

Derivative Accumulation on Terminal Series-Parallel Graphs

Andrew Lyons
Computation Institute, University of Chicago,
5640 S. Ellis Avenue, Chicago, IL 60637
lyonsam@gmail.com

December 20, 2008

Abstract

The process of accumulating the Jacobian matrix J for a function F can be described by an arithmetic circuit that computes the nonzero entries of J . The minimal nodes in the circuit are the local partial derivatives which label the edges of a directed acyclic graph G that represents a procedure for evaluating F . We give a polynomial time algorithm that finds all circuits with a minimum number of nodes for a particular class of directed acyclic graphs defined by a recursive construction procedure. We also characterize this class in terms of the structure of the corresponding optimal accumulation circuits.

TODO:

- Do I need to show that this minimizes multiplications as well?

1 Introduction and motivation

The accumulation of derivatives by means of automatic differentiation is a compute-intensive task.

Naumann has shown in [?] (by reduction from ENSEMBLE COMPUTATION [2]) that optimal accumulation of derivatives is an NP-complete problem, but the proof requires that algebraic dependences among the local partials exist. Throughout this paper, we assume that all local partials are algebraically independent. Furthermore, we don't allow edge rerouting operations. We might say that we are limited to "chain rule-based" methods of accumulation. The complexity of derivative accumulation under these restrictions has long been conjectured to be NP-complete, but no proof currently exists.

The main result of this paper is an algorithm that solves this problem for functions whose evaluation procedure admits a particular structure. The class of series-parallel graphs can be recognized in linear time.

In Sec. 2 we introduce the optimal Jacobian accumulation problem and provide background.

2 Linearized computational graphs and Bauer's formula

We consider vector functions of the form

$$\mathbf{y} = F(\mathbf{x}), \quad F : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

that map a vector $\mathbf{x} = (x_i)_{i=1,\dots,n}$ of *independent* variables to a vector $\mathbf{y} = (y_i)_{i=1,\dots,m}$ of *dependent* variables. We assume that F is given as an implementation in computer code, or, more generally, as an *evaluation procedure* that can be decomposed into a sequence of elementary assignments

$$\begin{aligned} v_{i-n} = x_i & \} & i = 1 \dots n \\ v_i = \varphi_i(v_j)_{j \prec i} & \} & i = 1 \dots l \\ y_{m-i} = v_{l-i} & \} & i = m - 1 \dots 0 \end{aligned} \quad (1)$$

where each φ_j is assumed to be continuously differentiable and the precedence relation \prec is defined by $j \prec i \Leftrightarrow v_j$ is an argument of φ_i . We will refer to F and a given evaluation procedure for F interchangeably. The variables v_j represent real values that are computed during an evaluation of F at a particular argument \mathbf{x}_0 . Automatic (or algorithmic) differentiation (AD) works by algorithmically applying the chain rule to F in order to produce a new function F^+ that computes not only \mathbf{y} for a given argument \mathbf{x}_0 but also some derivative information for F . In the context of this paper, we will be generating a sequence of statements that yields the entries of the Jacobian matrix $F'(\mathbf{x}_0)$. We follow the notation in the standard reference [3] throughout.

Example. Consider the vector function $F : \mathbb{R}^2 \rightarrow \mathbb{R}$ defined by

$$y = \sin(x_1 * x_2) * x_1 * x_2 * \cos(x_1 * x_2). \quad (2)$$

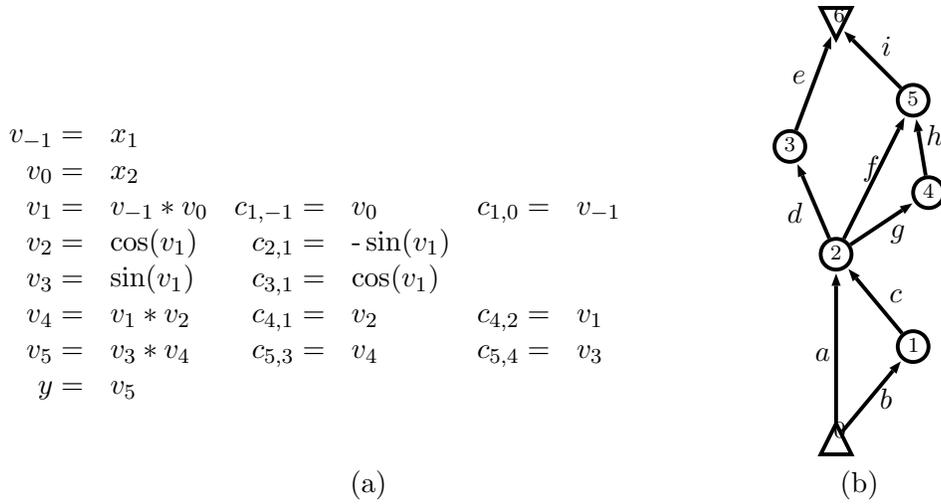


Figure 1: A linearized evaluation procedure for F (a) and the corresponding LCG G (b).

An evaluation procedure for F is shown in Fig. 1(a). We say that an evaluation procedure is *linearized* when it has been augmented to include assignments for the local partial derivatives

$$c_{i,j} \equiv \frac{\partial}{\partial v_j} \varphi_i(v_j)_{j \prec i}, \quad i = 1, \dots, l + m.$$

In the context of numerical programs (where AD is most often applied), an AD compiler will add the statements for the local partials $c_{i,j}$, which can easily be generated when φ_i is a basic differentiable mathematical function (such as $+$, $-$, $*$, $/$, \sin , \cos , \exp , etc.).

The dependence relation \prec on the variables induces a directed acyclic *linearized computational graph* (LCG) $G = (V, E)$ whose vertex set $V = (X, Y, Z)$ comprises the set $X = \{1 - n, \dots, 0\}$ of

independent vertices, the set $Y = \{l+1, \dots, l+m\}$ of dependent vertices, and the set $Z = \{1, \dots, l\}$ of *intermediate* vertices. There is an edge from j to i in G if and only if $v_i \prec v_j$. For a vertex $i \in V$ we denote the predecessor set $\{j : v_j \prec v_i\}$ of i by P_i and we denote the successor set $\{h : v_i \prec v_h\}$ of i by S_i . The edges $(j, i) \in E$ are labeled with their respective local partial derivatives $c_{i,j}$.

Baur and Strassen showed in [1] that in terms of G , the entries of the Jacobian matrix can be computed as

$$f'_{j,i} \equiv \frac{\partial y_j}{\partial x_i} = \sum_{[i \rightarrow j] \in G} \prod_{c_{l,k} \in [i \rightarrow j]} c_{l,k} \quad , \quad (3)$$

where $[i \rightarrow j]$ denotes a path in G from vertex k to l . In other words, $f'_{j,i}$ is the sum of the products of local partials along each path from i to j in G . Directly applying this formula (in a naive way) to our example function F would mean computing

$$F' = \left(c_{1,-1}c_{3,1}c_{5,3} + c_{1,-1}c_{4,1}c_{5,4} + c_{1,-1}c_{2,1}c_{4,2}c_{5,4} \quad c_{1,0}c_{3,1}c_{5,3} + c_{1,0}c_{4,1}c_{5,4} + c_{1,0}c_{2,1}c_{4,2}c_{5,4} \right) ,$$

at a cost of 4 additions and 14 multiplications.

The properties of the chain rule result in a great deal of freedom in how the entries of F' are *accumulated* from the local partials $c_{i,j}$. We are interested in generating an optimal evaluation procedure for F' in terms of the computational cost. More formally, we are interested in solving the following problem (stated as a decision problem):

Problem 1. OPTIMAL JACOBIAN ACCUMULATION (OJA)

*Given an LCG G for the evaluation procedure for some vector function F as defined in Eqn. (1) and a positive integer Ω , is there a sequence of scalar assignments $u_k = s_k \circ t_k$, $\circ \in \{+, *\}$, $k = 1, \dots, \omega$, where each s_k and t_k is either $c_{i,j}$ for some $(j, i) \in E$ or $u_{k'}$ for some $k' < k$ such that $\omega \leq \Omega$ and for every Jacobian entry there is some identical u_k , $k \leq \omega$?*

We might view the collection of accumulation statements as an arithmetic circuit A_J called the *accumulation circuit*. To avoid confusion, we will refer to the elements of A_J as nodes, and elements of G as vertices. A_J is a directed acyclic graph whose nodes have either 0 or 2 inedges. Nodes with 0 inedges correspond to the local partials $c_{i,j}$, and the other nodes are *gates* labeled with operations from $\{+, *\}$. The nodes without outedges correspond to the nonzero entries in the Jacobian J . We may interpret OJA as the problem of finding an accumulation circuit with minimum size, where the size of A_J is the number of gates.

The *underlying graph* of a dag G is the undirected graph obtained by removing the direction from all the edges.

3 Series-parallel dags

Definition 1 (Two-terminal series-parallel (TTSP)).

- (i) A dag consisting of a single edge is TTSP.
- (ii) series composition: Let G_1 and G_2 be TTSP dags. The dag that results from identifying the input of G_1 with the output of G_2 is TTSP.
- (iii) parallel composition: Let G_1 and G_2 be TTSP dags, at least one of which contains more than one edge. The dag that results from identifying the input of G_1 with the input of G_2 and the output of G_1 with the output of G_2 is TTSP.

This recursive definition is the definition in [1] of *edge series-parallel multidigraphs*. Note that we don't allow parallel edges in LCGs; for example, the graph formed by performing a parallel composition between two isolated edges would not be an LCG.

Definition 2 (Output-terminal series-parallel (OTSP)).

- (i) A dag G is OTSP if G is TTSP.
- (ii) series composition: Let G_1 be a TTSP dag and let G_2 be a OTSP dag. The dag that results from identifying the input of G_1 with the output of G_2 is OTSP.
- (iii) output composition: Let G_1 and G_2 be OTSP dags. The dag that results from identifying the output of G_1 with the output of G_2 is OTSP.

The class of input-terminal series-parallel dags (ITSP) is defined analogously. We say that a dag is terminal series-parallel (TSP) if it is OTSP or ITSP.

Throughout the remainder of this paper, we will assume that the underlying graph of G is connected; it is clear that each connected component can be treated separately. Furthermore, we assume that all TSP dags are either TTSP or have been constructed with the last operation being a series composition. In case of a TSP dag G that is not TTSP and the final construction operation was an output composition (or input composition) with G_1 and G_2 , we may treat G_1 and G_2 separately, as no path in G can contain edges from both G_1 and G_2 , thus any JAC for G will consist of multiple disjoint components.

3.1 Example

The optimal vertex elimination sequences for the LCG shown in Figure 2(a) are $(3, 2, 4, 1)$, $(2, 4, 3, 1)$, and $(2, 3, 4, 1)$. Each has a cost of five multiplications.

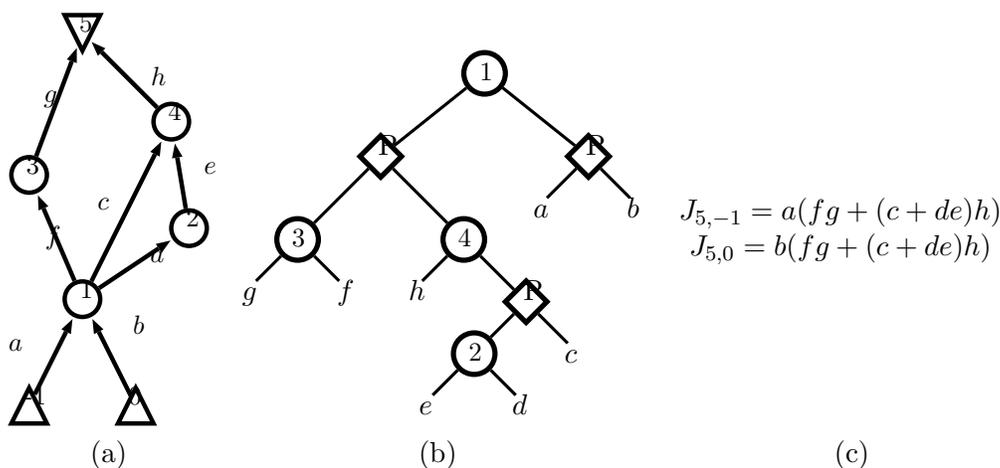


Figure 2: A TTSP dag G (a) and the corresponding decomposition tree T (b).

4 Solving OJA on TSP dags

The following algorithm is *robust* [4] in the sense that given any dag G it either finds an optimal JAC or provides a certificate that G is not TSP.

Algorithm 1.INPUT: A dag G .OUTPUT: A Jacobian accumulation circuit A_J if G is TSP, the reply NO if G is not TSP.

1. Return NO if G has more than one input or more than one output.
2. Eliminate all intermediate vertices v with $|P_v| * |S_v| = 1$, resulting in a new graph G' .
3. Check whether G' has the form of a rooted tree. If not, return NO.
4. Eliminate all intermediate vertices v from G' as follows.
 - If G' is OTSP, then recursively eliminate all predecessors of the sole output vertex.
 - If G' is ITSP, then recursively eliminate all successors of the sole input vertex.

Lemma 1. *Algorithm 1 will return NO if and only if G is not TSP.*

Proof. (\Rightarrow): It is obvious that G is not TSP if it contains more than input and more than one output. Let G' be the dag that results from eliminating all vertices with one inedge and one outedge, where the algorithm returns NO because G' does not have the form of a rooted tree. Note that all intermediate vertices in G' have either more than one inedge or more than one outedge, and there is at least one semicycle.

(\Leftarrow): Assume G is not TSP, and G has either one input or one output (otherwise, the algorithm would return NO trivially). Without loss of generality, assume G has only one output. □

4.1 Polytree accumulation circuits

A dag G is a *polytree* if its underlying graph is a tree (an undirected graph without cycles).

Definition 3 (semi-valid Jacobian Accumulation Circuit). *a semi-valid JAC for a dag G is a directed acyclic graph where every node has exactly zero or two children. Let x be some input in G and let y be some output in G , then the Jacobian node xy must be reachable from all local partial nodes that correspond to edges that occur on some path from x to y in G . In other words, we don't associate the non-minimal nodes with particular operations, and we relax the requirement that A_J correctly computes the Jacobian.*

Obviously,

Theorem 1 (characterization of TSP dags). *A dag G is TSP if and only if G has a Jacobian accumulation circuit that is a polytree.*

Proof. (\Rightarrow) Let G be a TSP dag. We will show that any Jacobian accumulation circuit A_J produced by Algorithm 1 will be a polytree. □

(\Leftarrow) Let A_J be a polytree JAC... □

Lemma 2. *Let $G = (V, E)$ be a TTSP dag. Then all polytree JACs A_J for G have size $|E| - 1$.*

Proof. □

Lemma 3. *Let $G = (V, E)$ be a TSP dag. Then all polytree JACs A_J for G have the same size.*

Proof. If G is TTSP, then all polytree JACs for G have size $|E| - 1$ by Lemma 2.

If G is not TTSP, then assume G was constructed by the series composition of some TTSP dag $G_1 = (V_1, E_1)$ and some OTSP dag $G_2 = (V_2, E_2)$ where G_2 has more than one input. Since the local partials in E_1 contribute to \square

Lemma 4. *Let G be a TSP dag, and let A_J be a JAC for G . Then A_J has minimum size if and only if A_J is a polytree.*

Proof. We will show that any JAC for G that is not a polytree is strictly larger than any polytree JAC for G . We use the fact that Algorithm 1 always produces a polytree JAC that correctly accumulates the Jacobian whenever the input graph G is TSP.

(\Rightarrow) Suppose that A_J contains two distinct paths $P_a = (u = a_0, a_1, \dots, a_i = v), P_b = (u = b_0, b_1, \dots, b_j = v)$ that honor the orientation on the edges, where it is assumed without loss of generality that $a_1 \neq v$. Let S be the set of all parents of a_1 and let t be the child of a_1 that is not u . Observe that every $s \in S$ can be a parent of u or t , but not both. Consider the JAC A'_J that results from removing a_1 and adding a new inedge for every $s \in S$ from either u or t , such that every $s \in S$ has exactly two inedges. It is clear that A'_J has fewer nodes than A_J , contradicting the assumed minimality of A_J .

Any cycle in the underlying graph of A_J that is not of the above type will violate the orientation on the edges in some way.

(\Leftarrow) For the sake of contradiction, assume that A_J is not a polytree. \square

The combination of Theorem 1 and Lemma 4 implies the following theorem.

Theorem 2. *Algorithm 1 solves OJA for TSP dags.*

Proof. ... \square

5 General Series-Parallel Dags

Further generalizing to general series-parallel graphs, we find that JACs derived from the decomposition tree can lead to sub-optimal accumulation sequences. This graph also illustrates the fact that the Markowitz heuristic is not optimal for all series-parallel graphs.

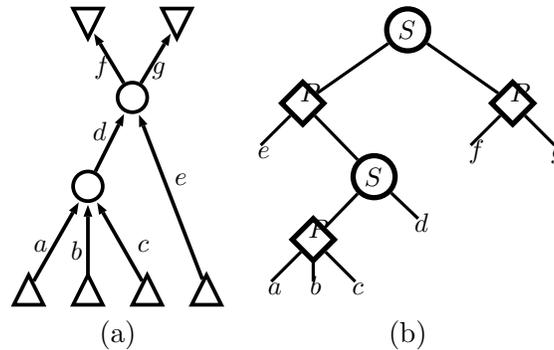


Figure 3: a SP LCG (a) and its canonical decomposition tree (b)

6 Conclusions

It has not yet been shown that OPTIMAL JACOBIAN ACCUMULATION = OPTIMAL FACE ELIMINATION, and it is clear that this is *not* the case when algebraic dependences between the local partial derivatives are allowed.

It is possible that elimination techniques applied to the comparability graph of P could be shown to be general enough to encompass all of OJA, and this could be used to show that certain algorithms solve OJA for particular subclasses of LCGs.

Observation 1. *Let G be a SP dag. Then G is a polytree if G is absorption-free.*

Future research:

- OJA on polytree SP dags.
- OJA on (general) SP dags

References

- [1] Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22:317–330, 1983.
- [2] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
- [3] Andreas Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, PA, 2000.
- [4] Vijay Raghavan and Jeremy Spinrad. Robust algorithms for restricted domains. *J. Algorithms*, 48(1):160–172, 2003.