kj} + \gamma$ with $\gamma \equiv c_{ki}/c_{ji}$ for the *increment edge* (j,k)
$c_{lk} = c_{lk} - c_{jl} \cdot \gamma$ for $l \prec j, l \neq i$ for the *decrement edges* (l,k)
followed by the removal of (i,k) from G.

If any of the above transformations leaves an intermediate vertex v without in or out edges, v and all incident edges are removed from G. Our primary goal is the approximation of G^* or G^+, respectively. However, we also track the count of operations (multiplications and divisions) as a secondary minimization goal for the overall transformation algorithm. A related concern is the fill-in, that is, the creation of new edges during the transformation process.

Normalization: An in edge (i,j) is *forward normalized* by setting
$c_{kj} = c_{kj} \cdot c_{ji} \forall k \succ j$ and $c_{jl} = c_{jl}/c_{ji} \forall l \prec j, l \neq i$ and finally $c_{ji} = 1$.
An out edge (j,k) is *backward normalized* by setting
$c_{ji} = c_{ji} \cdot c_{kj} \forall i \prec j$ and $c_{lj} = c_{lj}/c_{kj} \forall l \succ j, l \neq k$ and finally $c_{kj} = 1$.

Here, no fill-in is created, but normalization incurs multiplication and division operations.

2.2 Elimination and Scarsity

A complete sequence $\sigma = (\varepsilon_1, \ldots,)$ of eliminations makes a given G bipartite. Each elimination step ε_s in the sequence transforms $G_s = (V_s, E_s)$ into $G_{s+1} = (V_{s+1}, E_{s+1})$, and we can count the number of nonconstant edge labels $|E_s|^*$ or nonunit edge labels $|E_s|^+$. As indicated by the examples in Figs. 1 and 2, there is a path to approximating the minimal representation via incomplete elimination sequences. To obtain G^*, we prefer eliminations that *preserve scarsity*, that is, do not increase the nonconstant edge label count. To obtain G^+, we prefer eliminations that *preserve propagation*, that is, do not increase the nonunit edge label count. Figure 3 shows examples for cases in which the minimal edge count can be reached only after

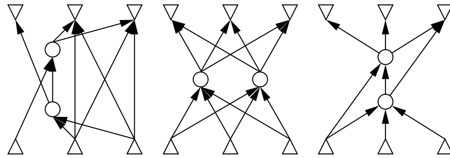

Fig. 3. Examples for graphs G in which there is no edge elimination sequence with monotone edge counts.

a temporary increase above the count in G because any edge elimination that can be attempted initially raises the edge count. Consequently, a scarsity-preserving elimination heuristic H_e should allow a complete elimination sequence and then backtrack to an intermediate stage G_s with minimal nonconstant or nonunit edge label count. Formally, we define the heuristic as a chain $F_q \circ \ldots \circ F_1 \mathscr{T}$ of q filters F_i that are applied to a set of elimination targets $\mathscr{T}$ such that $F_i \mathscr{T} \subseteq \mathscr{T}$ and if $F\mathscr{T} = \emptyset$ then $F \circ \mathscr{T} = \mathscr{T}$ else $F \circ \mathscr{T} = F\mathscr{T}$. Following the above rationale, we apply the heuristic $H_e E_s$ with five filters at each elimination stage $G_s = (V_s, E_s)$.

$$\begin{array}{ll} \mathscr{T}_1 = F_1 E_s & : \text{the set of all eliminatable edges} \\ \mathscr{T}_2 = F_2 \mathscr{T}_1 & : e \in \mathscr{T}_1 \text{ such that } |E_s|^* \leq |E_{s+1}|^* \text{ (or } |E_s|^+ \leq |E_{s+1}|^+ \text{ resp.)} \\ \mathscr{T}_3 = F_3 \mathscr{T}_2 & : e \in \mathscr{T}_2 \text{ such that } |E_s|^* < |E_{s+1}|^* \text{ (or } |E_s|^+ < |E_{s+1}|^+ \text{ resp.)} \\ \mathscr{T}_4 = F_4 \mathscr{T}_3 & : e \in \mathscr{T}_3 \text{ with lowest operations count (Markowitz degree)} \\ \mathscr{T}_5 = F_5 \mathscr{T}_4 & : \text{ reverse or forward as tie breaker} \end{array} \quad (3)$$

With the above definition of "$\circ$" the filter chain prefers scarsity-preserving (or propagation-preserving) eliminations and resorts to eliminations that increase the respective counts only when reducing or maintaining targets are no longer available. Note that the estimate for the edge counts after the elimination step has to consider constant and unit labels, not only to determine the proper structural label for fill-in edges but also to recognize that an incremented unit edge will no longer be unit and a constant edge absorbing variable labels will no longer be constant. As part of the elimination algorithm we can now easily determine the earliest stage s for the computed sequence σ at which the smallest $|E_s|^*$ (or $|E_s|^+$, respectively) was attained and take that G_s as the first approximation to G^* (or G^+, respectively).

Given the lack of monotonicity, it would not be surprising to find a smaller $|E_s|$ with a different σ. However, aside from choices in F_5 and fundamentally different approaches such as simulated annealing, there is no obvious reason to change any of the other filters with respect to our primary objective. On the other hand, given that we have a complete elimination sequence with H_e we can try to find improvements for our secondary objective, the minimization of elimination operations. We say that an edge has been *refilled* when it is eliminated and subsequently recreated as fill-in (see Sec. 2.1). Naumann conjectures that an elimination sequence with minimal operations count cannot incur refill. Having a complete elimination sequence σ_1, we can easily detect refilled (i,k) and insert the fact that in G there is a path $i \to j \to k$ as an edge-vertex pair $\langle (i,k) : j \rangle$ into a refill-dependency set $\mathscr{R}$. We inject another filter $F_{\mathscr{R}}$ before F_4 to avoid target edges that have been refilled in previous elimination sequences by testing nonexistence of paths.

$$F_{\mathscr{R}} \mathscr{T} : (i,k) \in \mathscr{T} \text{ such that } \forall j : \langle (i,k) : j \rangle \in \mathscr{R} \text{ it holds that } (i \not\to j \vee j \not\to k \text{ in } G_s) \quad (4)$$

We then can compute new elimination sequences $\sigma_2, \sigma_3, \ldots$ with the thus-augmented heuristic $H_{e\mathscr{R}}$ by backtracking to the earliest elimination of a refilled edge, updating $\mathscr{R}$ after computing each σ_i until $\mathscr{R}$ no longer grows. Clearly, this filter construction will not always avoid refill, but it is an appropriate compromise, not only because it is a relatively inexpensive test, but also because the backtracking to the minimal G_s for our primary objective may well exclude the refilling elimination steps anyway; see also Sec. 3. Among all the computed σ_i we then pick the one with the minimal smallest $|E_s|^*$ (or $|E_s|^+$, respectively) and among those the one at the earliest stage s.

2.3 Rerouting, Normalization and Scarsity

Griewank and Vogel [4] present simple examples showing that relying only on eliminations is insufficient for reaching a minimal representation; one may need rerouting and normalization. The use of division in rerouting and normalization immediately necessitates the same

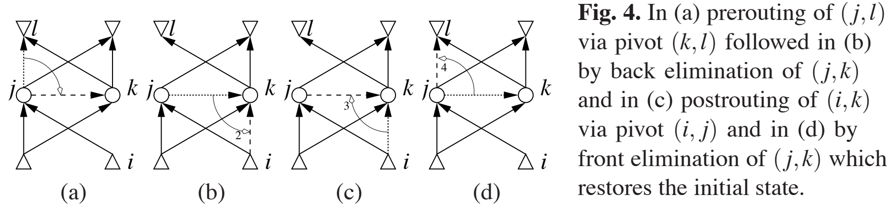

Fig. 4. In (a) prerouting of (j,l) via pivot (k,l) followed in (b) by back elimination of (j,k) and in (c) postrouting of (i,k) via pivot (i,j) and in (d) by front elimination of (j,k) which restores the initial state.

caution against numerically unsuitable pivots that has long been known in matrix factorization. Because only structural information is available in the source transformation context, the only time when either transformation can safely be applied is with constant edge labels whose values are known and can be checked at compile time. However, we temporarily defer the numerical concerns for the purpose of investigating a principal approach to exploiting rerouting and normalization.

When considering rerouting as a stand-alone transformation step that can be freely combined with eliminations into a transformation sequence, one quickly concludes that termination is not guaranteed. Figure 4 shows a cycle of reroutings and edge eliminations that can continue indefinitely. While such repetitions can (like the refill of edges) be prevented, a single rerouting clearly does not guarantee a decrease of the edge count or the maximal path length P_m in G or the total path length P_t (the sum of the length of all paths). An individual rerouting can reduce the edge count by at most one, but in such situations there is an edge elimination that we would prefer; see Fig. 5. This, however, is also an example where a pair of reroutings may be advantageous. As with other greedy heuristics, one might improve the results by introducing some look-ahead. However, we decided that little evidence exists to justify the complexity of such a heuristic. Instead we will concentrate on our primary objective and investigate the scenarios in which a single rerouting-elimination combination reduces $|E_s|^*$ (or $|E_s|^+$, respectively), which can happen in the following cases for prerouting (j,l) via pivot (k,l).

1: The increment edge (j,k) (see Fig. 5) can be eliminated immediately afterwards.
2: The pivot (k,l) (see Fig. 5) becomes the only inedge and can be front eliminated.
3: Removing a rerouted edge (j,l) enables a $|E_s|^*$ or $|E_s|^+$ reducing elimination of an edge (l,q) or (o,j).
4: Creating an increment edge (i,k) as fill-in enables a $|E_s|^*$ or $|E_s|^+$ preserving elimination of an edge (i,j) or (j,k) by absorption into (i,k).

The statements for postrouting are symmetric. Cases 1 and 2 are fairly obvious. Case 3 is illustrated by Fig. 6 and case 4 by Fig. 7. In case 4 we can, however, consider front eliminating (i,j), which also creates (i,k), and then front eliminating (i,k), which yields the same reduction while avoiding rerouting altogether. Given this alternative, we point out that case 3 in Fig. 6 permits an $|E_s|^*$ or $|E_s|^+$ maintaining back elimination of (l,q_1) but no further

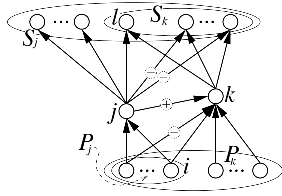

Fig. 5. Prerouting (j,l) via pivot (k,l) would reduce the edge count, but here one could also eliminate (j,k). On the other hand, after eliminating (j,k) one might decide to postroute (i,k) via pivot (i,j) thereby refilling (j,k) and then eliminating it again. Such a look-ahead over more than one step is possible but complicated and therefore costly as a heuristic.

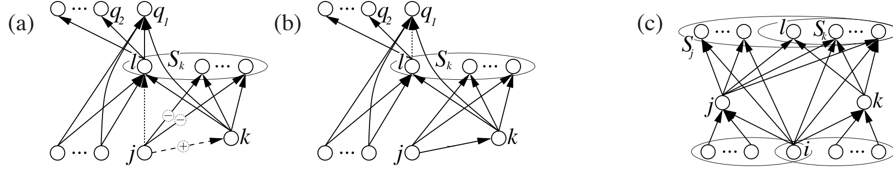

Fig. 6. After prerouting (j,l) where (j,k) is fill-in (a), we can back eliminate (l,q_1) to achieve an edge count reduction (b). In (c) the use of prerouting to eliminate (j,l) to achieve a reduction is unnecessary because front eliminating (i,j) and (i,k) leads to the same reduction.

reduction, while a second possible scenario for case 3 (see Fig. 6(c)) avoids rerouting. We can now use scenarios 1 to 4 to construct a filter F_r for edge-count-reducing rerouting steps to be used in a greedy heuristic.

In the above scenarios there is no provision for creating decrement edges as fill-in. Rather, we assume that (e.g., for prerouting (j,l) as in Fig. 5) the successor set S_k of vertex k is a subset of S_j. Given a numerically suitable pivot, one could have an decrement edge (j,h) as fill-in. However, doing so creates a pseudo-dependency in the graph. Although the labels of (j,k) combined with (j,k) and (k,h) cancel each other out in exact arithmetic, such a modification of the structural information in the graph violates our premise of preserving the structural information and is therefore not permitted.

When considering the effects of normalization steps from a structural point of view, one has to take into account that in a given graph G_s we can normalize at most one edge per intermediate vertex $i \in Z_s$; see also Sec. 1. If we exclude nonconstant edges incident to $i \in Z$, where i has another incident constant or unit edge , we can guarantee a reduction in $|E_s|^*$ or $|E_s|^+$, respectively. This permits a simple heuristic for applying normalizations, but it is clearly not optimal. For instance, one can consider a graph such as ①–u→②⟶③–u→④⟶⑤, where no intermediate vertex is free of incident unit edges; but clearly we could, for instance, forward normalize $(2,3)$, which would maintain the count, since $(3,4)$ would no longer be a unit edge, but then forward (re)normalize $(3,4)$ and have just one nonunit edge left. In general, however, permitting normalizations that do not strictly reduce the edge count would require an additional filter to ensure termination of the heuristic. Such a filter could be based on an ordering of the $i \in Z_s$, or it could prevent repeated normalizations with respect to the same i, but neither implied order has an obvious effect relating to the preexisting constant or unit edges.

Proposition: Normalization does not enable reductions in subsequent eliminations unless all involved edges are constant.

Proof: Consider a front elimination substep (equivalent to a face elimination [5]) of combining a nonconstant, normalized (i,j) with (j,k) into (i,k) where it is potentially absorbed; see Fig. 8. We distinguish three major cases and a number of subcases as follows.

1. If (i,j) is front normalized, then (j,k) is variable, so will be (i,k), ⇒ skip normalization.

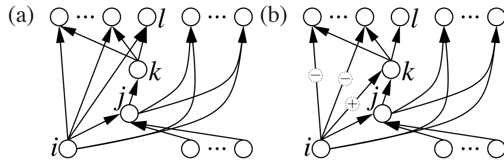

Fig. 7. In the initial scenario (a), back eliminating (j,k) would create (i,k) as fill-in. Alternatively, prerouting (i,l) creates (i,k) as increment fill-in (b); we can then back eliminate (j,k), absorbing into (i,k).

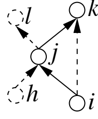

Fig. 8. We consider the normalization of (i, j) and the subsequent effects on eliminations related to (i, j). There is a case distinction depending on the existence of the vertices h and l and the dashed edges (h, j), (j, l), and (i, k).

2. If (i, j) is back normalized and (i, k) existed, then it will be variable, $\Rightarrow$ skip normalization.
3. If (i, j) is back normalized and (i, k) did not exist before
   a) If (j, k) is variable, $\Rightarrow$ skip normalization.
   b) If (j, k) is constant or unit, then so will be (i, k)
      - If (h, j) exists then all out edges of j must be retained, $\Rightarrow$ no edge count reduction.
      - If (j, l) exists then all in edges of j must be retained, $\Rightarrow$ no edge count reduction.
      - If neither (h, j) nor (j, l) exist, $\Rightarrow$ no reduction in $|E_s|^*$ or $|E_s|^+$, respectively.

For normalizing an edge to have an effect on any elimination, the normalized edge has to be consecutive to a constant or unit edge. Therefore we can restrict consideration to the immediately subsequent elimination. □

We conjecture the same to be true when we permit subsequent reroutings. Consequently, we will postpone all normalizations into a second phase after the minimal G_s has been found. Considering the above, we can now extend the elimination heuristic $H_{e\mathscr{R}}$ by extending the initial target set $\mathscr{T}$ to also contain reroutable edges (which exclude edges that have previously been rerouted) and using F_r defined above. The filters in $H_{e\mathscr{R}}$ act only on eliminatable edges, while F_r acts only on reroutable edges. We provide a filter F_e that filters out eliminatable edges and define our heuristic as

$$H_r = F_5 \circ F_4 \circ F_{\mathscr{R}} \circ F_e \circ F_3 \circ F_r \circ F_2 \circ F_1 \mathscr{T} . \tag{5}$$

3 Test Examples

For comparison we use computational graphs from four test cases A-D, which arise from kernels in fluid dynamics, and E, which is a setup from the MIT General Circulation Model. Table 1 shows the test results that exhibit the effects of the heuristics $H_{e\mathscr{R}}$ from Sec. 2.3 and H_r from (5). The results indicate, true to the nature of the heuristic, improvements in all

Table 1. Test results for pure elimination sequences according to H_e and sequences including rerouting steps according to (5), where $|E|$ is the initial edge count and $\#(\boldsymbol{J})$ is the number of nonzero entries in the final Jacobian. We note the minimal edge count, reached at step s, and the number of reroutings r and reducing normalization targets i at that point, where applicable.

			Pure Edge Eliminations							With Rerouting										
			min $\|E\|$		min $\|E^*\|$		min $\|E^+\|$			min $\|E\|$			min $\|E^*\|$			min $\|E^+\|$			
	$\|E\|$	$\#(\boldsymbol{J})$	s	$\|E_s\|$	s	$\|E_s^*\|$	s	$\|E_s^+\|$	i	s	$\|E_s\|$	r	s	$\|E_s^*\|$	r	s	$\|E_s^+\|$	r	i
A	444	615	197	249	192	231	192	231	5	200	248	2	362	226	7	362	226	7	5
B	105	34	70	34	45	22	45	22	0	70	34	0	85	22	0	85	22	0	0
C	209	325	191	130	14	93	14	93	1	173	122	11	97	83	4	97	83	4	0
D	419	271	282	192	373	185	371	185	17	384	150	6	486	178	11	652	167	26	8
E	4554	2136	2442	2094	1852	1463	1852	1463	0	2442	2094	0	2112	1459	4	2112	1459	4	0

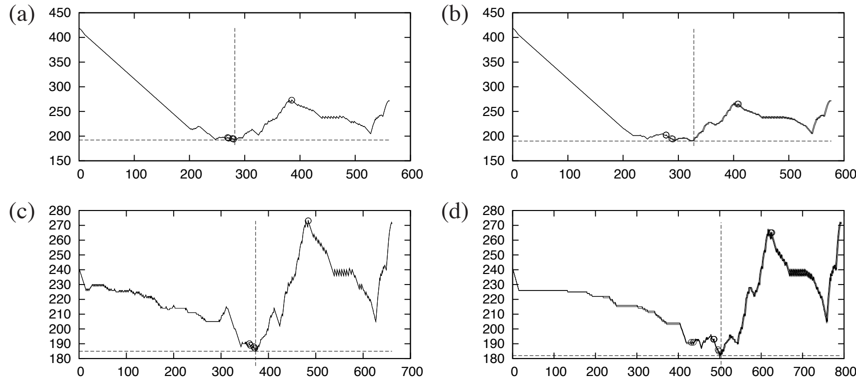

Fig. 9. Edge count over transformation step for test case D, with refill marked by circles: (a) all edges assumed to be variable, leads to two (identical) sequences with $|E_{282}| = 192$ and 4 active refills; (b) same as (a) but without F_3 with $|E_{326}| = 190$ and 2 active refills in σ_2; (c) same as (a) but considers constant edge labels with $|E^*_{373}| = 185$ and 4 active refills in 2 (identical) sequences (the result for unit labels is almost the same: $|E^+_{371}| = 185$); (d) same as (c) but without F_3, leading to 3 sequences with a reduction from 5 to 2 active refills and $|E^*_{500}| = 182$.

test cases. Compared to the number of nonzero Jacobian elements, the improvement factor varies but can, as in case C, be larger than 3. For minimizing $|E^+|$ we also provide i as the number of intermediate vertices that do not already have an incident unit edge. Assuming the existence of a suitable pivot, i gives a lower bound for the number of additional reductions as a consequence of normalization. We also observe that, compared to eliminations, there are relatively few reroutings and, with the exception of case D, the actual savings are rather small. Case D, however, has the single biggest graph in all the cases and offers a glimpse at the behavior of the heuristics shown in Fig. 9. Given the small number of reroutings, one might ask whether one could allow stand-alone rerouting steps that merely maintain the respective edge count and aren't necessarily followed by a reducing elimination. The profile in Fig. 10 exhibits the problems with controlling the behavior of free combinations of reroutings and eliminations that may take thousands of steps before reaching a minimum. Our experiments show that such heuristics sometimes produce a lower edge count, for instance $|E^+_{1881}| = 150$ for case D with 657 reroutings and 48 active refills. In such cases, the huge number of steps required to reach these lower edge counts renders them impractical.

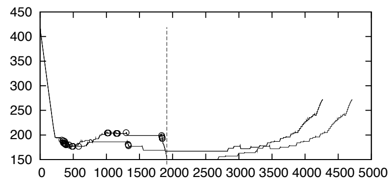

Fig. 10. Here the heuristic has been modified to allow isolated reroutings that do not increase the nontrivial edge count. The result is 3 sequences, the best of which obtains $|E_{1913}| = 150$ with 16 active refills and 768 reroutings.

4 Conclusions and Outlook

We have demonstrated an approach for exploiting the concept of Jacobian scarsity in a source transformation context. The examples showed savings for the propagation step up to a factor of three. We introduced heuristics to control the selection of eliminations and reroutings. A tight control of the rerouting steps proved to be necessary with the practical experiments. Even without any reroutings, however, we can achieve substantial savings. This approach bypasses the problem of choosing potentially unsuitable pivots, particularly in the source transformation setting considered here. One possible solution to this problem entails the generation of two preaccumulation source code versions. A first version would include rerouting/normalization steps, checking the pivot values at runtime, and switching over to the second, rerouting/normalization-free version if a numerical threshold was not reached. Currently we believe the implied substantial development effort is not justified by the meager benefits we observed with rerouting steps. However, the implementation of rerouting as a structural graph manipulation in the OpenAD framework allows us to track the potential benefits of rerouting for future applications.

Acknowledgement. We thank Andreas Griewank for discussions on the topic of scarsity. The authors were supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy under Contract DE-AC02-06CH11357.

References

1. Bücker, H.M., Corliss, G.F., Hovland, P.D., Naumann, U., Norris, B. (eds.): Automatic Differentiation: Applications, Theory, and Implementations, *Lecture Notes in Computational Science and Engineering*, vol. 50. Springer, New York (2005)
2. Griewank, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. No. 19 in Frontiers in Appl. Math. SIAM, Philadelphia (2000)
3. Griewank, A.: A mathematical view of automatic differentiation. In: Acta Numerica, vol. 12, pp. 321–398. Cambridge University Press (2003)
4. Griewank, A., Vogel, O.: Analysis and exploitation of Jacobian scarcity. In: H. Bock, E. Kostina, H. Phu, R. Rannacher (eds.) Modelling, Simulation and Optimization of Complex Processes, pp. 149–164. Springer, New York (2004)
5. Naumann, U.: Optimal accumulation of Jacobian matrices by elimination methods on the dual computational graph. Mathematical Programming, Ser. A **99**(3), 399–421 (2004)
6. Utke, J.: Exploiting macro- and micro-structures for the efficient computation of newton steps. Ph.D. thesis, Technical University of Dresden (1996)
7. Utke, J.: Flattening basic blocks. In: Bücker et al. [1], pp. 121–133

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.