Second Order Reverse Mode of AD : A Vertex Elimination Perspective

Mu Wang, Alex Pothen and Paul Hovland

Computer Science, Purdue University
MCS Division, Argonne National Lab

Thanks : NSF, DOE, Intel

October 10, 2016

PURDUE UNIVERSITY™

Outline

- Second order reverse mode of Automatic Differentiation
- Vertex elimination for evaluating the Gradient and the Hessian
- The correspondence between second order reverse mode and vertex elimination
- Discussion and board picture

AD Fundamentals

- Automatic Differentiation (AD) is a technique that augments a computer program so that the augmented program computes the derivatives as well as the values of the function defined by the original program.
- Scalar Objective Function $f : \mathcal{R}^n \to \mathcal{R}^1$
  - Implemented as a computer program
  - The evaluation is on a sequence of decomposed elemental functions

$$\text{For } k = 1, 2, \cdots, l$$
$$v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$$

AD Fundamentals

- Automatic Differentiation (AD) is a technique that augments a computer program so that the augmented program computes the derivatives as well as the values of the function defined by the original program.
- Scalar Objective Function $f : \mathcal{R}^n \rightarrow \mathcal{R}^1$
  - Implemented as a computer program
  - The evaluation is on a sequence of decomposed elemental functions

$$\text{For } k = 1, 2, \cdots, l$$
$$v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$$

AD Fundamentals

- Automatic Differentiation (AD) is a technique that augments a computer program so that the augmented program computes the derivatives as well as the values of the function defined by the original program.
- Scalar Objective Function $f : \mathcal{R}^n \rightarrow \mathcal{R}^1$
  - Implemented as a computer program
  - The evaluation is on a sequence of decomposed elemental functions

$$\text{For } k = 1, 2, \cdots, l$$
$$v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$$

- y = pow(pow(x*x, 2.0), x), $\qquad (x > 0, y = x^{4x})$
  - $v_0 <<= x$
  - $v_1 = \varphi_1(v_0) = v_0 * v_0$
  - $v_2 = \varphi_2(v_1) = pow(v_1, 2.0)$
  - $v_3 = \varphi_3(v_2, v_0) = pow(v_2, v_0)$
  - $v_3 >>= y$

AD Fundamentals

- Automatic Differentiation (AD) is a technique that augments a computer program so that the augmented program computes the derivatives as well as the values of the function defined by the original program.
- Scalar Objective Function $f : \mathcal{R}^n \rightarrow \mathcal{R}^1$
  - Implemented as a computer program
  - The evaluation is on a sequence of decomposed elemental functions

$$\text{For } k = 1, 2, \cdots, l$$
$$v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$$

- Indexing convention :
  - Independent variables : $v_{1-n}, \cdots, v_0$
  - Intermediate variables : $v_1, \cdots, v_{l-1}$
  - Dependent variable : v_l

Second Order Reverse Mode : Story Line

- First Proposed by Gower and Mello[1]
  - Called `Edge_Pushing` initially
  - From the closed form of second order derivative for composite functions
- Wang, Gebremedhin, and Pothen provided a second perspective by adopting *live variable* analysis [2] from compiler theory.
  - Better complexity bound
  - Correct Implementation
  - Further improved with preaccumulation
- The new proof can be extended into general high orders.

[1] Gower, Robert Mansel, and Margarida P. Mello. Hessian matrices via automatic differentiation. Universidade Estadual de Campinas, Instituto de Matemtica, Estatstica e Computao Cientfica, 2010.

[2] Wang, Mu, Assefaw Gebremedhin, and Alex Pothen. "Capitalizing on live variables: new algorithms for efficient Hessian computation via automatic differentiation." Mathematical Programming Computation (2016): 1-41.

Second Order Reverse Mode : Story Line

- First Proposed by Gower and Mello[1]
  - Called `Edge_Pushing` initially
  - From the closed form of second order derivative for composite functions
- Wang, Gebremedhin, and Pothen provided a second perspective by adopting *live variable* analysis [2] from compiler theory.
  - Better complexity bound
  - Correct Implementation
  - Further improved with preaccumulation
- The new proof can be extended into general high orders.

---

[1] Gower, Robert Mansel, and Margarida P. Mello. Hessian matrices via automatic differentiation. Universidade Estadual de Campinas, Instituto de Matemtica, Estatstica e Computao Cientfica, 2010.

[2] Wang, Mu, Assefaw Gebremedhin, and Alex Pothen. "Capitalizing on live variables: new algorithms for efficient Hessian computation via automatic differentiation." Mathematical Programming Computation (2016): 1-41.

Second Order Reverse Mode : Story Line

- First Proposed by Gower and Mello[1]
  - Called `Edge_Pushing` initially
  - From the closed form of second order derivative for composite functions
- Wang, Gebremedhin, and Pothen provided a second perspective by adopting *live variable* analysis [2] from compiler theory.
  - Better complexity bound
  - Correct Implementation
  - Further improved with preaccumulation
- The new proof can be extended into general high orders.

---

[1] Gower, Robert Mansel, and Margarida P. Mello. Hessian matrices via automatic differentiation. Universidade Estadual de Campinas, Instituto de Matemtica, Estatstica e Computao Cientfica, 2010.

[2] Wang, Mu, Assefaw Gebremedhin, and Alex Pothen. "Capitalizing on live variables: new algorithms for efficient Hessian computation via automatic differentiation." Mathematical Programming Computation (2016): 1-41.

Reverse Mode of AD

- Function evaluation : evaluate each elemental function

$$\text{for } k = 1, 2, \cdots, l$$
$$v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$$

- Reverse mode of AD : process sequence of elemental functions in reverse order

$$\text{for } k = l, l-1, \cdots, 1$$
$$\text{do something with } v_k = \varphi_k(v_i)_{\{v_i \prec v_k\}}$$

- Equivalent function $f_k(S_k)$: a function defined by the elemental functions $\varphi_l, \cdots, \varphi_k$ that have been processed at the end of step k, in reverse mode
  - $f = \underbrace{\varphi_l \circ \cdots \circ \varphi_k}_{f_k(S_k)} \circ \varphi_{k-1} \circ \cdots \circ \varphi_1.$
  - The independent variables of f_k are denoted by S_k.

Reverse Mode of AD

- Function evaluation : evaluate each elemental function

$$\text{for } k = 1, 2, \cdots, l$$
$$v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$$

- Reverse mode of AD : process sequence of elemental functions in reverse order

$$\text{for } k = l, l-1, \cdots, 1$$
$$\text{do something with } v_k = \varphi_k(v_i)_{\{v_i \prec v_k\}}$$

- Equivalent function $f_k(S_k)$: a function defined by the elemental functions $\varphi_l, \cdots, \varphi_k$ that have been processed at the end of step k, in reverse mode
  - $f = \underbrace{\varphi_l \circ \cdots \circ \varphi_k}_{f_k(S_k)} \circ \varphi_{k-1} \circ \cdots \circ \varphi_1.$
  - The independent variables of f_k are denoted by S_k.

Reverse Mode of AD

- Function evaluation : evaluate each elemental function

$$\text{for } k = 1, 2, \cdots, l$$
$$v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$$

- Reverse mode of AD : process sequence of elemental functions in reverse order

$$\text{for } k = l, l-1, \cdots, 1$$
$$\text{do something with } v_k = \varphi_k(v_i)_{\{v_i \prec v_k\}}$$

- Equivalent function $f_k(S_k)$: a function defined by the elemental functions $\varphi_l, \cdots, \varphi_k$ that have been processed at the end of step k, in reverse mode
    - $f = \underbrace{\varphi_l \circ \cdots \circ \varphi_k}_{f_k(S_k)} \circ \varphi_{k-1} \circ \cdots \circ \varphi_1.$
    - The independent variables of f_k are denoted by S_k.

Reverse Mode of AD

For $k = l, l-1, \cdots, 1$

do something with $v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$

- $f_k(S_k) = f_{k+1}(S_{k+1} \setminus \{v_k\}, v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}})$

Reverse Mode of AD

For $k = l, l-1, \cdots, 1$

do something with $v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$

- $f_k(S_k) = f_{k+1}(S_{k+1} \setminus \{v_k\}, v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}})$

$$f = \overbrace{\varphi_l \circ \cdots \circ \varphi_{k+1}}^{f_{k+1}(S_{k+1})} \circ \varphi_k \circ \varphi_{k-1} \circ \cdots \circ \varphi_1$$

$$f = \underbrace{\varphi_l \circ \cdots \circ \varphi_{k+1} \circ \varphi_k}_{f_k(S_k)} \circ \varphi_{k-1} \circ \cdots \circ \varphi_1$$

Reverse Mode of AD

For $k = l, l-1, \cdots, 1$

do something with $v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$

- $f_k(S_k) = f_{k+1}(S_{k+1} \setminus \{v_k\}, v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}})$
- First order chain rule : $\frac{\partial f_k}{\partial v_i} = \frac{\partial f_{k+1}}{\partial v_i} + \frac{\partial v_k}{\partial v_i} \frac{\partial f_{k+1}}{\partial v_k}$

Reverse Mode of AD

For $k = l, l-1, \cdots, 1$

For all $v_i \prec v_k$:

$\frac{\partial f_k}{\partial v_i} = \frac{\partial f_{k+1}}{\partial v_i} + \frac{\partial v_k}{\partial v_i} \frac{\partial f_{k+1}}{\partial v_k}$

- $f_k(S_k) = f_{k+1}(S_{k+1} \setminus \{v_k\}, v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}})$
- First order chain rule : $\frac{\partial f_k}{\partial v_i} = \frac{\partial f_{k+1}}{\partial v_i} + \frac{\partial v_k}{\partial v_i} \frac{\partial f_{k+1}}{\partial v_k}$

Reverse Mode of AD

For $k = l, l-1, \cdots, 1$

For all $v_i \prec v_k$:

$$\frac{\partial f_k}{\partial v_i} = \frac{\partial f_{k+1}}{\partial v_i} + \frac{\partial v_k}{\partial v_i}\frac{\partial f_{k+1}}{\partial v_k}$$

- $f_k(S_k) = f_{k+1}(S_{k+1} \setminus \{v_k\}, v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}})$
- Second order chain rule :

$$\frac{\partial^2 f_k}{\partial v_i \partial v_j} = \frac{\partial^2 f_{k+1}}{\partial v \partial u} + \frac{\partial v_k}{\partial v_i}\frac{\partial^2 f_{k+1}}{\partial v_j \partial v_k} + \frac{\partial v_k}{\partial v_j}\frac{\partial^2 f_{k+1}}{\partial v_i \partial v_k} + \frac{\partial v_k}{\partial v_i}\frac{\partial v_k}{\partial v_j}\frac{\partial^2 f_{k+1}}{\partial v_k \partial v_k} + \frac{\partial^2 v_k}{\partial v_i \partial v_i}\frac{\partial f_{k+1}}{\partial v_k}$$

Reverse Mode of AD

For $k = l, l-1, \cdots, 1$

For all $v_i \prec v_k$:

$$\frac{\partial f_k}{\partial v_i} = \frac{\partial f_{k+1}}{\partial v_i} + \frac{\partial v_k}{\partial v_i}\frac{\partial f_{k+1}}{\partial v_k}$$

For all unordered pairs (v_i, v_j), $v_i \prec v_k$ or $v_j \prec v_k$:

$$\frac{\partial^2 f_k}{\partial v_i \partial v_j} = \frac{\partial^2 f_{k+1}}{\partial v \partial u} + \frac{\partial v_k}{\partial v_i}\frac{\partial^2 f_{k+1}}{\partial v_j \partial v_k} + \frac{\partial v_k}{\partial v_j}\frac{\partial^2 f_{k+1}}{\partial v_i \partial v_k} + \frac{\partial v_k}{\partial v_i}\frac{\partial v_k}{\partial v_j}\frac{\partial^2 f_{k+1}}{\partial v_k \partial v_k} + \frac{\partial^2 v_k}{\partial v_i \partial v_i}\frac{\partial f_{k+1}}{\partial v_k}$$

- $f_k(S_k) = f_{k+1}(S_{k+1} \setminus \{v_k\}, v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}})$
- Second order chain rule :

$$\frac{\partial^2 f_k}{\partial v_i \partial v_j} = \frac{\partial^2 f_{k+1}}{\partial v \partial u} + \frac{\partial v_k}{\partial v_i}\frac{\partial^2 f_{k+1}}{\partial v_j \partial v_k} + \frac{\partial v_k}{\partial v_j}\frac{\partial^2 f_{k+1}}{\partial v_i \partial v_k} + \frac{\partial v_k}{\partial v_i}\frac{\partial v_k}{\partial v_j}\frac{\partial^2 f_{k+1}}{\partial v_k \partial v_k} + \frac{\partial^2 v_k}{\partial v_i \partial v_i}\frac{\partial f_{k+1}}{\partial v_k}$$

Reverse Mode of AD

For $k = l, l-1, \cdots, 1$

For all $v_i \prec v_k$:

$$\frac{\partial f_k}{\partial v_i} = \frac{\partial f_{k+1}}{\partial v_i} + \frac{\partial v_k}{\partial v_i}\frac{\partial f_{k+1}}{\partial v_k} \rightarrow \bar{v}_i += \frac{\partial v_k}{\partial v_i}\bar{v}_k$$

For all unordered pairs (v_i, v_j), $v_i \prec v_k$ or $v_j \prec v_k$:

$$\frac{\partial^2 f_k}{\partial v_i \partial v_j} = \frac{\partial^2 f_{k+1}}{\partial v \partial u} + \frac{\partial v_k}{\partial v_i}\frac{\partial^2 f_{k+1}}{\partial v_j \partial v_k} + \frac{\partial v_k}{\partial v_j}\frac{\partial^2 f_{k+1}}{\partial v_i \partial v_k}$$
$$+ \frac{\partial v_k}{\partial v_i}\frac{\partial v_k}{\partial v_j}\frac{\partial^2 f_{k+1}}{\partial v_k \partial v_k} + \frac{\partial^2 v_k}{\partial v_i \partial v_i}\frac{\partial f_{k+1}}{\partial v_k}$$

- $f_k(S_k) = f_{k+1}(S_{k+1} \setminus \{v_k\}, v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}})$
- Adjoint variable $\bar{v}_i$:
  - Holds the value of $\frac{\partial f_k}{\partial v_i}$ after the step k
  - Incremental updates in implementation

Reverse Mode of AD

For $k = l, l-1, \cdots, 1$

For all $v_i \prec v_k$:

$$\frac{\partial f_k}{\partial v_i} = \frac{\partial f_{k+1}}{\partial v_i} + \frac{\partial v_k}{\partial v_i}\frac{\partial f_{k+1}}{\partial v_k} \rightarrow \bar{v}_i += \frac{\partial v_k}{\partial v_i}\bar{v}_k$$

For all unordered pairs (v_i, v_j), $v_i \prec v_k$ or $v_j \prec v_k$:

$$\frac{\partial^2 f_k}{\partial v_i \partial v_j} = \frac{\partial^2 f_{k+1}}{\partial v \partial u} + \frac{\partial v_k}{\partial v_i}\frac{\partial^2 f_{k+1}}{\partial v_j \partial v_k} + \frac{\partial v_k}{\partial v_j}\frac{\partial^2 f_{k+1}}{\partial v_i \partial v_k}$$
$$+ \frac{\partial v_k}{\partial v_i}\frac{\partial v_k}{\partial v_j}\frac{\partial^2 f_{k+1}}{\partial v_k \partial v_k} + \frac{\partial^2 v_k}{\partial v_i \partial v_i}\frac{\partial f_{k+1}}{\partial v_k}$$

- $f_k(S_k) = f_{k+1}(S_{k+1} \setminus \{v_k\}, v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}})$
- Adjoint variable $\bar{v}_i$:
  - Holds the value of $\frac{\partial f_k}{\partial v_i}$ after the step k
  - Incremental updates in implementation
- More implementation details for second order for exploiting sparsity and symmetry.

Reverse Mode of AD

For $k = l, l-1, \cdots, 1$

For all $v_i \prec v_k$:

$$\frac{\partial f_k}{\partial v_i} = \frac{\partial f_{k+1}}{\partial v_i} + \frac{\partial v_k}{\partial v_i}\frac{\partial f_{k+1}}{\partial v_k} \rightarrow \bar{v}_i += \frac{\partial v_k}{\partial v_i}\bar{v}_k$$

For all unordered pairs (v_i, v_j), $v_i \prec v_k$ or $v_j \prec v_k$:

$$\frac{\partial^2 f_k}{\partial v_i \partial v_j} = \frac{\partial^2 f_{k+1}}{\partial v \partial u} + \frac{\partial v_k}{\partial v_i}\frac{\partial^2 f_{k+1}}{\partial v_j \partial v_k} + \frac{\partial v_k}{\partial v_j}\frac{\partial^2 f_{k+1}}{\partial v_i \partial v_k} + \frac{\partial v_k}{\partial v_i}\frac{\partial v_k}{\partial v_j}\frac{\partial^2 f_{k+1}}{\partial v_k \partial v_k} + \frac{\partial^2 v_k}{\partial v_i \partial v_i}\frac{\partial f_{k+1}}{\partial v_k}$$

- $f_k(S_k) = f_{k+1}(S_{k+1} \setminus \{v_k\}, v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}})$
- General high order chain rule $\rightarrow$ general high order reverse mode
- Taking advantage of symmetry becomes more important

Reverse Mode of AD : Implementation

- Second order reverse mode : Initially implemented as `LivarH` in `ADOL-C`
  - `https://github.com/CSCsw/LivarH`
- `ReverseAD` : an operator overloading implementation of general high order reverse mode in C++11.
  - `https://github.com/wangmu0701/ReverseAD`
  - Available for experimentation
- Monotonic indexing for variables on the trace

$$v_i \prec v_k \implies index(v_i) < index(v_j)$$

  - Not satisfied by `ADOL-C`
  - An immature fix was provided for `LivarH`

Reverse Mode of AD : Implementation

- Second order reverse mode : Initially implemented as `LivarH` in `ADOL-C`
  - `https://github.com/CSCsw/LivarH`
- `ReverseAD` : an operator overloading implementation of general high order reverse mode in C++11.
  - `https://github.com/wangmu0701/ReverseAD`
  - Available for experimentation
- Monotonic indexing for variables on the trace

$$v_i \prec v_k \implies index(v_i) < index(v_j)$$

  - Not satisfied by `ADOL-C`
  - An immature fix was provided for `LivarH`

Reverse Mode of AD : Implementation

- Second order reverse mode : Initially implemented as `LivarH` in `ADOL-C`
  - `https://github.com/CSCsw/LivarH`
- `ReverseAD` : an operator overloading implementation of general high order reverse mode in C++11.
  - `https://github.com/wangmu0701/ReverseAD`
  - Available for experimentation
- Monotonic indexing for variables on the trace

$$v_i \prec v_k \implies index(v_i) < index(v_j)$$

  - Not satisfied by `ADOL-C`
  - An immature fix was provided for `LivarH`

Reverse Mode of AD : Performance

- The FeasNewt Benchmark (T. S. Munson and P. D. Hovland, 2005)
- A mesh optimization problem with sparse Hessian matrix.
- Compared with compression-and-recovery approach implemented in `ADOL-C + ColPack`

n :		2,598	12,597	39,379
$\#nnz$ in H :		46,488	253,029	828,129
Direct	#colors :	54	62	65
	runtime(S) :	3.77	39.34	137.07
Indirect	#colors :	31	30	31
	runtime(S) :	3.56	31.07	119.04
ReverseAD	runtime(S) :	0.51	3.37	12.40

From Analytical to Combinatorial

- The second (high) order reverse mode is derived from a purely analytical point of view.
  - Same as the original derivation of Edge Pushing.
- There are combinatorial models for AD algorithms based on the concept of Computational Graph G of the objective function.
  - Edge Elimination
  - Vertex Elimination
  - Face Elimination
- Closely related to the classical linear algebra problem of sparse Gaussian elimination.

From Analytical to Combinatorial

- The second (high) order reverse mode is derived from a purely analytical point of view.
  - Same as the original derivation of Edge Pushing.
- There are combinatorial models for AD algorithms based on the concept of Computational Graph G of the objective function.
  - Edge Elimination
  - Vertex Elimination
  - Face Elimination
- Closely related to the classical linear algebra problem of sparse Gaussian elimination.

From Analytical to Combinatorial

- The second (high) order reverse mode is derived from a purely analytical point of view.
  - Same as the original derivation of Edge Pushing.
- There are combinatorial models for AD algorithms based on the concept of Computational Graph G of the objective function.
  - Edge Elimination
  - Vertex Elimination
  - Face Elimination
- Closely related to the classical linear algebra problem of sparse Gaussian elimination.

Computational Graph

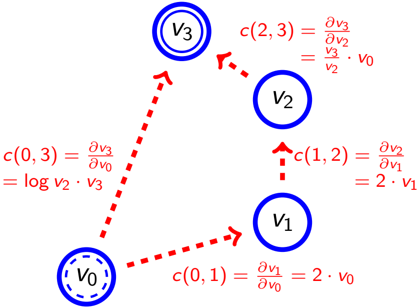
- *Computational graph* : $G = (V, E)$
  - Variables are vertices : $V = \{v_i | 1 - n \le i \le l\}$
  - Precedence relations are directed edges :
    $$E = \{v_i \rightarrow v_k | v_i \prec v_k, 1 - n \le i < k \le l\}$$
  - Edge weights : $c(i, k) \doteq w(v_i, v_k) = \frac{\partial v_k}{\partial v_i}$

- $v_1 = \varphi_1(v_0) = v_0 * v_0$
- $v_2 = \varphi_2(v_1) = pow(v_1, 2.0)$
- $v_3 = \varphi_3(v_2, v_0) = pow(v_2, v_0)$

Computational Graph

- *Computational graph* : $G = (V, E)$
  - Variables are vertices : $V = \{v_i | 1-n \le i \le l\}$
  - Precedence relations are directed edges :
  $$E = \{v_i \to v_k | v_i \prec v_k, 1-n \le i < k \le l\}$$
  - Edge weights : $c(i,k) \doteq w(v_i, v_k) = \frac{\partial v_k}{\partial v_i}$

- $v_1 = \varphi_1(v_0) = v_0 * v_0$
- $v_2 = \varphi_2(v_1) = pow(v_1, 2.0)$
- $v_3 = \varphi_3(v_2, v_0) = pow(v_2, v_0)$



Vertex Elimination

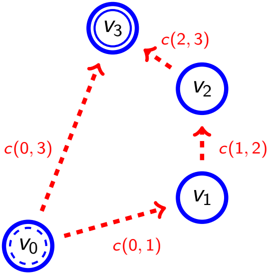
Repeat

- Pick intermediate node v_j
- For all (i, k), s.t, $i \prec j \prec k$ do

  $$c(i,k) += c(i,j) * c(j,k)$$

- Remove v_j from V

Until V has no intermediate vertices



- Proposed by Griewank and Reese, and studied extensively by Naumann and students

Vertex Elimination

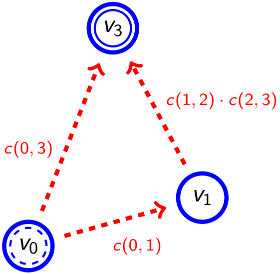
Repeat

- Pick intermediate node v_j
- For all (i, k), s.t, $i \prec j \prec k$ do

$$c(i,k) += c(i,j) * c(j,k)$$

- Remove v_j from V

Until V has no intermediate vertices



- Proposed by Griewank and Reese, and studied extensively by Naumann and students

Vertex Elimination

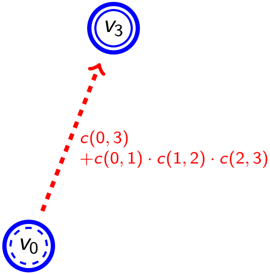
Repeat

- Pick intermediate node v_j
- For all (i, k), s.t, $i \prec j \prec k$ do

$$c(i,k) += c(i,j) * c(j,k)$$

- Remove v_j from V

Until V has no intermediate vertices



- Proposed by Griewank and Reese, and studied extensively by Naumann and students

Vertex Elimination

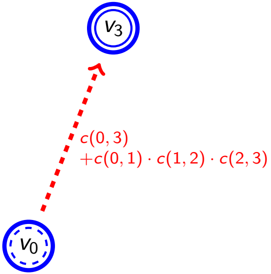
Repeat

- Pick intermediate node v_j
- For all (i, k), s.t, $i \prec j \prec k$ do

  $$c(i,k) += c(i,j) * c(j,k)$$

- Remove v_j from V

Until V has no intermediate vertices



- Proposed by Griewank and Reese, and studied extensively by Naumann and students
  - Any elimination order will give the same final results.
  - The time complexity (number of edge weights computed) varies with the ordering. Minimizing the space complexity also is likely to be intractable.
  - NP-hard to determine the optimal ordering.

Vertex Elimination for Hessian

- The vertex elimination algorithm applies on G, gives $\nabla \cdot f$.
- To evaluate the Hessian of f we need the computational graph of the gradient G_g, i.e, the computational graph of evaluating $\nabla \cdot f$.
- G_g can be constructed from first order non-incremental reverse mode

---

Function evaluation :

for $k = 1, 2, \cdots, l$

$$v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$$

First order (nonincremental) reverse mode :

Initialize :

$$\bar{v}_l = 1.0, \bar{v}_{l-1} = \cdots = 0$$

for $i = l-1, \cdots, 1, 0, \cdots, 1-n$

$$\bar{v}_i = \sum_{v_i \prec v_k} \frac{\partial v_k}{\partial v_i} \bar{v}_k$$

$$\bar{v}_i = \bar{\varphi}_i(\cup_{v_i \prec v_k} \{v_j : v_j \prec v_k\} \cup \{v_k\})$$

Vertex Elimination for Hessian

- The vertex elimination algorithm applies on G, gives $\nabla \cdot f$.
- To evaluate the Hessian of f we need the computational graph of the gradient G_g, i.e, the computational graph of evaluating $\nabla \cdot f$.
- G_g can be constructed from first order non-incremental reverse mode

---

Function evaluation :

for $k = 1, 2, \cdots, l$

$$v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$$

First order (nonincremental) reverse mode :

Initialize :

$$\bar{v}_l = 1.0, \bar{v}_{l-1} = \cdots = 0$$

for $i = l-1, \cdots, 1, 0, \cdots, 1-n$

$$\bar{v}_i = \sum_{v_i \prec v_k} \frac{\partial v_k}{\partial v_i} \bar{v}_k$$

$$\bar{v}_i = \bar{\varphi}_i(\cup_{v_i \prec v_k} \{v_j : v_j \prec v_k\} \cup \{v_k\})$$

Vertex Elimination for Hessian

- The vertex elimination algorithm applies on G, gives $\nabla \cdot f$.
- To evaluate the Hessian of f we need the computational graph of the gradient G_g, i.e, the computational graph of evaluating $\nabla \cdot f$.
- G_g can be constructed from first order non-incremental reverse mode

---

Function evaluation :

for $k = 1, 2, \cdots, l$

$v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$

First order (nonincremental) reverse mode :

Initialize :

$\bar{v}_l = 1.0, \bar{v}_{l-1} = \cdots = 0$

for $i = l-1, \cdots, 1, 0, \cdots, 1-n$

$$\bar{v}_i = \sum_{v_i \prec v_k} \frac{\partial v_k}{\partial v_i} \bar{v}_k$$

$$\bar{v}_i = \bar{\varphi}_i(\cup_{v_i \prec v_k} \{v_j : v_j \prec v_k\} \cup \{v_k\})$$

Vertex Elimination for Hessian

- The vertex elimination algorithm applies on G, gives $\nabla \cdot f$.
- To evaluate the Hessian of f we need the computational graph of the gradient G_g, i.e, the computational graph of evaluating $\nabla \cdot f$.
- G_g can be constructed from first order non-incremental reverse mode

---

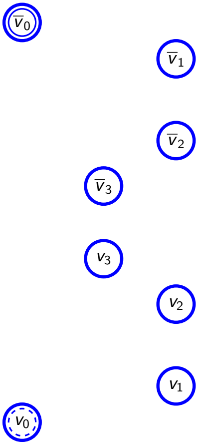
Function evaluation :

$$\begin{aligned}
&\text{for } k = 1, 2, \cdots, l \\
&\quad v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}
\end{aligned}$$

First order (nonincremental) reverse mode :

$$\begin{aligned}
&\text{Initialize :} \\
&\quad \bar{v}_l = 1.0, \bar{v}_{l-1} = \cdots = 0 \\
&\text{for } i = l-1, \cdots, 1, 0, \cdots, 1-n \\
&\quad \bar{v}_i = \sum_{v_i \prec v_k} \frac{\partial v_k}{\partial v_i} \bar{v}_k \\
&\quad \bar{v}_i = \bar{\varphi}_i(\cup_{v_i \prec v_k} \{v_j : v_j \prec v_k\} \cup \{v_k\})
\end{aligned}$$

Computational Graph of the Gradient



Function evaluation :

- for $k = 1, 2, \cdots, l$

  $$v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$$

First order (nonincremental) reverse mode :

- Initialize :

  $$\bar{v}_l = 1.0, \bar{v}_{l-1} = \cdots = 0$$

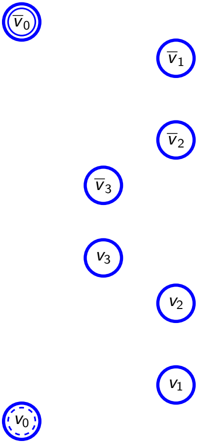
- for $i = l-1, \cdots, 1, 0, \cdots, 1-n$

  $$\bar{v}_i = \sum_{v_i \prec v_k} \frac{\partial v_k}{\partial v_i} \bar{v}_k$$

  $$\bar{v}_i = \bar{\varphi}_i(\cup_{v_i \prec v_k} \{v_j : v_j \prec v_k\} \cup \{v_k\})$$

$$V_g = V \cup \bar{V}, \; E_g = E_G \cup E_{\bar{G}} \cup E_C$$

Computational Graph of the Gradient



Function evaluation :

- for $k = 1, 2, \cdots, l$

  $v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$

First order (nonincremental) reverse mode :

- Initialize :

  $\bar{v}_l = 1.0, \bar{v}_{l-1} = \cdots = 0$

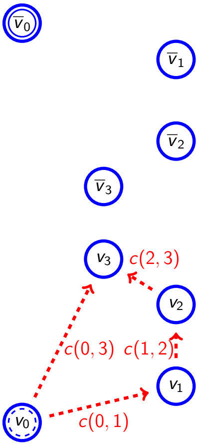
- for $i = l - 1, \cdots, 1, 0, \cdots, 1 - n$

  $\bar{v}_i = \sum_{v_i \prec v_k} \frac{\partial v_k}{\partial v_i} \bar{v}_k$

  $\bar{v}_i = \bar{\varphi}_i(\cup_{v_i \prec v_k} \{v_j : v_j \prec v_k\} \cup \{v_k\})$

$V_g = V \cup \bar{V},\ E_g = E_G \cup E_{\bar{G}} \cup E_C$

Computational Graph of the Gradient



Function evaluation :

- for $k = 1, 2, \cdots, l$
  $$v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$$

First order (nonincremental) reverse mode :

- Initialize :
  $$\bar{v}_l = 1.0, \bar{v}_{l-1} = \cdots = 0$$
- for $i = l-1, \cdots, 1, 0, \cdots, 1-n$
  $$\bar{v}_i = \sum_{v_i \prec v_k} \frac{\partial v_k}{\partial v_i} \bar{v}_k$$
  $$\bar{v}_i = \bar{\varphi}_i(\cup_{v_i \prec v_k} \{v_j : v_j \prec v_k\} \cup \{v_k\})$$

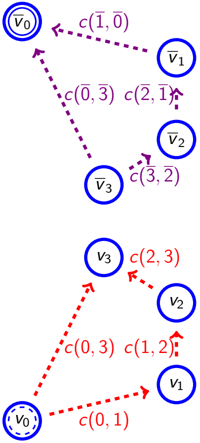
$$V_g = V \cup \bar{V},\ E_g = E_G \cup E_{\bar{G}} \cup E_C$$

$$E_G : (v_i, v_k) \in E_g \iff v_i \prec v_k$$

$$v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$$

$$c(i,k) = \frac{\partial v_k}{\partial v_i}$$

Computational Graph of the Gradient



Function evaluation :

- for $k = 1, 2, \cdots, l$
  $$v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$$

First order (nonincremental) reverse mode :

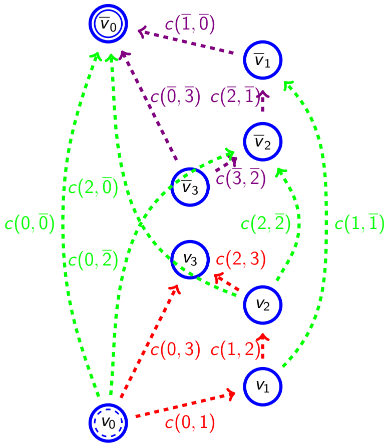
- Initialize :
  $$\bar{v}_l = 1.0, \bar{v}_{l-1} = \cdots = 0$$
- for $i = l-1, \cdots, 1, 0, \cdots, 1-n$
  $$\bar{v}_i = \sum_{v_i \prec v_k} \frac{\partial v_k}{\partial v_i} \bar{v}_k$$
  $$\bar{v}_i = \bar{\varphi}_i(\cup_{v_i \prec v_k} \{v_j : v_j \prec v_k\} \cup \{v_k\})$$

$$V_g = V \cup \bar{V},\ E_g = E_G \cup E_{\bar{G}} \cup E_C$$

$$E_{\bar{G}} : (\bar{v}_k, \bar{v}_i) \in E_g \iff \bar{v}_k \prec \bar{v}_i \iff v_i \prec v_k$$

$$\bar{v}_i = \sum_{v_i \prec v_k} \frac{\partial v_k}{\partial v_i} \bar{v}_k$$
$$= \bar{\varphi}_i(\cup_{v_i \prec v_k} \{v_j : v_j \prec v_k\} \cup \{v_k\})$$
$$c(\bar{k}, \bar{i}) = \frac{\partial v_j}{\partial v_i}$$

Computational Graph of the Gradient



Function evaluation :

- for $k = 1, 2, \cdots, l$

  $$v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$$

First order (nonincremental) reverse mode :

- Initialize :

  $$\bar{v}_l = 1.0, \bar{v}_{l-1} = \cdots = 0$$

- for $i = l-1, \cdots, 1, 0, \cdots, 1-n$

  $$\bar{v}_i = \sum_{v_i \prec v_k} \frac{\partial v_k}{\partial v_i} \bar{v}_k$$

  $$\bar{v}_i = \bar{\varphi}_i(\cup_{v_i \prec v_k}\{v_j : v_j \prec v_k\} \cup \{v_k\})$$

$$V_g = V \cup \bar{V},\ E_g = E_G \cup E_{\bar{G}} \cup E_C$$

$$E_C : (v_i, \bar{v}_j) \in E_g \iff \exists v_k, s.t, v_i, v_j \prec v_k$$

$$\bar{v}_i = \sum_{v_i \prec v_k} \frac{\partial v_k}{\partial v_i} \bar{v}_k$$

$$= \bar{\varphi}_i(\cup_{v_i \prec v_k}\{v_j : v_j \prec v_k\} \cup \{v_k\})$$

$$c(i, \bar{j}) = \sum_{v_i, v_j \prec v_k} \frac{\partial^2 v_k}{\partial v_i \partial v_j} \bar{v}_k$$

Equivalence

- Vertex elimination on the gradient graph G_g gives the Hessian (combinatorial approach).
- Second order reverse mode gives the Hessian (analytical approach).

Second order reverse mode:

- Initialize :
  $\bar{v}_l = 1.0, \bar{v}_{l-1} = \cdots = 0$
- for $k = l, l-1, \cdots, 1$
  for each unordered pair (v_i, v_j)
  $$h_k(v_i, v_j) = h_{k+1}(v_i, v_j) + \frac{\partial v_k}{\partial v_i} h_{k+1}(v_j, v_k) + \frac{\partial v_k}{\partial v_j} h_{k+1}(v_i, v_k) + \frac{\partial v_k}{\partial v_i}\frac{\partial v_k}{\partial v_j} h_{k+1}(v_k, v_k) + \frac{\partial^2 v_k}{\partial v_i \partial v_j}\bar{v}_k$$

Vertex Elimination on G_g

- Pick intermediate node v_j
- For all (i, k), s.t. $i \prec j \prec k$ do
  $c(i, k) += c(i, j) * c(j, k)$
- Remove v_j from V
- Repeat until V has no intermediate vertices

Theorem

If vertex elimination is performed on G_g in a symmetric reverse topological ordering, i.e, $(v_k, \bar{v}_k)$ are eliminated in pairs, in the order $k = l, l-1, \cdots, 1$, then the two algorithms correspond step-by-step.

Equivalence

- Vertex elimination on the gradient graph G_g gives the Hessian (combinatorial approach).
- Second order reverse mode gives the Hessian (analytical approach).

Second order reverse mode:

- Initialize : $\bar{v}_l = 1.0, \bar{v}_{l-1} = \cdots = 0$
- for $k = l, l-1, \cdots, 1$
  for each unordered pair (v_i, v_j)
  $$h_k(v_i, v_j) = h_{k+1}(v_i, v_j) + \frac{\partial v_k}{\partial v_i} h_{k+1}(v_j, v_k) + \frac{\partial v_k}{\partial v_j} h_{k+1}(v_i, v_k) + \frac{\partial v_k}{\partial v_i}\frac{\partial v_k}{\partial v_j} h_{k+1}(v_k, v_k) + \frac{\partial^2 v_k}{\partial v_i \partial v_j}\bar{v}_k$$

Vertex Elimination on G_g

- Pick intermediate node v_j
- For all (i, k), s.t. $i \prec j \prec k$ do
  $c(i, k) += c(i, j) * c(j, k)$
- Remove v_j from V
- Repeat until V has no intermediate vertices

Theorem

If vertex elimination is performed on G_g in a symmetric reverse topological ordering, i.e, $(v_k, \bar{v}_k)$ are eliminated in pairs, in the order $k = l, l-1, \cdots, 1$, then the two algorithms correspond step-by-step.

Equivalence

- Vertex elimination on the gradient graph G_g gives the Hessian (combinatorial approach).
- Second order reverse mode gives the Hessian (analytical approach).

Second order reverse mode:

- Initialize :
  $\bar{v}_l = 1.0, \bar{v}_{l-1} = \cdots = 0$
- for $k = l, l-1, \cdots, 1$
  for each unordered pair (v_i, v_j)
  $$h_k(v_i, v_j) = h_{k+1}(v_i, v_j) + \frac{\partial v_k}{\partial v_i} h_{k+1}(v_j, v_k) + \frac{\partial v_k}{\partial v_j} h_{k+1}(v_i, v_k) + \frac{\partial v_k}{\partial v_i}\frac{\partial v_k}{\partial v_j} h_{k+1}(v_k, v_k) + \frac{\partial^2 v_k}{\partial v_i \partial v_j}\bar{v}_k$$

Vertex Elimination on G_g

- Pick intermediate node v_j
- For all (i, k), s.t, $i \prec j \prec k$ do
  $c(i, k) += c(i, j) * c(j, k)$
- Remove v_j from V
- Repeat until V has no intermediate vertices

Theorem

If vertex elimination is performed on G_g in a symmetric reverse topological ordering, i.e, $(v_k, \bar{v}_k)$ are eliminated in pairs, in the order $k = l, l-1, \cdots, 1$, then the two algorithms correspond step-by-step.

Equivalence

- Vertex elimination on the gradient graph G_g gives the Hessian (combinatorial approach).
- Second order reverse mode gives the Hessian (analytical approach).

Second order reverse mode:

- Initialize :
  $\bar{v}_l = 1.0, \bar{v}_{l-1} = \cdots = 0$
- for $k = l, l-1, \cdots, 1$
  for each unordered pair (v_i, v_j)

$$h_k(v_i, v_j) = h_{k+1}(v_i, v_j) + \frac{\partial v_k}{\partial v_i} h_{k+1}(v_j, v_k) + \frac{\partial v_k}{\partial v_j} h_{k+1}(v_i, v_k) + \frac{\partial v_k}{\partial v_i}\frac{\partial v_k}{\partial v_j} h_{k+1}(v_k, v_k) + \frac{\partial^2 v_k}{\partial v_i \partial v_j}\bar{v}_k$$

Vertex Elimination on G_g

- Pick intermediate node v_j
- For all (i, k), s.t, $i \prec j \prec k$ do
  $c(i, k) += c(i, j) * c(j, k)$
- Remove v_j from V
- Repeat until V has no intermediate vertices

Theorem

If vertex elimination is performed on G_g in a symmetric reverse topological ordering, i.e, $(v_k, \bar{v}_k)$ are eliminated in pairs, in the order $k = l, l-1, \cdots, 1$, then the two algorithms correspond step-by-step.

Theorem

- The two algorithms perform the same computations, and thus maintain the same intermediate results after each step.
  - With two minor tweaks of vertex elimination on G_g
- Tweak one : parallel edges in E_C
  - Break the edge $c(i,\bar{j}) = \sum_{v_i, v_j \prec v_k} \frac{\partial^2 v_k}{\partial v_i \partial v_j} \bar{v}_k$
  - Into parallel edges $c^k(i,\bar{j}) = \frac{\partial^2 v_k}{\partial v_i \partial v_j} \bar{v}_k$
- Tweak two : new set of edges E_H :
  - Rule 1 : all added edges are added into E_H
  - Rule 2 : After eliminating $(v_k, \bar{v}_k)$, move all $c^k(i,\bar{j})$ from E_C to E_H
- Claim : E_H corresponds to the nonzeros in the Hessian of $f_k(S_k)$ after each step.

Theorem

- The two algorithms perform the same computations, and thus maintain the same intermediate results after each step.
  - With two minor tweaks of vertex elimination on G_g
- Tweak one : parallel edges in E_C
  - Break the edge $c(i, \bar{j}) = \sum_{v_i, v_j \prec v_k} \frac{\partial^2 v_k}{\partial v_i \partial v_j} \bar{v}_k$
  - Into parallel edges $c^k(i, \bar{j}) = \frac{\partial^2 v_k}{\partial v_i \partial v_j} \bar{v}_k$
- Tweak two : new set of edges E_H :
  - Rule 1 : all added edges are added into E_H
  - Rule 2 : After eliminating $(v_k, \bar{v}_k)$, move all $c^k(i, \bar{j})$ from E_C to E_H
- Claim : E_H corresponds to the nonzeros in the Hessian of $f_k(S_k)$ after each step.

Theorem

- The two algorithms perform the same computations, and thus maintain the same intermediate results after each step.
  - With two minor tweaks of vertex elimination on G_g
- Tweak one : parallel edges in E_C
  - Break the edge $c(i,\bar{j}) = \sum_{v_i, v_j \prec v_k} \frac{\partial^2 v_k}{\partial v_i \partial v_j} \bar{v}_k$
  - Into parallel edges $c^k(i,\bar{j}) = \frac{\partial^2 v_k}{\partial v_i \partial v_j} \bar{v}_k$
- Tweak two : new set of edges E_H :
  - Rule 1 : all added edges are added into E_H
  - Rule 2 : After eliminating $(v_k, \bar{v}_k)$, move all $c^k(i,\bar{j})$ from E_C to E_H
- Claim : E_H corresponds to the nonzeros in the Hessian of $f_k(S_k)$ after each step.

Theorem

- The two algorithms perform the same computations, and thus maintain the same intermediate results after each step.
  - With two minor tweaks of vertex elimination on G_g
- Tweak one : parallel edges in E_C
  - Break the edge $c(i,\bar{j}) = \sum_{v_i, v_j \prec v_k} \frac{\partial^2 v_k}{\partial v_i \partial v_j} \bar{v}_k$
  - Into parallel edges $c^k(i,\bar{j}) = \frac{\partial^2 v_k}{\partial v_i \partial v_j} \bar{v}_k$
- Tweak two : new set of edges E_H :
  - Rule 1 : all added edges are added into E_H
  - Rule 2 : After eliminating $(v_k, \bar{v}_k)$, move all $c^k(i,\bar{j})$ from E_C to E_H
- Claim : E_H corresponds to the nonzeros in the Hessian of $f_k(S_k)$ after each step.

Discussion

- Second order reverse mode is equivalent to a special form of vertex elimination on the computational graph of the gradient G_g.
- May not be the optimal form of vertex elimination due to the structure of G_g. But, in practice it can be implemented with efficient storage and memory access.
  - Second order reverse mode does not require the graph G_g to be formed.
  - Can be implemented with a single reverse sweep.
  - Can incorporate checkpointing to overcome memory/disk limits
- Possibilities of optimizing second order reverse mode by exploiting structural properties
  - Out-of-order processing of $v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$
  - Benefit must outweigh the optimization overhead

Discussion

- Second order reverse mode is equivalent to a special form of vertex elimination on the computational graph of the gradient G_g.
- May not be the optimal form of vertex elimination due to the structure of G_g. But, in practice it can be implemented with efficient storage and memory access.
  - Second order reverse mode does not require the graph G_g to be formed.
  - Can be implemented with a single reverse sweep.
  - Can incorporate checkpointing to overcome memory/disk limits
- Possibilities of optimizing second order reverse mode by exploiting structural properties
  - Out-of-order processing of $v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$
  - Benefit must outweigh the optimization overhead

Discussion

- Second order reverse mode is equivalent to a special form of vertex elimination on the computational graph of the gradient G_g.
- May not be the optimal form of vertex elimination due to the structure of G_g. But, in practice it can be implemented with efficient storage and memory access.
  - Second order reverse mode does not require the graph G_g to be formed.
  - Can be implemented with a single reverse sweep.
  - Can incorporate checkpointing to overcome memory/disk limits
- Possibilities of optimizing second order reverse mode by exploiting structural properties
  - Out-of-order processing of $v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$
  - Benefit must outweigh the optimization overhead

Discussion

- Second order reverse mode is equivalent to a special form of vertex elimination on the computational graph of the gradient G_g.
- May not be the optimal form of vertex elimination due to the structure of G_g. But, in practice it can be implemented with efficient storage and memory access.
  - Second order reverse mode does not require the graph G_g to be formed.
  - Can be implemented with a single reverse sweep.
  - Can incorporate checkpointing to overcome memory/disk limits
- Possibilities of optimizing second order reverse mode by exploiting structural properties
  - Out-of-order processing of $v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$
  - Benefit must outweigh the optimization overhead

Discussion

- Second order reverse mode is equivalent to a special form of vertex elimination on the computational graph of the gradient G_g.
- May not be the optimal form of vertex elimination due to the structure of G_g. But, in practice it can be implemented with efficient storage and memory access.
  - Second order reverse mode does not require the graph G_g to be formed.
  - Can be implemented with a single reverse sweep.
  - Can incorporate checkpointing to overcome memory/disk limits
- Possibilities of optimizing second order reverse mode by exploiting structural properties
  - Out-of-order processing of $v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$
  - Benefit must outweigh the optimization overhead

Discussion

- Second order reverse mode is equivalent to a special form of vertex elimination on the computational graph of the gradient G_g.
- May not be the optimal form of vertex elimination due to the structure of G_g. But, in practice it can be implemented with efficient storage and memory access.
  - Second order reverse mode does not require the graph G_g to be formed.
  - Can be implemented with a single reverse sweep.
  - Can incorporate checkpointing to overcome memory/disk limits
- Possibilities of optimizing second order reverse mode by exploiting structural properties
  - Out-of-order processing of $v_k = \varphi_k(v_i)_{\{v_i : v_i \prec v_k\}}$
  - Benefit must outweigh the optimization overhead

Future Work : Broad Picture

- This work reveals the correspondence between analytical and combinatorial points of view of AD algorithms.
  - First order forward/reverse mode corresponds to edge elimination on G with specific elimination ordering.
  - Second order reverse mode corresponds to vertex elimination on G_g with reverse symmetric elimination ordering.
  - Is there a generalization to high orders?
- The analytical form of the high order reverse mode is the implementation of high order chain rule.
- What is the generalization of the combinatorial form of high order reverse mode?
- What is the computational graph of the Hessian G_H?
- What is the elimination technique that we should perform on G_H?

Future Work : Broad Picture

- This work reveals the correspondence between analytical and combinatorial points of view of AD algorithms.
  - First order forward/reverse mode corresponds to edge elimination on G with specific elimination ordering.
  - Second order reverse mode corresponds to vertex elimination on G_g with reverse symmetric elimination ordering.
  - Is there a generalization to high orders?
- The analytical form of the high order reverse mode is the implementation of high order chain rule.
- What is the generalization of the combinatorial form of high order reverse mode?
- What is the computational graph of the Hessian G_H?
- What is the elimination technique that we should perform on G_H?

Future Work : Broad Picture

- This work reveals the correspondence between analytical and combinatorial points of view of AD algorithms.
  - First order forward/reverse mode corresponds to edge elimination on G with specific elimination ordering.
  - Second order reverse mode corresponds to vertex elimination on G_g with reverse symmetric elimination ordering.
  - Is there a generalization to high orders?
- The analytical form of the high order reverse mode is the implementation of high order chain rule.
- What is the generalization of the combinatorial form of high order reverse mode?
- What is the computational graph of the Hessian G_H?
- What is the elimination technique that we should perform on G_H?

Future Work : Broad Picture

- This work reveals the correspondence between analytical and combinatorial points of view of AD algorithms.
  - First order forward/reverse mode corresponds to edge elimination on G with specific elimination ordering.
  - Second order reverse mode corresponds to vertex elimination on G_g with reverse symmetric elimination ordering.
  - Is there a generalization to high orders?
- The analytical form of the high order reverse mode is the implementation of high order chain rule.
- What is the generalization of the combinatorial form of high order reverse mode?
- What is the computational graph of the Hessian G_H?
- What is the elimination technique that we should perform on G_H?

Future Work : Broad Picture

- This work reveals the correspondence between analytical and combinatorial points of view of AD algorithms.
  - First order forward/reverse mode corresponds to edge elimination on G with specific elimination ordering.
  - Second order reverse mode corresponds to vertex elimination on G_g with reverse symmetric elimination ordering.
  - Is there a generalization to high orders?
- The analytical form of the high order reverse mode is the implementation of high order chain rule.
- What is the generalization of the combinatorial form of high order reverse mode?
- What is the computational graph of the Hessian G_H?
- What is the elimination technique that we should perform on G_H?

Future Work : Broad Picture

- This work reveals the correspondence between analytical and combinatorial points of view of AD algorithms.
  - First order forward/reverse mode corresponds to edge elimination on G with specific elimination ordering.
  - Second order reverse mode corresponds to vertex elimination on G_g with reverse symmetric elimination ordering.
  - Is there a generalization to high orders?
- The analytical form of the high order reverse mode is the implementation of high order chain rule.
- What is the generalization of the combinatorial form of high order reverse mode?
- What is the computational graph of the Hessian G_H?
- What is the elimination technique that we should perform on G_H?

Future Work : Broad Picture

- This work reveals the correspondence between analytical and combinatorial points of view of AD algorithms.
  - First order forward/reverse mode corresponds to edge elimination on G with specific elimination ordering.
  - Second order reverse mode corresponds to vertex elimination on G_g with reverse symmetric elimination ordering.
  - Is there a generalization to high orders?
- The analytical form of the high order reverse mode is the implementation of high order chain rule.
- What is the generalization of the combinatorial form of high order reverse mode?
- What is the computational graph of the Hessian G_H?
- What is the elimination technique that we should perform on G_H?

References

- Griewank, Andreas, and Andrea Walther. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. SIAM, 2008.
- Griewank, Andreas, and Shawn Reese. On the calculation of Jacobian matrices by the Markowitz rule. In Andreas Griewank and George F. Corliss, editors, Automatic Differentiation of Algorithms: Theory, Implementation, and Application, pages 126-135. SIAM, Philadelphia, PA, 1991.
- Naumann, Uwe. Optimal Jacobian Accumulation is NP-complete. Mathematical Programming, 112(2):427-441, 2008.

- Gower, Robert Mansel, and Margarida P. Mello. Hessian matrices via automatic differentiation. Universidade Estadual de Campinas, Instituto de Matemtica, Estatstica e Computao Cientfica, 2010.
- Wang, Mu, Assefaw Gebremedhin, and Alex Pothen. Capitalizing on live variables: new algorithms for efficient Hessian computation via automatic differentiation. Mathematical Programming Computation (2016): 1-41.
- Wang, Mu, Alex Pothen and Paul Hovland. Edge Pushing is Equivalent to Vertex Elimination for Computing Hessians. SIAM CSC16.

- Wang, Mu and Alex Pothen. Evaluating High Order Derivative Tensors in Reverse Mode of Automatic Differentiation. AD2016
- Wang, Mu, and Alex Pothen. High Order Reverse Mode of AD : Theory and Implementation. In preparation.

Backup Slides

placeholder

Vertex Elimination as Gaussian Elimination

- We can build a matrix as $\mathrm{C} = [c_{ij}]_{1-n \leq i,j \leq l}$.
  - $c_{ij} = \frac{\partial v_i}{\partial v_j}$ as the edge weight in G,when $v_j \prec v_i$
  - $c_{ii} = -1$, diagonal elements
  - Other elements are zero

$$\begin{array}{cc} & \begin{array}{ccc} n & l-m & m \end{array} \\ \mathrm{C} = \begin{array}{c} n \\ l-m \\ m \end{array} & \begin{bmatrix} -\mathrm{I} & 0 & 0 \\ \mathrm{B} & \mathrm{L}-\mathrm{I} & 0 \\ \mathrm{R} & \mathrm{T} & -\mathrm{I} \end{bmatrix} \end{array}$$

- C is a lower triangular matrix
- The Jacobian $\nabla \cdot f = \mathrm{R} + \mathrm{T} \cdot (\mathrm{L} - \mathrm{I})^{-1} \cdot \mathrm{B}$ is the Schur complement
- Can use a Gaussian elimination procedure to compute it.

Adjacency Matrix for G_g

$$\mathrm{H} = \begin{array}{cc} & \begin{array}{cccccc} n & l-m & m & m & l-m & n \end{array} \\ \begin{array}{c} n \\ l-m \\ m \\ m \\ l-m \\ n \end{array} & \begin{bmatrix} -\mathrm{I} & 0 & 0 & & & \\ \mathrm{B} & \mathrm{L}-\mathrm{I} & 0 & & & \\ \mathrm{R} & \mathrm{T} & -\mathrm{I} & & & \\ 0 & 0 & 0 & -I & 0 & 0 \\ \mathrm{Z} & \mathrm{Y} & 0 & \mathrm{T}' & \mathrm{L}'-\mathrm{I} & 0 \\ \mathrm{X} & \mathrm{Z}' & 0 & \mathrm{R}' & \mathrm{B}' & -\mathrm{I} \end{bmatrix} \end{array}$$

- C' is the transpose of C along the antidiagonal.
- The Hessian is the Schur complement of X with the rest of the matrix