Using Directed Edge Separators to Increase Efficiency in the Determination of Jacobian Matrices via Automatic Differentiation *

Thomas F. Coleman, Xin Xiong, and Wei Xu

Abstract Every numerical function evaluation can be represented as a directed acyclic graph (DAG), beginning at the initial input variable settings, and terminating at the output or corresponding function value(s). The "reverse mode" of automatic differentiation (AD) generates a "tape" which is a representation of this underlying DAG. In this work we illustrate that a directed edge separator in this underlying DAG can yield space and time efficiency gains in the application of AD. Use of directed edge separators to increase AD efficiency in different ways than proposed here has been suggested by other authors [2, 4]. In contrast to these previous works, our focus here is primarily on space. Furthermore, we explore two simple algorithms to find good directed edge separators, and show how these ideas can be applied recursively to great advantage. Initial numerical experiments are presented.

Key words: Automatic differentiation, reverse mode, adjoint method, directed acyclic graph, computational graph, edge separator, Jacobian matrix, Ford-Fulkerson algorithm, minimum cutset, Newton step

---

Thomas F. Coleman
Department of Combinatorics and Optimization, University of Waterloo, On. Canada, N2L 3G1. tfcoleman@uwaterloo.ca

Xin Xiong
Department of Combinatorics and Optimization, University of Waterloo, On. Canada, N2L 3G1. x6xiong@uwaterloo.ca

Wei Xu
Department of Mathematics, Tongji University, Shanghai, China, 200092. wdxu@tongji.edu.cn

* This work was supported in part by Ophelia Lazaridis University Research Chair (held by Thomas F. Coleman), the National Sciences and Engineering Research Concil of Canada and the Natural Science Foudation of China (Project No: 11101310).

1 Introduction

Many scientific and engineering computations require the repeated calculation of matrices of derivatives. The repeated calculation of these derivative matrices often represents a significant portion of the overall computational cost of the overall computation.

Automatic differentiation (AD) can deliver matrices of derivatives given a source code to evaluate the function F (or in the case of minimization, the objective function f). Good methods that exploit sparsity, constant values, or duplicate values, have also been developed, e.g. [3, 16]. In addition, if the objective function exhibits certain kinds of structures, and this structure is conveniently noted in the expression of the objective function, then the efficiency of the automatic differentiation process can be greatly enhanced [1, 6, 7, 9, 11, 14].

This paper is concerned with the case where the problem structure is *not noted* á priori and AD may subsequently be regarded as too costly either in time or space.

1.1 Automatic Differentiation and The Edge Separator

Let us consider a nonlinear mapping

$$F : \mathbb{R}^n \mapsto \mathbb{R}^m$$

where $F(x) = [f_1(x), \cdots, f_m(x)]^T$, and each component function $f_i : \mathbb{R}^n \mapsto \mathbb{R}^1$ is differentiable. The Jacobian matrix $J(x)$ is the $m \times n$ matrix of first derivatives: $J_{ij} = \frac{\partial f_i}{\partial x_j} (i = 1, \cdots, m; j = 1, \cdots, n)$. Given the source code to evaluate $F(x)$, automatic differentiation can be used to determine $J(x)$. Generally, the work required to evaluate $J(x)$ via a *combination* of the forward and reverse modes of AD, and in the presence of sparsity in $J(x)$, is propotional to $\chi_B(G^D(J)) \cdot \omega(F)$ where χ_B is the bi-chromatic number of the double intersection graph $G^D(J)$, and $\omega(\cdot)$ is the work required, (i.e., number of basic computational steps) to evaluate $F(x)$ -see [9]. We note that when reverse mode AD is invoked the space required to compute the Jacobian is proportional to $\omega(F)$, and this can be prohibitively large. If AD is restricted to forward mode then the space required is much less, i.e., it is proportional to $\sigma(F)$, the space required to evaluate $F(x)$, and typically $\omega(F) \gg \sigma(F)$; however, forward mode alone can be much more costly than a combination of forward and reverse modes. [9, 11]

Consider now the (directed) computational graph that represents the structure of the program to evaluate $F(x)$:

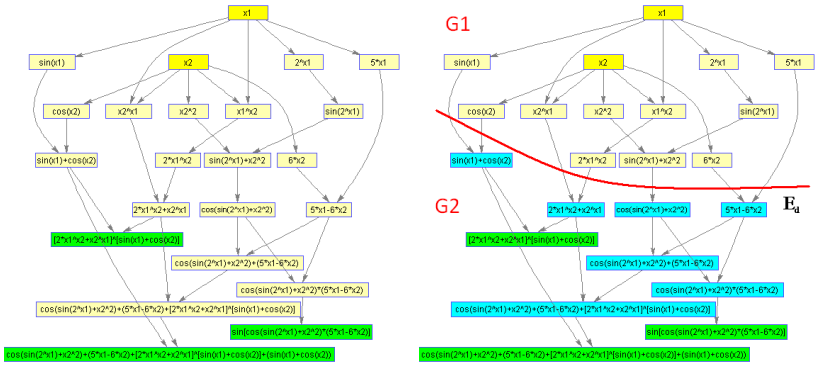
$$\mathbf{G}(F) = (V, \mathbf{E}) \tag{1}$$

where V consists of three sets of vertices. Specifically, $V = \{V_x, V_y, V_z\}$ where vertices in V_x represent the input variables; a vertex in V_y represent ***both*** a basic or

elementary *operation* receiving one or two inputs, producing a single ouput variable ***and*** the output *intermediate variable*; vertices in V_z represent the output variables. So input variable x_i corresponds to vertex $v_{x_i} \in V_x$, intermediate variable y_k corresponds to vertex $v_{y_k} \in V_y$, and output $z_j = [F(x)]_j$ corresponds to vertex $v_{z_j} \in V_z$. Note that the number of vertices in V_y, i.e., $|V_y|$, is the number of basic operations required to evaluate $F(x)$. Hence $\omega(F) = |V_y|$.

The edge set $\mathbf{E}$ represents the traffic pattern of the variables. For example, there is a directed edge $e_k = (v_{y_i}, v_{y_j}) \in \mathbf{E}$ if intermediate variable y_i is required by computational node v_{y_j} to produce intermediate variable y_j. If $e_k = (v_{y_i}, v_{y_j}) \in \mathbf{E}$ is a directed edge from vertex v_{y_i} to vertex v_{y_j} then we refer to vertex v_{y_i} as the *tail node* of edge e_k and vertex v_{y_j} as the *head node* of edge e_k. It is clear that if F is well-defined then $\mathbf{G}(F)$ is an acyclic graph.

Definition 1. $E_d \subset \mathbf{E}$ is a *directed edge separator* in directed graph $\mathbf{G}$ if $\mathbf{G} - \{E_d\}$ consists of disjoint components $\mathbf{G}_1$ and $\mathbf{G}_2$ where all edges in E_d have the same orientation relative to $\mathbf{G}_1$, $\mathbf{G}_2$.



(a) A computational graph G

(b) An example of graph G's directed edge separator E_d

Fig. 1 An example of computational graphs and a sample directed edge separator

Suppose $E_d \subset \mathbf{E}_y$ is an edge separator of the computational graph $\mathbf{G}(F)$ with orientation forward in time. Then the nonlinear function $F(x)$ can be broken into two parts:

$$\left.\begin{array}{l} \text{solve for } y\text{: } F_1(x,y) = 0 \\ \text{solve for } z\text{: } F_2(x,y) - z = 0 \end{array}\right\} \tag{2}$$

where y is the vector of intermediate variables defined by the *tail vertices* of the edge separator E_d, and z is the output vector, i.e., $z = F(x)$. Let p be the number of

tail vertices of edge set E_d, i.e., $y \in \mathbb{R}^p$. Note: $|E_d| \geq p$. The nonlinear function F_1 is defined by the computational graph above E_d, i.e., G_1, and nonlinear function F_2 is defined by the computational graph below E_d, i.e., G_2. See Figure 1(b). We note that the system (1) can be differentiated wrt (x,y) to yield an 'extended' Jacobian matrix [8, 13]:

$$J_E \stackrel{\Delta}{=} \begin{bmatrix} (F_1)_x & (F_1)_y \\ (F_2)_x & (F_2)_y \end{bmatrix} \tag{3}$$

Since y is a well-defined unique output of function $F_1 : \mathbb{R}^{n+p} \mapsto \mathbb{R}^p$, $(F_1)_y$ is a $p \times p$ nonsingular matrix. The Jacobian of F is the Schur-complement of (2), i.e.,

$$J(x) = (F_2)_x - (F_2)_y (F_1)_y^{-1} (F_1)_x \tag{4}$$

There are two important computational issues to note. The first is that the work to evaluate J_E is often less than that required to evaluate $J(x)$ directly. The second is that less space is often required to calculate and save J_E relative to calculating and saving J directly by AD (when the AD technique involves the use of "reverse mode" as in the bi-coloring technique).

It is usually less expensive, in time and space, to compute $J_E(x)$ rather than $J(x)$, using a combination of forward and reverse modes of automatic differentiation[10]. However, what is the utility of $J_E(x)$? The answer is that $J_E(x)$ can often be used directly to simulate the action of J and this computation can often be less expensive (due to sparsity in J_E that is not present in J) than explicitly forming and using J. For example, the Newton system 'solve $Js = -F$' can be replaced with

$$\text{solve } J_E \begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} 0 \\ -F \end{bmatrix}. \tag{5}$$

The main points are that calculating matrix J_E can be less costly than calculating matrix J, and solving (5) can also be relatively inexpensive given sparsity that can occur in J_E that may not be present in J.

The ideas discussed above can be generalized to the case with multiple mutually independent directed edge separators, $E_{d_1}, \cdots, E_{d_k} \in \mathbf{E}$, where we assume $G - \{E_{d_1}, \cdots, E_{d_k}\} = \{G_1, \cdots, G_{k+1}\}$. The connected graphs $G_1, \cdots, G_{k+1}$ are pairwise disjoint and are ordered such that when evaluating F, G_i can be fully evaluated before $G_{i+1}, i = 1, \cdots, k$.

Suppose $E_{d_1}, \cdots, E_{d_k} \in \mathbf{E}$ are pairwise disjoint separators of the computational graph $\mathbf{G}(F)$ with orientation forward in time (as indicated above). Then the evaluation of nonlinear function $F(x)$ can be broken into $k+1$ steps:

$$\left.\begin{array}{rc} \text{solve for } y_1: & F_1(x, y_1) = 0 \\ \vdots & \vdots \\ \text{solve for } y_k: & F_k(x, y_1, \cdots, y_k) = 0 \\ \text{solve for } z: & F_{k+1}(x, y_1, \cdots, y_k) - z = 0 \end{array}\right\} \tag{6}$$

where y_i is the vector of intermediate variables defined by the *tail vertices* of the edge separator E_{d_i}, for $i = 1, \cdots, k+1$ and z is the output vector, i.e., $z = F(x)$.

2 On Finding Separators to Increase Efficiency in the Application of Automatic Differentiation

In Section 1.1 we observed that if a small directed edge separator divides the computational graph G into roughly two equal components G_1 and G_2, then the space requirements are minimized (roughly halved). Moreover, the required work will not increase, and due to inreasing sparsity, will likely decrease.

Therefore, our approach is to seek a small directed edge separator that will (roughly) bisect the fundamental computational graph. In this section, we present two algorithms to find good separators.

2.1 Minimum Weighted Separator

This minimum weighted separator approach is based on the Ford Fulkerson (FF) algorithm [12], a well known max-flow/min-cut algorithm. The Ford Fulkerson algorithm finds the minimum $s-t$ cut, a set of edges whose removal separates specified node s and node t, two arbitary nodes in the graph. A minimum cut does not always correspond to a directed separator; we "post process" the min-cut solution to obtain a directed separator.

We desire that the determined separator (roughly) divide the fundamental computational graph in half. To add this preference into the optimization, we assign capacities to edges to reflect distance from the input or output nodes, whichever is closer. With this kind of weight distribution, a 'small' cut will likely be located towards the middle of the fundamental computational graph.

To determine the weights we first calculate depth of nodes and edges.

Definition 2. We define the *depth* of a node v in a DAG to be the shorter of shortest directed path from an input node (source) to v and the shortest directed path from v to an output node (sink). We define the *depth* of an edge y in a DAG in an analogous fashion.

Our proposed method is as follows:

1. Assign weights to edges to reflect depth of an edge.
2. Solve the weighted mincut problem, e.g. using the Ford-Fulkerson method.
3. If the cut is not a directed separator, modify according to Algorithm 1.

Algorithm 1 *Let $E \subset \mathbf{E}$ such that graph $\mathbf{G} - E$ consists of two components $\mathbf{G}_1$ and $\mathbf{G}_2$, where source nodes are in $\mathbf{G}_1$ and sink nodes are in $\mathbf{G}_2$. If E is not a directed*

separator, then E contains both edges from $\mathbf{G}_2$ to $\mathbf{G}_1$ and edges from $\mathbf{G}_1$ to $\mathbf{G}_2$. Let $S = V(\mathbf{G_1})$ and $T = V(\mathbf{G_2})$. A directed separator (S,T) can be generated either by moving tail nodes of $T \to S$ edges from T to S recusively, or by moving head nodes of $T \to S$ edges from S to T recursively. The formal description is stated as follows:

1. $T_1 \leftarrow \{v : v \in T\} \cup \{v :$ *there exists a directed uv-path in* $\mathbf{G}, u \in T\}$.
2. $S_1 \leftarrow V(\mathbf{G}) - T_1, E_1 = E(\mathbf{G}) - E(\mathbf{G}(S_1)) - E(\mathbf{G}(T_1))$.
3. $S_2 \leftarrow \{v : v \in S\} \cup \{v :$ *there exists a directed vu-path in* $\mathbf{G}, u \in S\}$.
4. $T_2 \leftarrow V(\mathbf{G}) - S_2, E_2 = E(\mathbf{G}) - E(\mathbf{G}(S_2)) - E(\mathbf{G}(T_2))$.
5. *Pick the smaller between E_1 and E_2 as the desired separator.*

2.2 Natural Order Edge Separator

A second method to generate directed separators comes from the observation that if the 'tape' generated by reverse-mode AD is snipped at any point then effectively a directed separator is located.

Suppose we are given a compuational graph $\mathbf{G}$ and the correponding computational tape T with length $|V(\mathbf{G})|$. A natural partition ($\mathbf{G}_1$,$\mathbf{G}_2$) of $\mathbf{G}$ is $\mathbf{G}_1 = \mathbf{G}(T(1 : i))$, $\mathbf{G}_2 = \mathbf{G}(T(i+1 : |V(\mathbf{G})|))$, where i is some integer between 1 and $|V(\mathbf{G})| - 1$. Since cells in the tape are in chronological order, all basic operations represented in $\mathbf{G}_1$ are evaluated before those represented in $\mathbf{G}_2$, therefore all edges between $\mathbf{G}_1$, $\mathbf{G}_2$ are directed from $\mathbf{G}_1$ to $\mathbf{G}_2$. Since these edges form a directed edge separator, we can then choose i to get the preferred edge separator in terms of separator size and partition ratio.

Multiple Separators

Either of the proposed directed separator methods can be applied, recursively, to yield multiple separators. We do exactly this in our code and in our computational experiments below, always working on the largest remaining subgraph (details will be provided in [10]).

3 Experiments

In this section we provide computational results on some preliminary experiments to automatically reveal 'AD-useful' structure using the separator idea. These experiments are based on the minimum weighted separator algorithm and natural order separator algorithm described in previous section, to find directed edge separators that bisect the fundamental computational graph.

We use the AD-tool, ADMAT [5], to generate the computational graphs. However, for efficiency reasons, ADMAT sometimes condenses the fundamental computational graph to produce a condensed computational graph. In a condensed computatonal graph nodes may represent matrix operations such as matrix-multiplication. Therefore our weighting heuristic is adjusted to account for this.

In our numerical experiments we focus on two types of structures that represent the two shape extreme cases.

3.1 Thin Computational Graphs

A function involving recusive iterations usually produces a "thin" computational graph.

Example. Define

$$F\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \begin{bmatrix} x_3 \cdot \cos(\sin(2^{x_1} + x_2^2)) \\ 5x_1 - 6x_2 \\ 2x_2^{x_2} + x_2^{x_1} \end{bmatrix} \tag{7}$$

and

$$F_1 = F \circ F \circ F \circ F \circ F$$

Note that F_1's computational graph is long and narrow (i.e. 'thin').

After three interations, three separators in Figure 2 are found. The graph is divided into four subgraphs. Visually, these edge separator are good in terms of size and evenly dividing the graph.

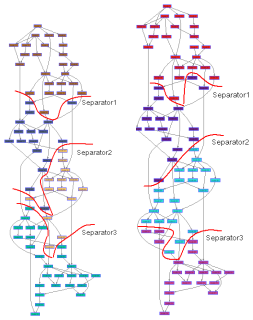
3.2 Fat Computational Graphs

A "fat" computational graph is produced when macro-computations are independent of each other. A typical example is:

$$F_2 = \sum_{i=1}^{6} F(x + \text{rand}_i(3,1))$$

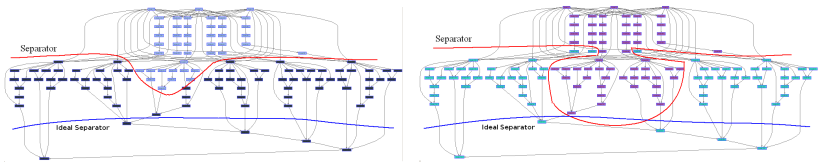
where F is defined by equation (7) in the previous experiment.

The separators found by our two algorithms on this example are useful but are less than ideal in contrast to the separators found in the "long thin" class. Additional experiments using different weighting schemes, are ongoing.



(a) Minimum Weighted Separator (b) Natural Order Separator

Fig. 2 Obtained separators of F_1's condensed computational graph by the two different algorithms



(a) Minimum Weighted Separator (b) Natural Order Separator

Fig. 3 Obtained separators of F_2's condensed computational graph by the two different algorithms

4 Accelerating the Calculation of the Jacobian matrix

To illustrate how separators accelerate computation, we construct the following numeric example:

Let

$$f\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \begin{bmatrix} \frac{x_2+3x_3}{4} \\ \sqrt{x_1 x_3} \\ \frac{x_1+2x_2+x_3}{4} \end{bmatrix}$$

and

$$F_k = f \circ f \circ \cdots \circ f \qquad \text{where there are } k \text{ } f\text{'s} \tag{8}$$

It is obvious that $F_n \equiv F_{k_1} \circ F_{k_2} \circ \cdots \circ F_{k_m}$ provided $n = \sum_{i=1}^{m} k_i$.

We calculate the Jacobian matrix $J \in \mathbb{R}^{3\times 3}$ of $F_{2400}(x_0)$ at $x_0 = [6,9,3]^T$. We will use ADMAT reverse mode to obtain J both directly and by constructing directed separators.

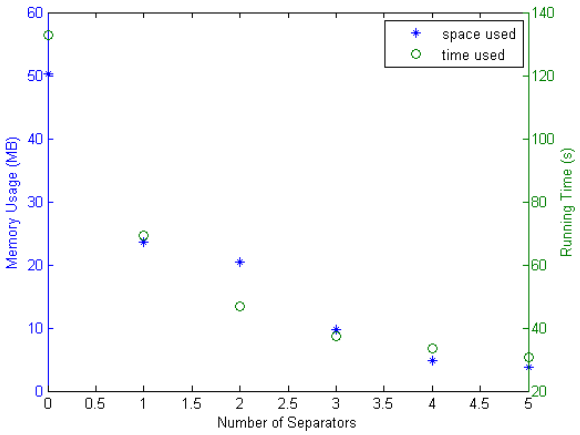


Fig. 4 Acceleration of separator method

The performance plot in Figure 4 does not count in time used to locate separators. The 'running time' refers to the time used to obtain Jacobian matrix once the separators are found. Work is ongoing to perform the separator determine step efficiently, in space and time. We note that this separator structure can (typically) be re-used over many iterations.

5 Concluding Remarks

Our initial experiments and analysis indicate that separation of nonlinear systems with use of directed separators can significantly reduce the space and time requirements. Directed separators have also been proposed to improve the performance of hierarchical preaccumulation strategies[2, 15]. Issues to be investigated include:

- The amortization remarks above assume that the structure of F is invariant with x. This is not always the case.
- To further reduce memory usage, we are investigating use of an "online" algorithm, i.e., generation of separators with only partial information.

References

1. Bischof, C.H., Bouaricha, A., Khademi, P., Moré, J.J.: Computing gradients in large-scale optimization using automatic differentiation. INFORMS J. Computing **9**, 185–194 (1997)
2. Bischof, C.H., Haghighat, M.R.: Hierarchical approaches to automatic differentiation. In: M. Berz, C. Bischof, G. Corliss, A. Griewank (eds.) Computational Differentiation: Techniques, Applications, and Tools, pp. 83–94. SIAM, Philadelphia, PA (1996)
3. Bischof, C.H., Khademi, P.M., Bouaricha, A., Carle, A.: Efficient computation of gradients and Jacobians by dynamic exploitation of sparsity in automatic differentiation. Optimization Methods and Software **7**, 1–39 (1997)
4. Bücker, H.M., Rasch, A.: Modeling the performance of interface contraction. ACM Transactions on Mathematical Software **29**(4), 440–457 (2003). DOI http://doi.acm.org/10.1145/962437.962442
5. Cayuga Research Associates, L.: Admat-2.0 users guide (2009). URL `http://www.cayugaresearch.com/`
6. Coleman, T.F., Jonsson, G.F.: The efficient computation of structured gradients using automatic differentiation. SIAM Journal on Scientific Computing **20**(4), 1430–1437 (1999). DOI 10.1137/S1064827597320794
7. Coleman, T.F., Santosa, F., Verma, A.: Efficient calculation of Jacobian and adjoint vector products in wave propagational inverse problem using automatic differentiation. J. Comp. Phys. **157**, 234–255 (2000)
8. Coleman, T.F., Verma, A.: Structure and efficient Jacobian calculation. In: M. Berz, C. Bischof, G. Corliss, A. Griewank (eds.) Computational Differentiation: Techniques, Applications, and Tools, pp. 149–159. SIAM, Philadelphia, PA (1996)
9. Coleman, T.F., Verma, A.: The efficient computation of sparse Jacobian matrices using automatic differentiation. SIAM J. Sci. Comput. **19**(4), 1210–1233 (1998). DOI 10.1137/S1064827595295349. URL `http://link.aip.org/link/?SCE/19/1210/1`
10. Coleman, T.F., Xiong, X.: New graph approaches to the determination of Jacobian and Hessian matrices, and Newton steps, via automatic differentiation (in preparation)
11. Coleman, T.F., Xu, W.: Fast (structured) Newton computations. SIAM Journal on Scientific Computing **31**(2), 1175–1191 (2008). DOI 10.1137/070701005. URL `http://link.aip.org/link/?SCE/31/1175/1`
12. Ford, L., Fulkerson, D.: Maximal flow through a network. Canadian Journal of Mathematics **8**, 399–404 (1956)
13. Griewank, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. No. 19 in Frontiers in Appl. Math. SIAM, Philadelphia, PA (2000)
14. Rall, L.B.: Automatic Differentiation: Techniques and Applications, *Lecture Notes in Computer Science*, vol. 120. Springer, Berlin (1981). DOI 10.1007/3-540-10861-0
15. Tadjouddine, E.M.: Vertex-ordering Algorithms for Automatic Differentiation of Computer Codes. The Computer Journal **51**(6), 688–699 (2008). DOI 10.1093/comjnl/bxm115. URL `http://comjnl.oxfordjournals.org/cgi/content/abstract/51/6/688`
16. Xu, W., Coleman, T.F.: Efficient (Partial) Determination of Derivative Matrices via Automatic Differentiation (To appear in SIAM journal on Scientific Computing, 2012)