

Randomized Heuristics for Exploiting Jacobian Scarcity

Andrew Lyons^{1,2} Ilya Safro²

¹University of Chicago

²Argonne National Laboratory

Dagstuhl Seminar on Combinatorial Scientific Computing
February 3, 2009

Outline

Introduction and Motivation

- Linearization

- Vector Propagation

- Preaccumulation

Jacobian Scarcity

- Structural Properties of Jacobians

- Randomized Algorithms

- Results

- Rerouting and Normalization

Future Work

- Connections to Optimal Jacobian Accumulation

- Leveraging Face Elimination

Context

Given program for $\mathbf{y} = F(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$

Context

Given program for $\mathbf{y} = F(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$

Want a program $F^+(\mathbf{x})$ that computes $F(\mathbf{x})$ *plus* a collection of p Jacobian-vector products

$$F'(\mathbf{x})\dot{\mathbf{x}}^1, F'(\mathbf{x})\dot{\mathbf{x}}^2, \dots, F'(\mathbf{x})\dot{\mathbf{x}}^p$$

Context

Given program for $\mathbf{y} = F(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$

Want a program $F^+(\mathbf{x})$ that computes $F(\mathbf{x})$ *plus* a collection of p Jacobian-vector products

$$F'(\mathbf{x})\dot{\mathbf{x}}^1, F'(\mathbf{x})\dot{\mathbf{x}}^2, \dots, F'(\mathbf{x})\dot{\mathbf{x}}^p$$

or a collection of p Jacobian-transpose-vector products

$$F'(\mathbf{x})^T \bar{\mathbf{y}}^1, F'(\mathbf{x})^T \bar{\mathbf{y}}^2, \dots, F'(\mathbf{x})^T \bar{\mathbf{y}}^p$$

Context

Given program for $\mathbf{y} = F(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$

Want a program $F^+(\mathbf{x})$ that computes $F(\mathbf{x})$ *plus* a collection of p Jacobian-vector products

$$F'(\mathbf{x})\dot{\mathbf{x}}^1, F'(\mathbf{x})\dot{\mathbf{x}}^2, \dots, F'(\mathbf{x})\dot{\mathbf{x}}^p$$

or a collection of p Jacobian-transpose-vector products

$$F'(\mathbf{x})^T \bar{\mathbf{y}}^1, F'(\mathbf{x})^T \bar{\mathbf{y}}^2, \dots, F'(\mathbf{x})^T \bar{\mathbf{y}}^p$$

Assume $p \gg \max\{n, m\}$ (we want a lot of them)

As needed in Newton Krylov methods, etc.

Evaluation Procedures

$$y_1 = x_1 * x_1 * x_2, \quad y_2 = \sin(x_1 * x_1 * x_2)$$

$$v_{-1} = x_1$$

$$v_0 = x_2$$

$$v_1 = v_{-1} * v_0$$

$$v_2 = v_{-1} * v_1$$

$$v_3 = v_2 + 1$$

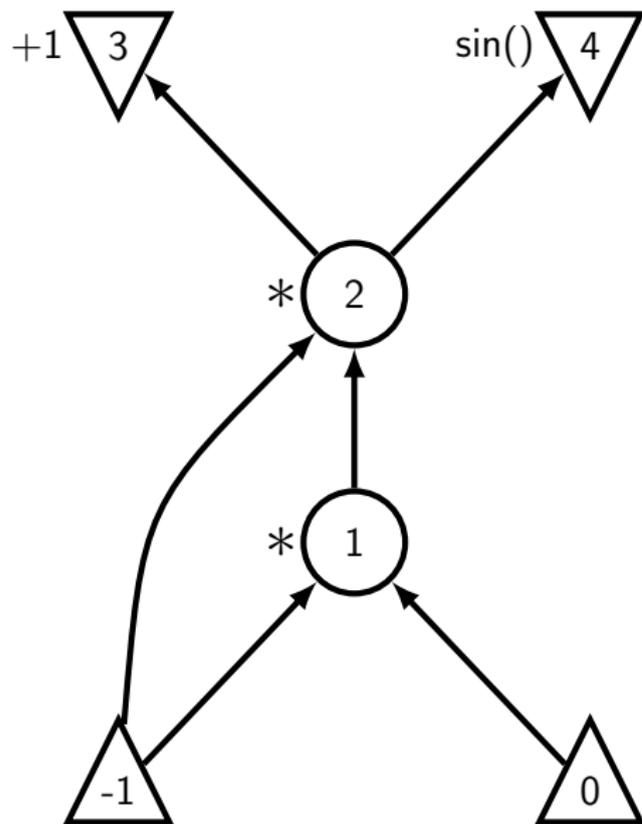
$$v_4 = \sin(v_2)$$

$$y_1 = v_3$$

$$y_2 = v_4$$

Computational Graphs and Evaluation Procedures

$$y_1 = x_1 * x_1 * x_2, \quad y_2 = \sin(x_1 * x_1 * x_2)$$



$$v_{-1} = x_1$$

$$v_0 = x_2$$

$$v_1 = v_{-1} * v_0$$

$$v_2 = v_{-1} * v_1$$

$$v_3 = v_2 + 1$$

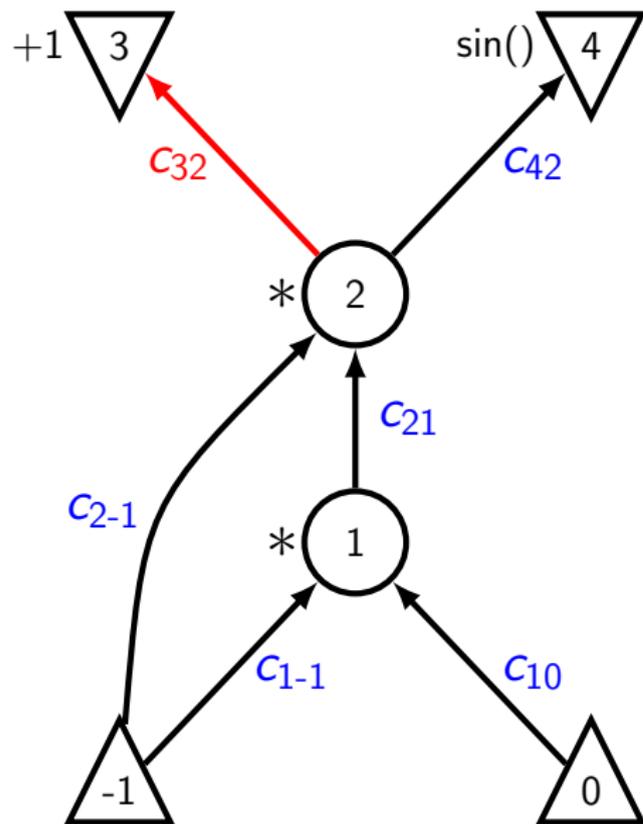
$$v_4 = \sin(v_2)$$

$$y_1 = v_3$$

$$y_2 = v_4$$

Computational Graphs and Linearization

$$y_1 = x_1 * x_1 * x_2, \quad y_2 = \sin(x_1 * x_1 * x_2)$$



$$v_{-1} = x_1$$

$$v_0 = x_2$$

$$v_1 = v_{-1} * v_0$$

$$C_{1-1} = v_0$$

$$C_{10} = v_1$$

$$v_2 = v_{-1} * v_1$$

$$C_{2-1} = v_1$$

$$C_{21} = v_{-1}$$

$$v_3 = v_2 + 1$$

$$C_{32} = 1$$

$$v_4 = \sin(v_2)$$

$$C_{42} = \cos(v_2)$$

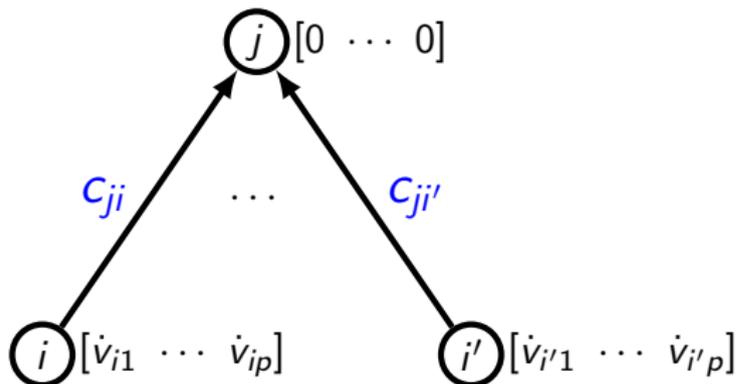
$$y_1 = v_3$$

$$y_2 = v_4$$

Forward Propagation of Vectors

Associate a derivative vector $\dot{\mathbf{v}}_j \in \mathbb{R}^p$ with each variable v_j , propagate through G by

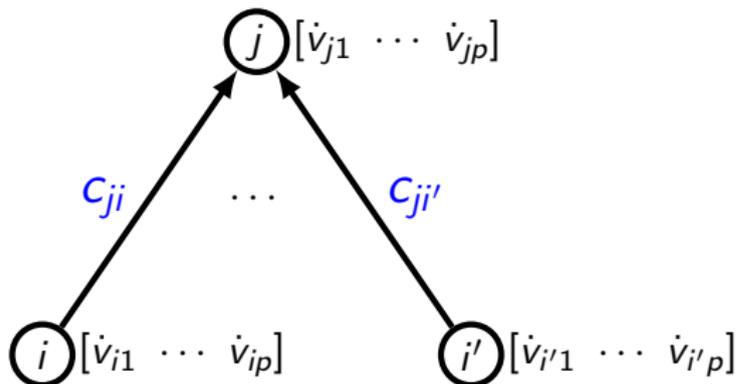
$$\dot{\mathbf{v}}_j = \sum_{i \prec j} c_{ji} \dot{\mathbf{v}}_i \quad (\text{BLAS level 1 axpy operation})$$



Forward Propagation of Vectors

Associate a derivative vector $\dot{\mathbf{v}}_j \in \mathbb{R}^p$ with each variable v_j , propagate through G by

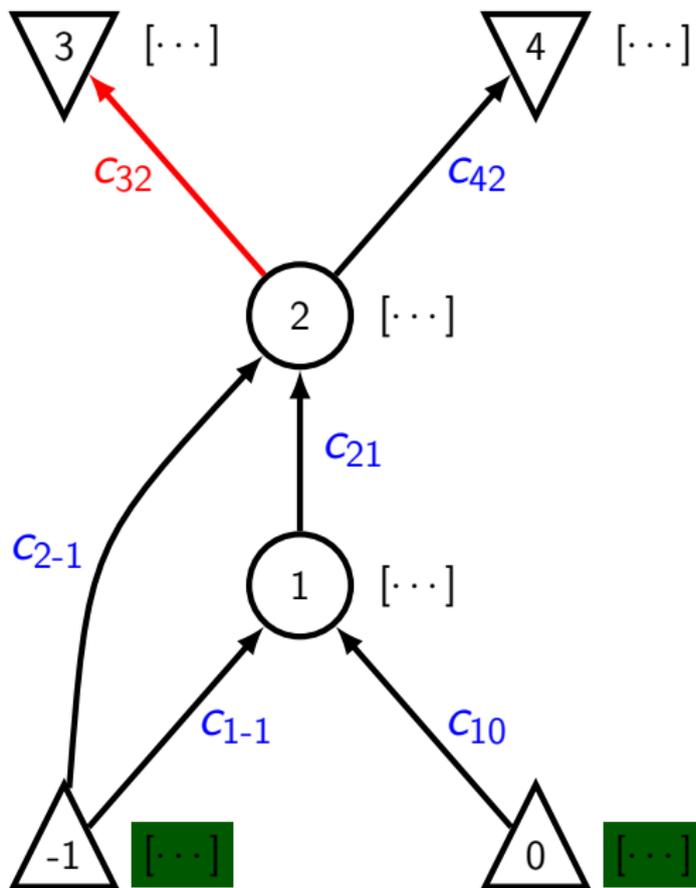
$$\dot{\mathbf{v}}_j = \sum_{i \prec j} c_{ji} \dot{\mathbf{v}}_i \quad (\text{BLAS level 1 axpy operation})$$



Generated propagation code:

$$\begin{aligned} \dot{\mathbf{v}}_j &= c_{ji} * \dot{\mathbf{v}}_i \\ &\vdots \\ \dot{\mathbf{v}}_j &+= c_{ji'} * \dot{\mathbf{v}}_{i'} \end{aligned}$$

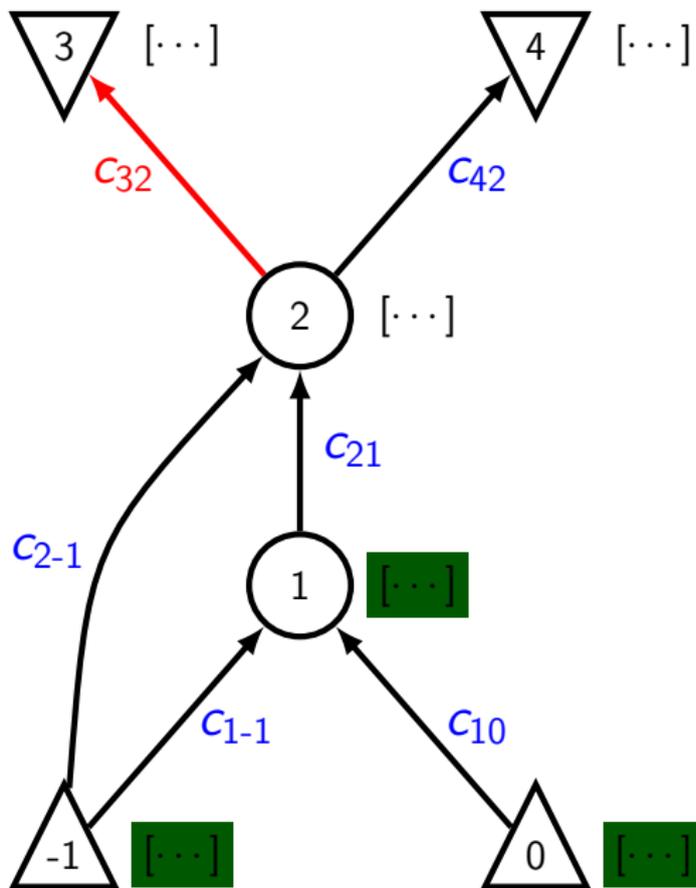
Forward Propagation of Vectors



$$\dot{v}_{-1} = \dot{x}_1$$

$$\dot{v}_0 = \dot{x}_2$$

Forward Propagation of Vectors



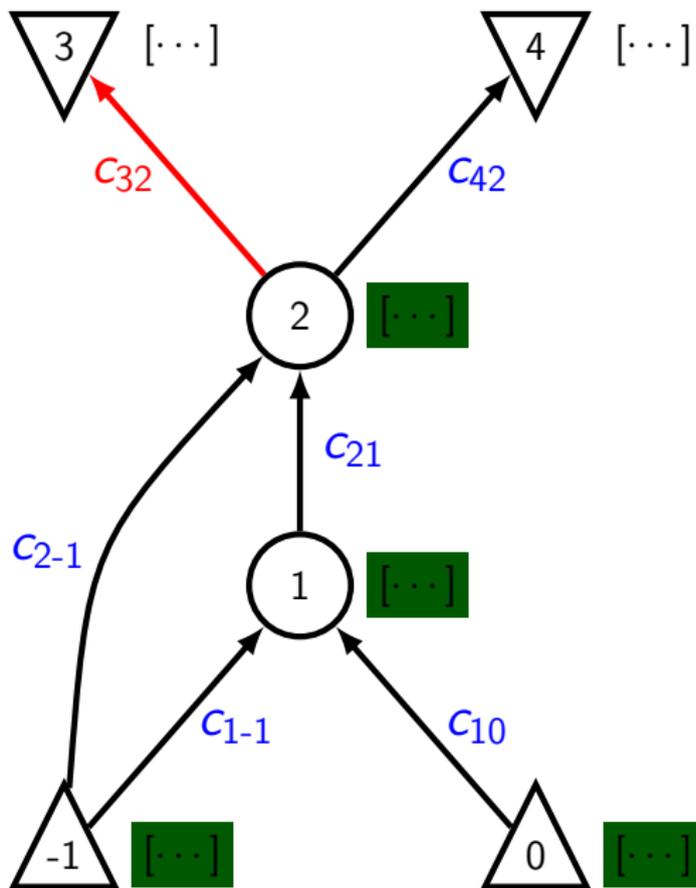
$$\dot{v}_{-1} = \dot{x}_1$$

$$\dot{v}_0 = \dot{x}_2$$

$$\dot{v}_1 = c_{1-1} * \dot{v}_{-1}$$

$$\dot{v}_1 += c_{10} * \dot{v}_0$$

Forward Propagation of Vectors



$$\dot{\mathbf{v}}_{-1} = \dot{\mathbf{x}}_1$$

$$\dot{\mathbf{v}}_0 = \dot{\mathbf{x}}_2$$

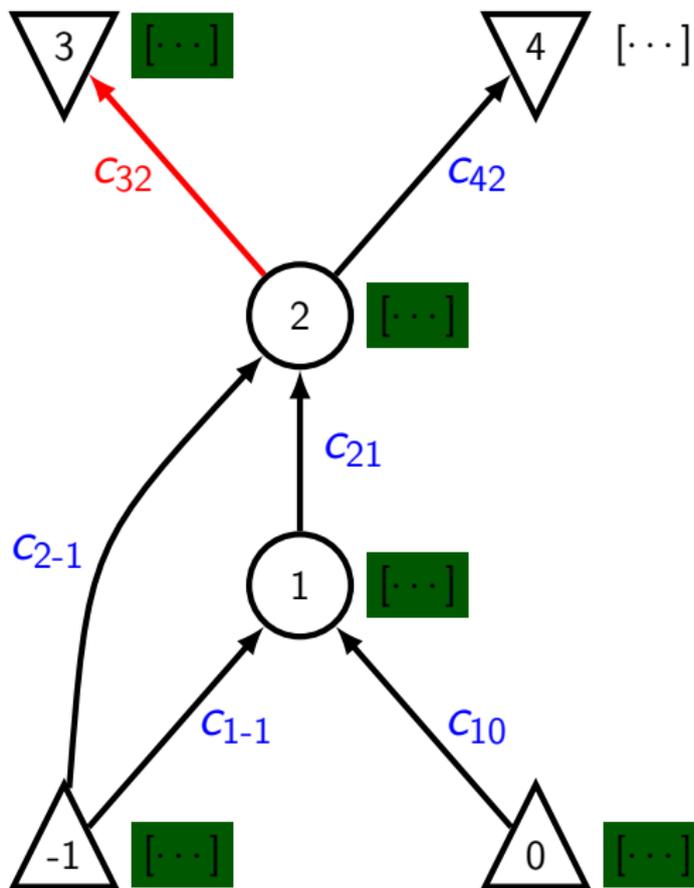
$$\dot{\mathbf{v}}_1 = c_{1-1} * \dot{\mathbf{v}}_{-1}$$

$$\dot{\mathbf{v}}_1 += c_{10} * \dot{\mathbf{v}}_0$$

$$\dot{\mathbf{v}}_2 = c_{2-1} * \dot{\mathbf{v}}_{-1}$$

$$\dot{\mathbf{v}}_2 += c_{21} * \dot{\mathbf{v}}_1$$

Forward Propagation of Vectors



$$\dot{v}_{-1} = \dot{x}_1$$

$$\dot{v}_0 = \dot{x}_2$$

$$\dot{v}_1 = c_{1-1} * \dot{v}_{-1}$$

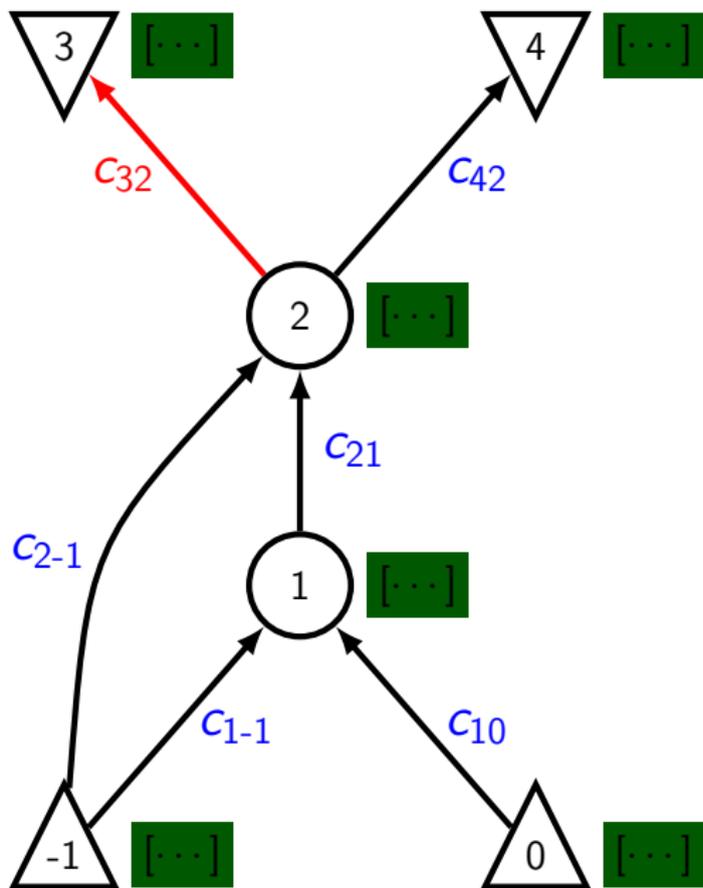
$$\dot{v}_1 += c_{10} * \dot{v}_0$$

$$\dot{v}_2 = c_{2-1} * \dot{v}_{-1}$$

$$\dot{v}_2 += c_{21} * \dot{v}_1$$

$$\dot{v}_3 = c_{32} * \dot{v}_2 = \dot{v}_2$$

Forward Propagation of Vectors



$$\dot{v}_{-1} = \dot{x}_1$$

$$\dot{v}_0 = \dot{x}_2$$

$$\dot{v}_1 = C_{1-1} * \dot{v}_{-1}$$

$$\dot{v}_1 += C_{10} * \dot{v}_0$$

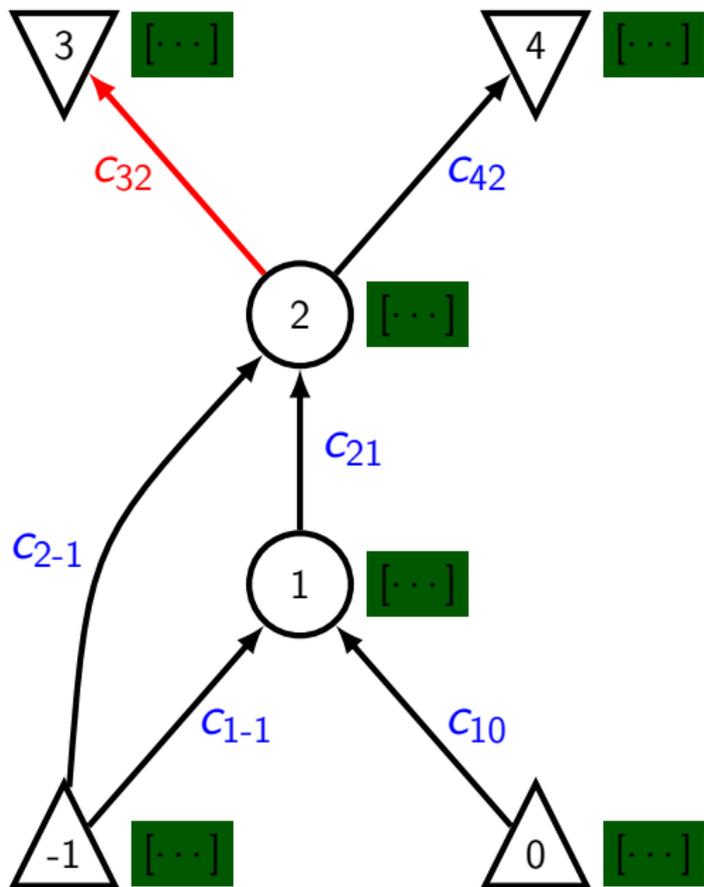
$$\dot{v}_2 = C_{2-1} * \dot{v}_{-1}$$

$$\dot{v}_2 += C_{21} * \dot{v}_1$$

$$\dot{v}_3 = C_{32} * \dot{v}_2 = \dot{v}_2$$

$$\dot{v}_4 = C_{42} * \dot{v}_2$$

Forward Propagation of Vectors



$$\dot{v}_{-1} = \dot{x}_1$$

$$\dot{v}_0 = \dot{x}_2$$

$$\dot{v}_1 = C_{1-1} * \dot{v}_{-1}$$

$$\dot{v}_1 += C_{10} * \dot{v}_0$$

$$\dot{v}_2 = C_{2-1} * \dot{v}_{-1}$$

$$\dot{v}_2 += C_{21} * \dot{v}_1$$

5p mults

$$\dot{v}_3 = C_{32} * \dot{v}_2 = \dot{v}_2$$

$$\dot{v}_4 = C_{42} * \dot{v}_2$$

$$\dot{y}_1 = \dot{v}_3$$

$$\dot{y}_2 = \dot{v}_4$$

Reverse Vector Propagation

Propagates vectors $\bar{\mathbf{y}}^1, \dots, \bar{\mathbf{y}}^p$ backwards

Works symmetrically

(same mult. cost, possibly different number of adds)

Yields Jacobian-transpose-vector products

$$F'(\mathbf{x})^T \bar{\mathbf{y}}^1, F'(\mathbf{x})^T \bar{\mathbf{y}}^2, \dots, F'(\mathbf{x})^T \bar{\mathbf{y}}^p$$

Preaccumulation

Cost is proportional to the number of nonunit edges \Rightarrow transform the graph!

Baur's formula (from chain rule) yields the entries of $F'(\mathbf{x})$

$$\frac{\partial y_j}{\partial x_i} = \sum_{P \in \mathcal{P}_{x_i}^{y_j}} \prod_{(k,\ell) \in P} c_{\ell k}$$

Preaccumulation

Cost is proportional to the number of nonunit edges \Rightarrow transform the graph!

Baur's formula (from chain rule) yields the entries of $F'(\mathbf{x})$

$$\frac{\partial y_j}{\partial x_i} = \sum_{P \in \mathcal{P}_{x_i}^{y_j}} \prod_{(k,\ell) \in P} c_{\ell k}$$

Preaccumulation applies transformations $G \rightarrow G'$

Afterwards, Baur's formula still expresses the entries of J (we can still propagate vectors through it)

Preaccumulation

Cost is proportional to the number of nonunit edges \Rightarrow transform the graph!

Baur's formula (from chain rule) yields the entries of $F'(\mathbf{x})$

$$\frac{\partial y_j}{\partial x_i} = \sum_{P \in \mathcal{P}_{x_i}^{y_j}} \prod_{(k,\ell) \in P} c_{\ell k}$$

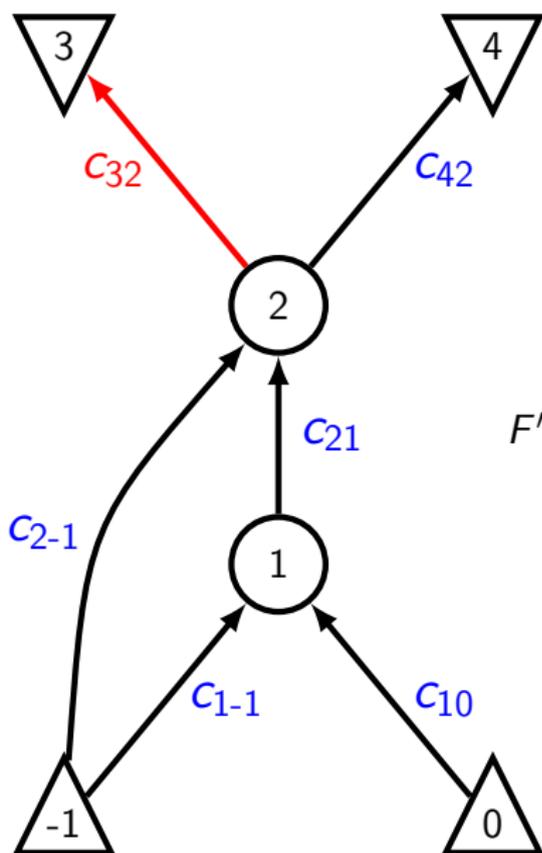
Preaccumulation applies transformations $G \rightarrow G'$

Afterwards, Baur's formula still expresses the entries of J (we can still propagate vectors through it)

Complete preaccumulation results in a bipartite graph, whose edges correspond to the nonzero entries of $F'(\mathbf{x})$

(In general, complete preaccumulation with minimal ops (OJA) is **NP-hard**)

Baur's Formula

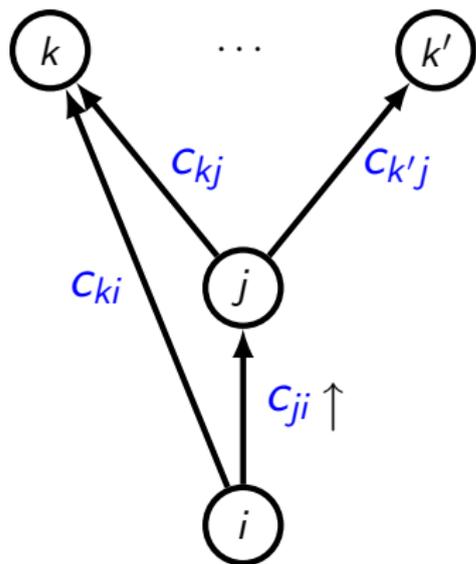


$$\frac{\partial y_j}{\partial x_i} = \sum_{P \in \mathcal{P}_{x_i}^{y_j}} \prod_{(k,\ell) \in P} c_{\ell k}$$

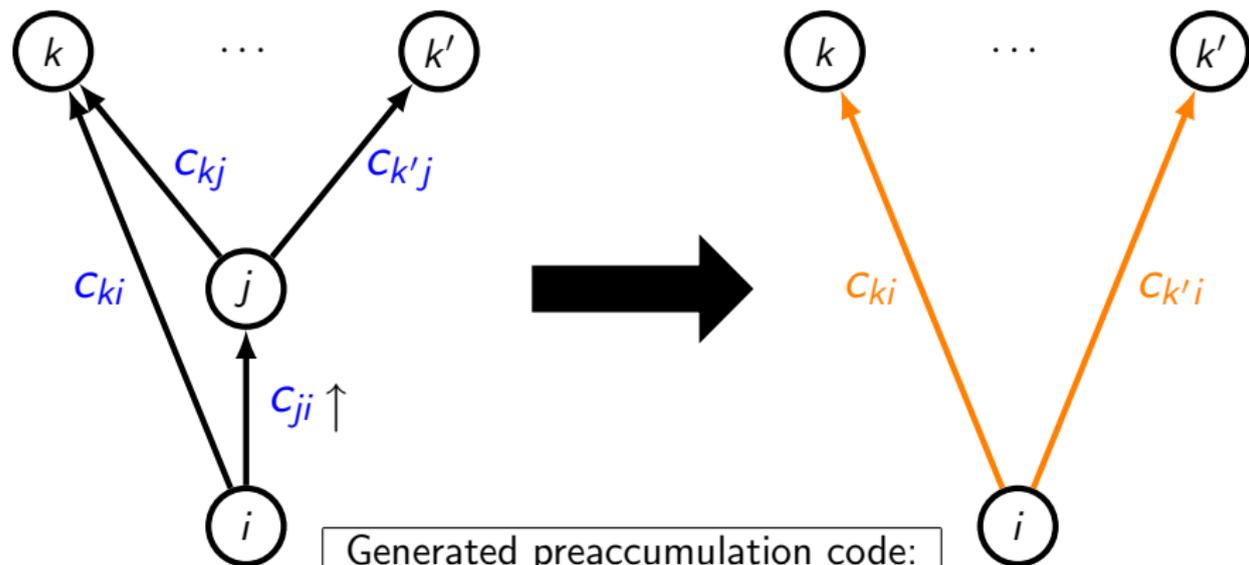
$$F'(\mathbf{x}) = \begin{bmatrix} c_{2,-1}c_{3,2} + c_{1,-1}c_{2,1}c_{3,2} & c_{1,0}c_{2,1}c_{3,2} \\ c_{2,-1}c_{4,2} + c_{1,-1}c_{2,1}c_{4,2} & c_{1,0}c_{2,1}c_{4,2} \end{bmatrix}$$

$$= \begin{bmatrix} c_{2,-1} + c_{1,-1}c_{2,1} & c_{1,0}c_{2,1} \\ c_{2,-1}c_{4,2} + c_{1,-1}c_{2,1}c_{4,2} & c_{1,0}c_{2,1}c_{4,2} \end{bmatrix}$$

Front Edge Elimination



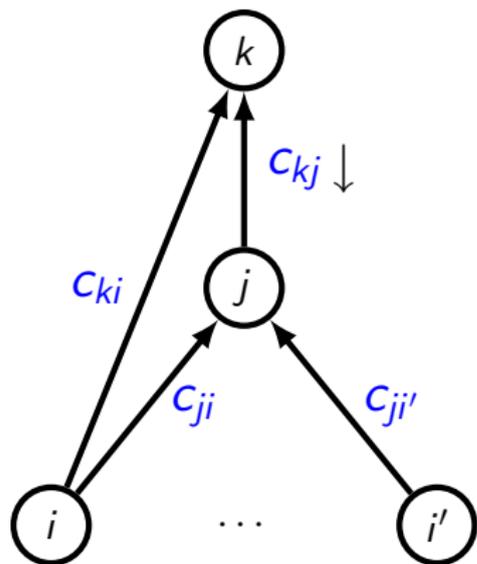
Front Edge Elimination



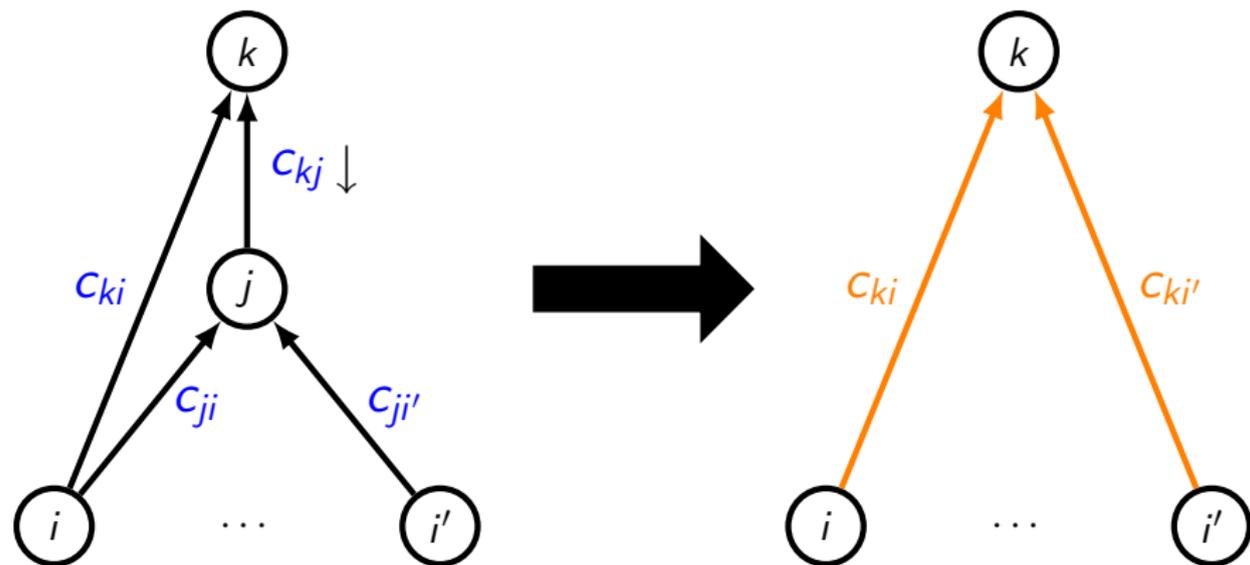
Generated preaccumulation code:

$$\begin{aligned} c_{ki} & += c_{ji} * c_{kj} \\ & \vdots \\ c_{k'i} & = c_{ji} * c_{k'j} \end{aligned}$$

Back Edge Elimination



Back Edge Elimination

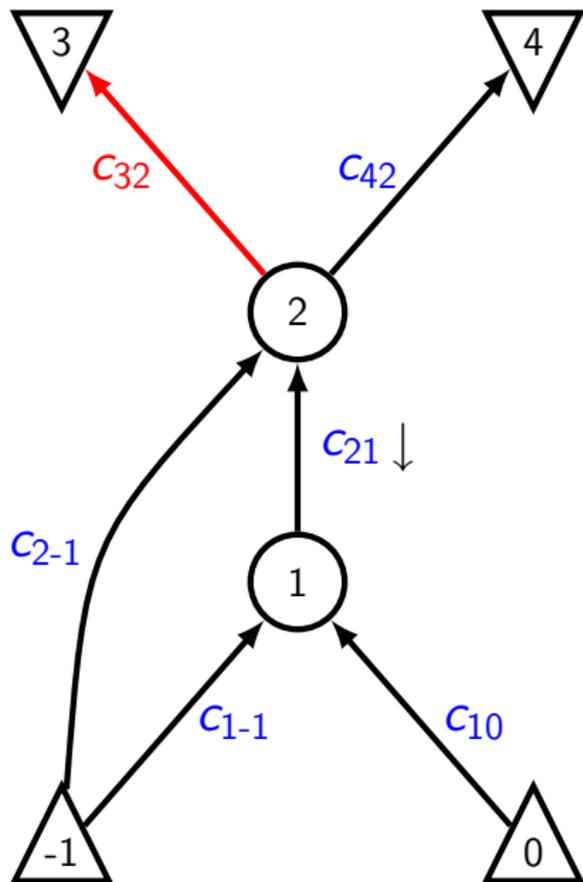


$$c_{ki} += c_{ji} * c_{kj}$$

$$\vdots$$

$$c_{ki'} = c_{ji'} * c_{kj}$$

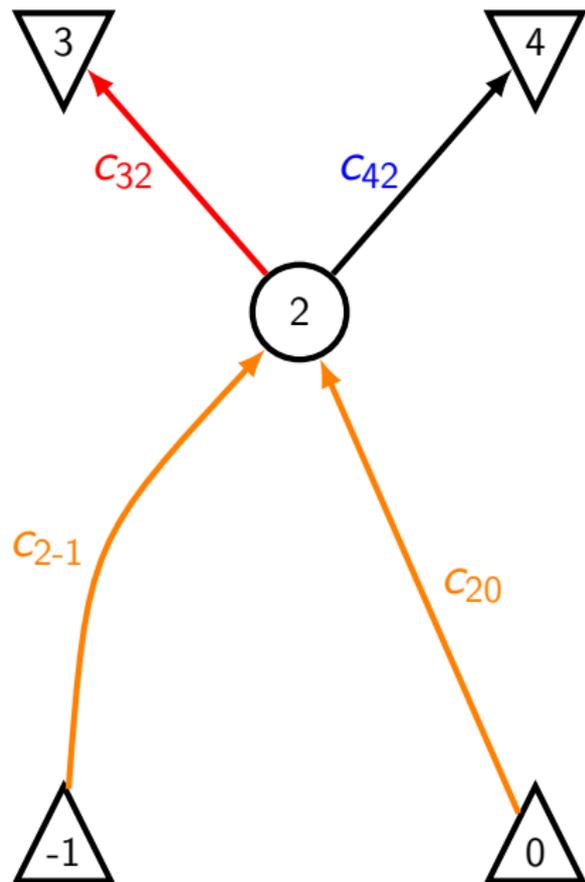
Preaccumulation



(Code for F , linearization)

⋮

Preaccumulation



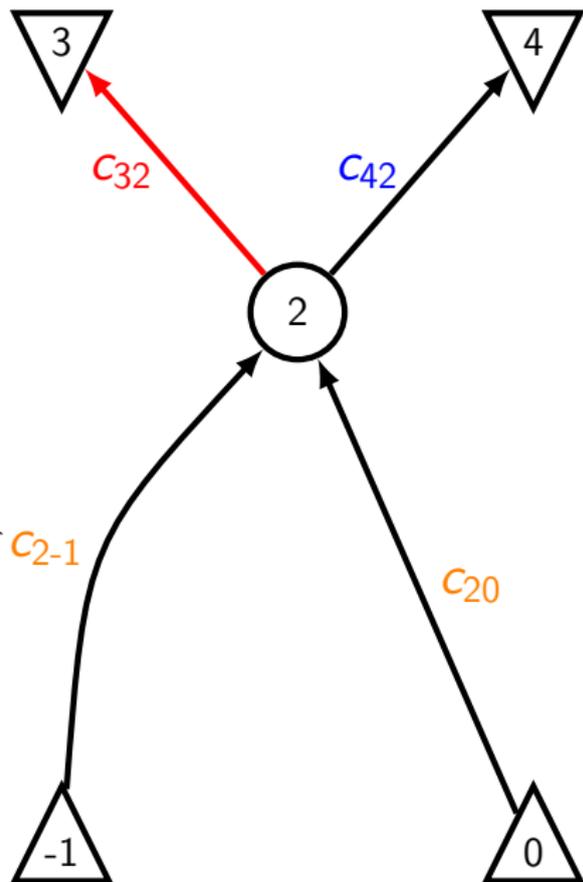
(Code for F , linearization)

\vdots

$$c_{2-1} = c_{1-1} * c_{21}$$

$$c_{20} = c_{10} * c_{21}$$

Preaccumulation



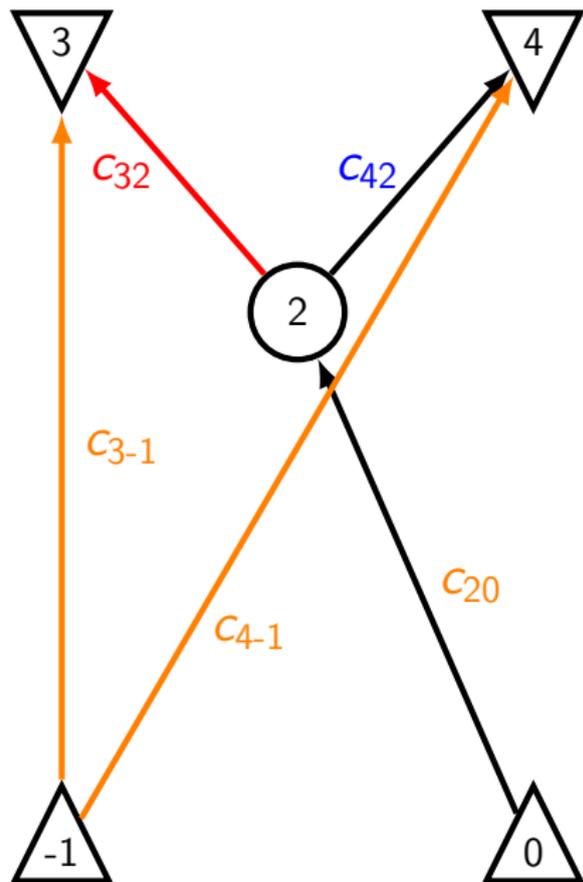
(Code for F , linearization)

\vdots

$$c_{2-1} = c_{1-1} * c_{21}$$

$$c_{20} = c_{10} * c_{21}$$

Preaccumulation



(Code for F , linearization)

\vdots

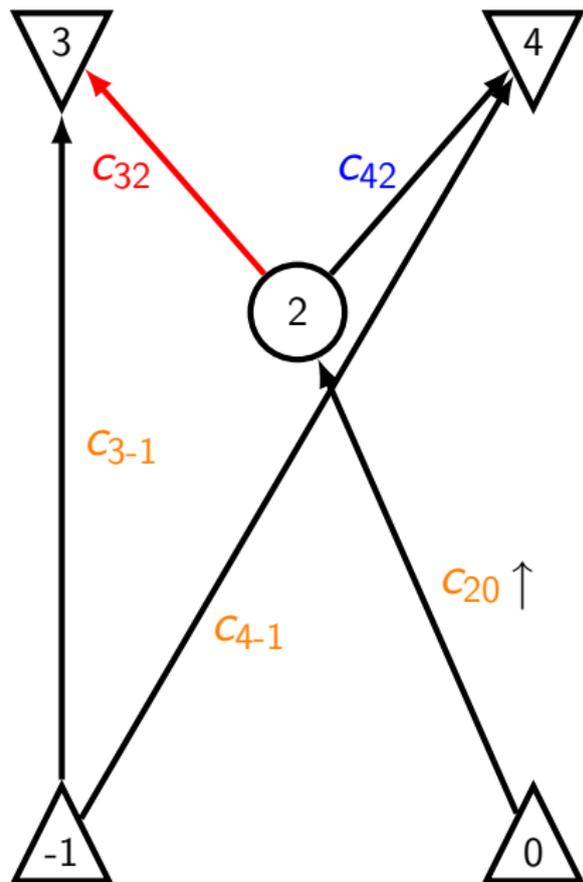
$$C_{2-1} = C_{1-1} * C_{21}$$

$$C_{20} = C_{10} * C_{21}$$

$$C_{3-1} = C_{2-1} * C_{32}$$

$$C_{4-1} = C_{2-1} * C_{42}$$

Preaccumulation



(Code for F , linearization)

\vdots

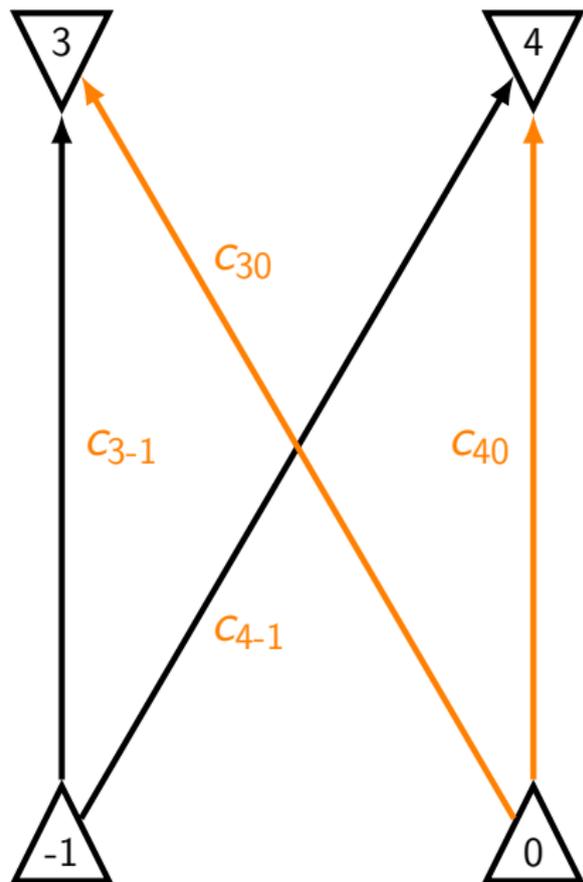
$$c_{2-1} = c_{1-1} * c_{21}$$

$$c_{20} = c_{10} * c_{21}$$

$$c_{3-1} = c_{2-1} * c_{32}$$

$$c_{4-1} = c_{2-1} * c_{42}$$

Preaccumulation



(Code for F , linearization)

⋮

$$C_{2-1} = C_{1-1} * C_{21}$$

$$C_{20} = C_{10} * C_{21}$$

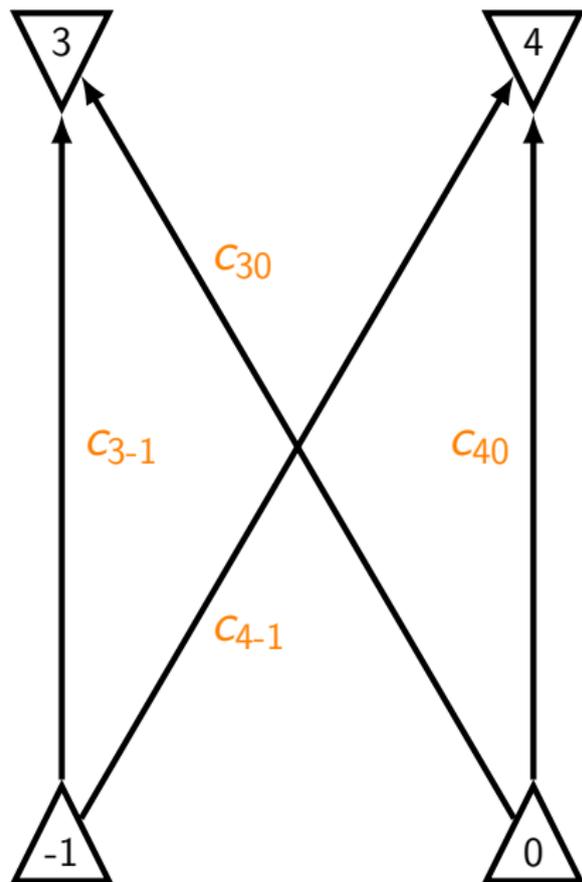
$$C_{3-1} = C_{2-1} * C_{32}$$

$$C_{4-1} = C_{2-1} * C_{42}$$

$$C_{30} = C_{20} * C_{32}$$

$$C_{40} = C_{20} * C_{42}$$

Preaccumulation



(Code for F , linearization)

\vdots

$$C_{2-1} = C_{1-1} * C_{21}$$

$$C_{20} = C_{10} * C_{21}$$

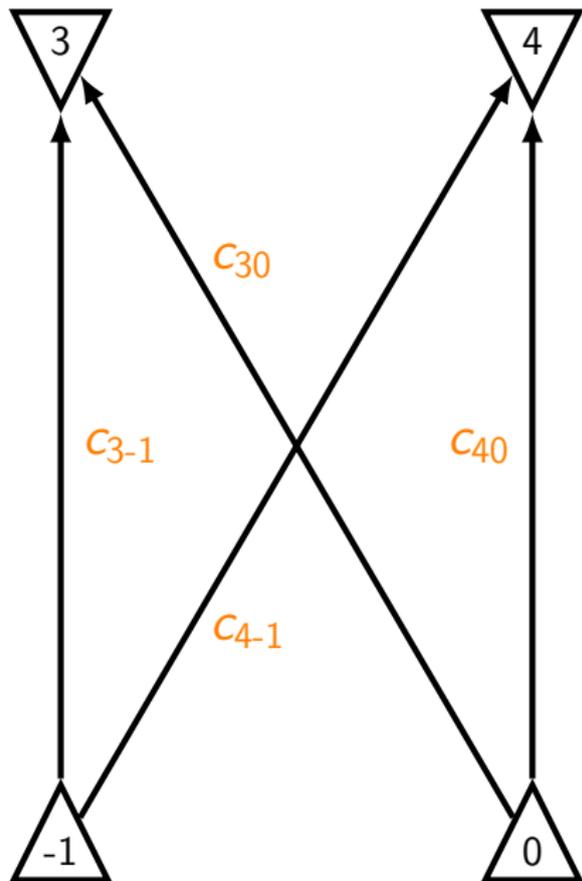
$$C_{3-1} = C_{2-1} * C_{32}$$

$$C_{4-1} = C_{2-1} * C_{42}$$

$$C_{30} = C_{20} * C_{32}$$

$$C_{40} = C_{20} * C_{42}$$

Preaccumulation



(Code for F , linearization)

\vdots

$$C_{2-1} = C_{1-1} * C_{21}$$

$$C_{20} = C_{10} * C_{21}$$

$$C_{3-1} = C_{2-1} * C_{32}$$

$$C_{4-1} = C_{2-1} * C_{42}$$

4 mults

$$C_{30} = C_{20} * C_{32}$$

$$C_{40} = C_{20} * C_{42}$$

$$\dot{V}_3 = C_{3-1} * \dot{V}_{-1}$$

$$\dot{V}_3 += C_{30} * \dot{V}_0$$

4p mults

$$\dot{V}_4 = C_{4-1} * \dot{V}_{-1}$$

$$\dot{V}_4 += C_{40} * \dot{V}_0$$

Costs

Fixed costs: Evaluation of F and linearization

Costs

Fixed costs: Evaluation of F and linearization

Propagation:

$5p$ multiplications

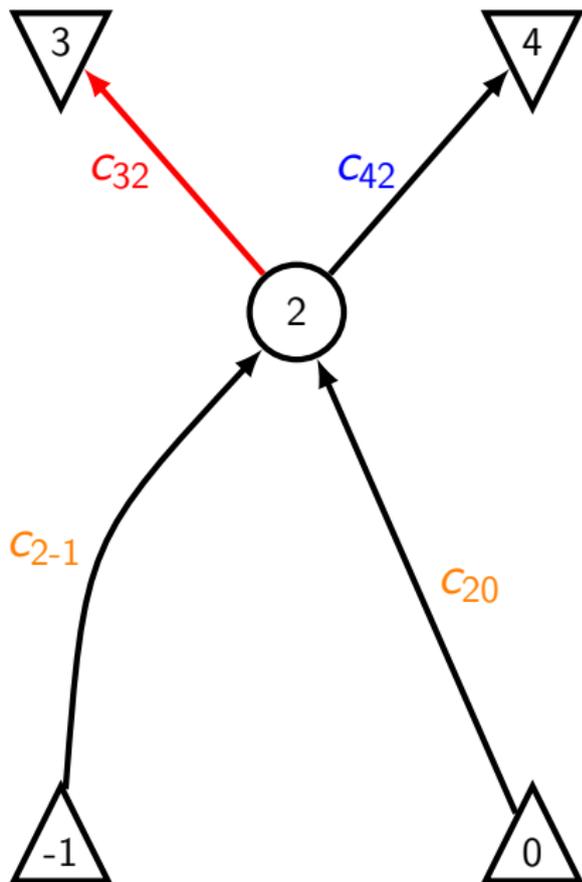
Costs

Fixed costs: Evaluation of F and linearization

Propagation: $5p$ multiplications

Complete Preaccumulation + Propagation: $4 + 4p$ multiplications

Partial Preaccumulation



(Code for F , linearization)

\vdots

$$C_{2-1} = C_{1-1} * C_{21} \quad 2 \text{ mults}$$

$$C_{20} = C_{10} * C_{21}$$

$$\dot{v}_2 = C_{2-1} * \dot{v}_{-1}$$

$$\dot{v}_2 += C_{20} * \dot{v}_0$$

$$\dot{v}_3 = C_{32} * \dot{v}_2 = \dot{v}_2$$

$$\dot{v}_4 = C_{42} * \dot{v}_2$$

$3p$ mults

Costs

Fixed costs: Evaluation of F and linearization

Propagation: $5p$ multiplications

Complete Preaccumulation + Propagation: $4 + 4p$ multiplications

Partial Preaccumulation + Propagation: $2 + 3p$ multiplications

\Rightarrow Assume p is large, so ignore preaccumulation cost and focus on propagation cost.

Outline

Introduction and Motivation

Linearization

Vector Propagation

Preaccumulation

Jacobian Scarcity

Structural Properties of Jacobians

Randomized Algorithms

Results

Rerouting and Normalization

Future Work

Connections to Optimal Jacobian Accumulation

Leveraging Face Elimination

Jacobian Scarcity

Example Jacobian happens to be dense, but some structure is lost when $F'(\mathbf{x})$ is accumulated to a matrix.

For example, the Jacobian is low rank ($\#$ of vertex-disjoint paths).

\Rightarrow Jacobian **scarcity** (Griewank)

Jacobian Scarcity

Example Jacobian happens to be dense, but some structure is lost when $F'(\mathbf{x})$ is accumulated to a matrix.

For example, the Jacobian is low rank ($\#$ of vertex-disjoint paths).

\Rightarrow Jacobian **scarcity** (Griewank)

Scarcity is a kind of deficiency, approximated by the number of nonunit edges in G

Graph transformations that *don't increase* the number of nonunit edges are said to be **scarcity preserving**.

Jacobian Scarcity

Example Jacobian happens to be dense, but some structure is lost when $F'(\mathbf{x})$ is accumulated to a matrix.

For example, the Jacobian is low rank ($\#$ of vertex-disjoint paths).

\Rightarrow Jacobian **scarcity** (Griewank)

Scarcity is a kind of deficiency, approximated by the number of nonunit edges in G

Graph transformations that *don't increase* the number of nonunit edges are said to be **scarcity preserving**.

\Rightarrow Exploiting Scarcity: finding minimal representation of $G(F'(\mathbf{x}))$

Jacobian Scarcity

Example Jacobian happens to be dense, but some structure is lost when $F'(\mathbf{x})$ is accumulated to a matrix.

For example, the Jacobian is low rank ($\#$ of vertex-disjoint paths).

\Rightarrow Jacobian **scarcity** (Griewank)

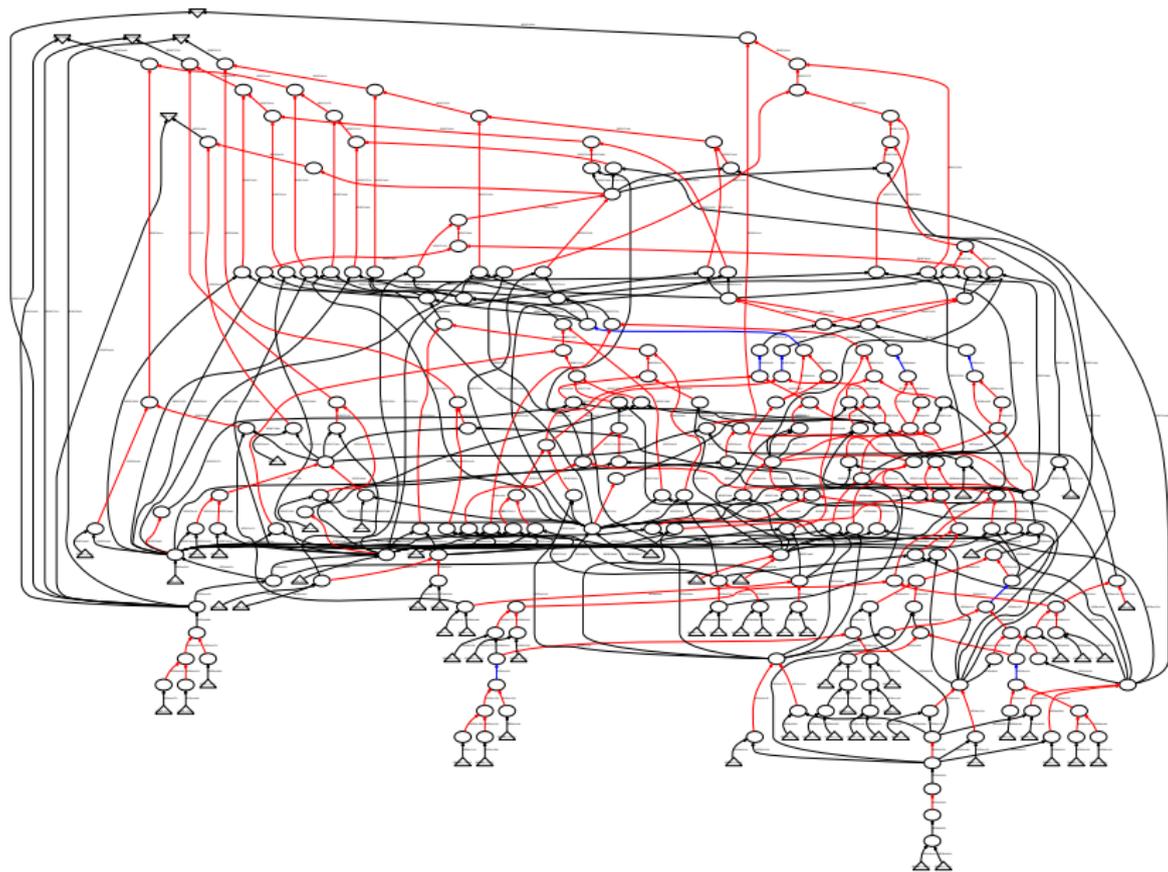
Scarcity is a kind of deficiency, approximated by the number of nonunit edges in G

Graph transformations that *don't increase* the number of nonunit edges are said to be **scarcity preserving**.

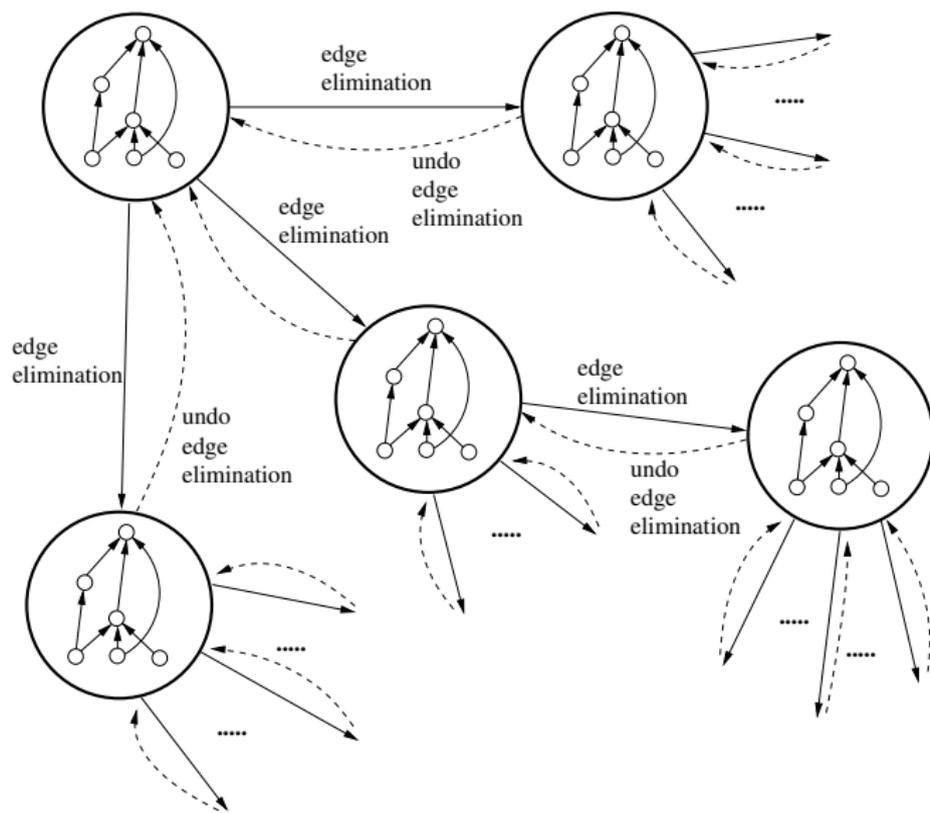
\Rightarrow Exploiting Scarcity: finding minimal representation of $G(F'(\mathbf{x}))$

Greedy Algorithm - Lyons & Utke (AD2008)

Scarcity in Practice



Metagraph $M(G)$



Transitions in $M(G) \leftrightarrow$ change in number of nonunit edges $|E^+|$

Randomized Traversal of the Metagraph

Simple random walk: yields up to 10% improvement over greedy algorithm

Randomized Traversal of the Metagraph

Simple random walk: yields up to 10% improvement over greedy algorithm

... but it takes a LONG TIME (weeks?)

Randomized Traversal of the Metagraph

Simple random walk: yields up to 10% improvement over greedy algorithm

... but it takes a LONG TIME (weeks?)

Nice proof of concept, but can we do better? (I hope so!)

Simulated Annealing vs. Metropolis

Assign each transition (including backwards) in the metagraph a probability based on the change in energy δE

$$Pr = e^{-\frac{\delta E}{kT}}$$

where T is a temperature and k is a constant.

- ▶ Simulated Annealing: gradual heating/cooling scheme up to 20% improvement
- ▶ Metropolis: fixed T – up to 35%+ improvement

Simulated Annealing vs. Metropolis

“Surprisingly enough, many problems cannot be solved more efficiently by SA than by the Metropolis.”

–Wegener, “SA beats Metropolis in Combinatorial Optimization”,
Electronic Colloquium on Computational Complexity, 2004.

Simulated Annealing vs. Metropolis

“Surprisingly enough, many problems cannot be solved more efficiently by SA than by the Metropolis.”

–Wegener, “SA beats Metropolis in Combinatorial Optimization”,
Electronic Colloquium on Computational Complexity, 2004.

Our conclusion: Our best results were obtained with a fixed temperature scheme (Metropolis). Variations in temperature don't appear to help much (plus adds additional parameter).

Simulated Annealing vs. Metropolis

“Surprisingly enough, many problems cannot be solved more efficiently by SA than by the Metropolis.”

–Wegener, “SA beats Metropolis in Combinatorial Optimization”,
Electronic Colloquium on Computational Complexity, 2004.

Our conclusion: Our best results were obtained with a fixed temperature scheme (Metropolis). Variations in temperature don't appear to help much (plus adds additional parameter).

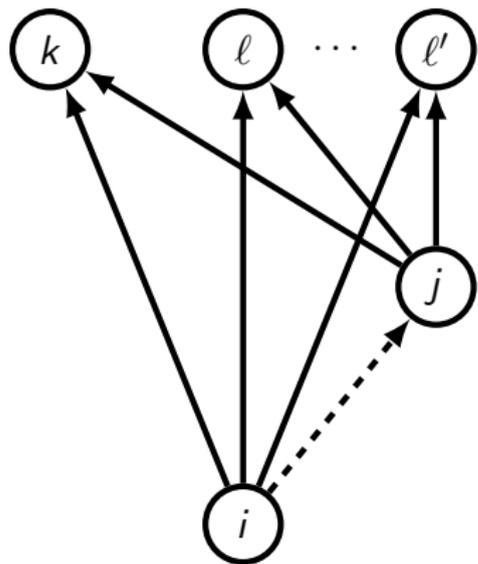
- ▶ Bound the running time scheme by $100|V_{orig}|$ transitions
- ▶ Small number (5 or so) of restarts
- ▶ Runs in minutes on a laptop (not performed at runtime)

Results

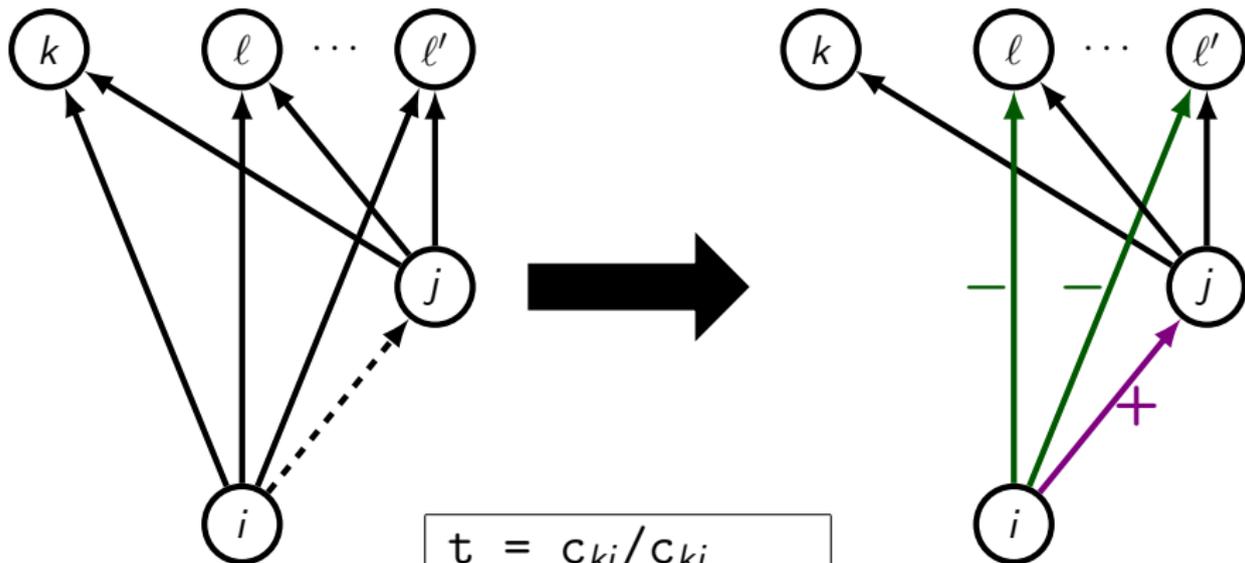
	Original	Greedy Alg.	Random Alg.
I	$259p$	$119 + 231p$	$146 + 222p$
II	$108p$	$34 + 93p$	$129 + 83p$
III	$241p$	$372 + 185p$	$780 + 140p$

(Metric is multiplications performed when propagating p vectors.)

Edge Prerouting



Edge Prerouting



$$t = c_{ki} / c_{kj}$$

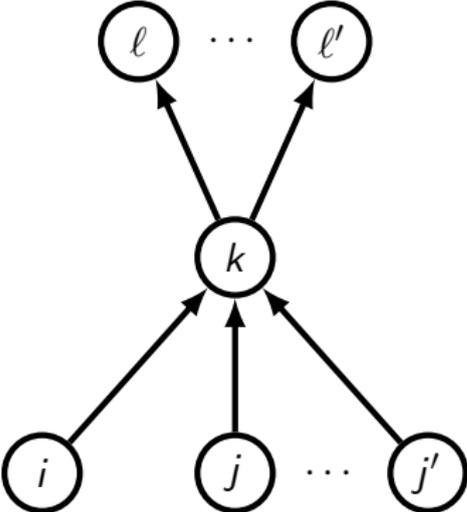
$$c_{ji} += t$$

$$c_{li} -= c_{lj} * t$$

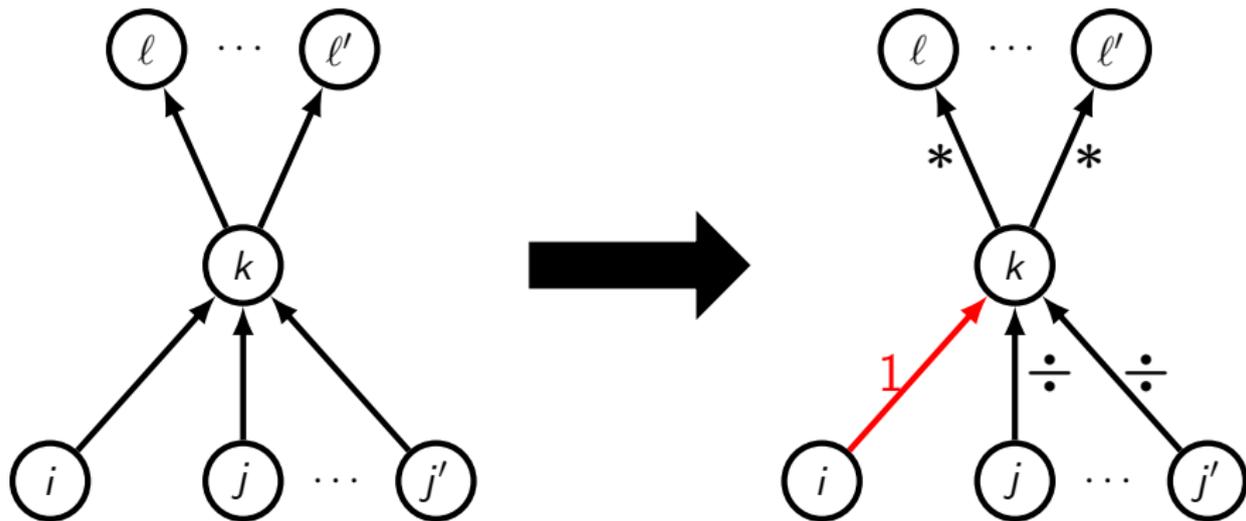
$$\vdots$$

$$c_{l'i} -= c_{l'i} * t$$

Edge Normalization Forward



Edge Normalization Forward



$$c_{kj} \neq c_{ki}$$

$$\vdots$$

$$c_{kj'} \neq c_{ki}$$

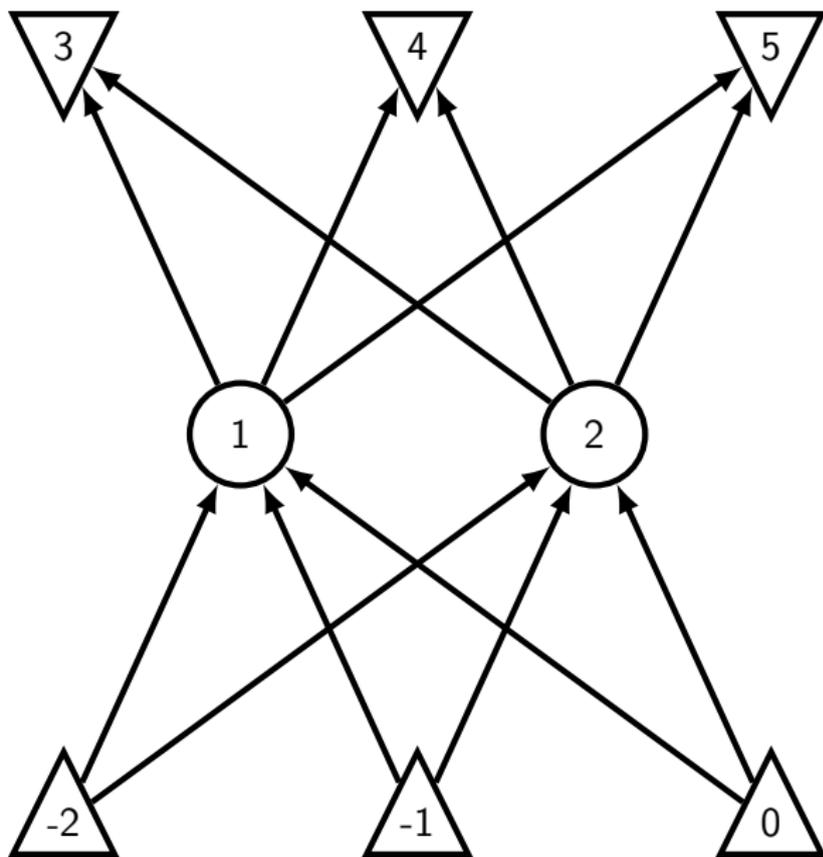
$$c_{ki} = 1$$

$$c_{lk} * = c_{ki}$$

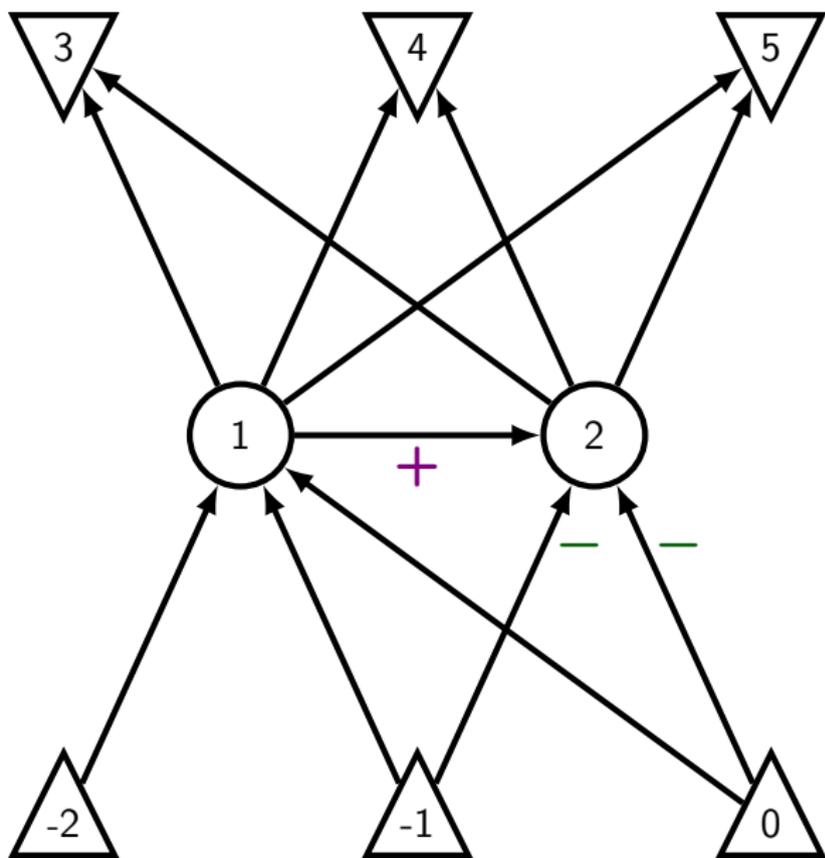
$$\vdots$$

$$c_{l'k} * = c_{ki}$$

Example

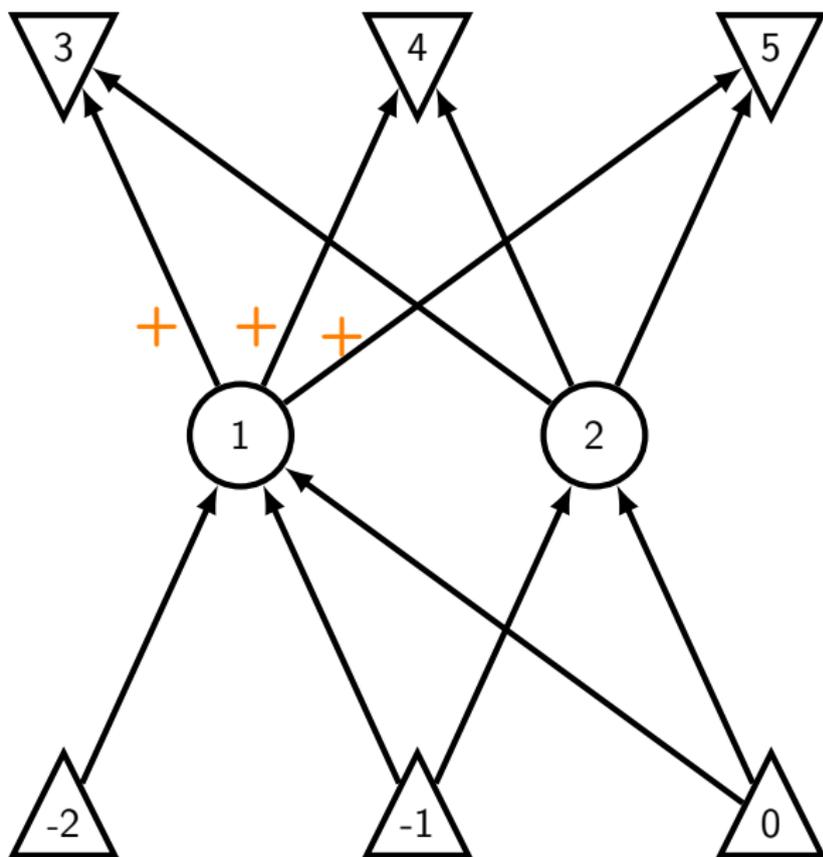


Example



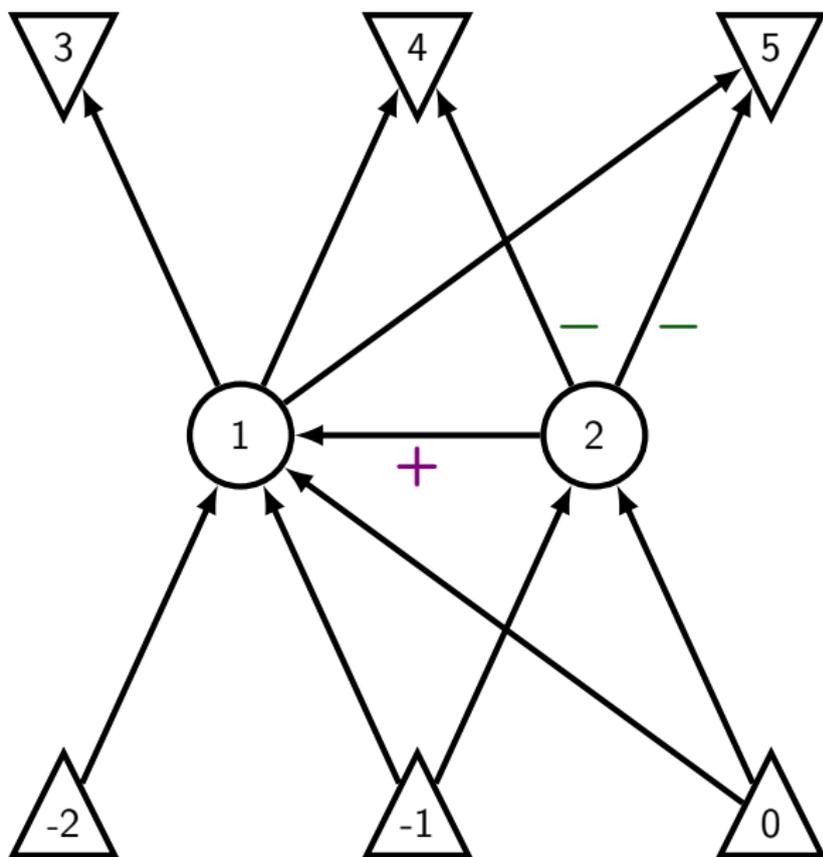
► Postroute $(-2, 2)$

Example



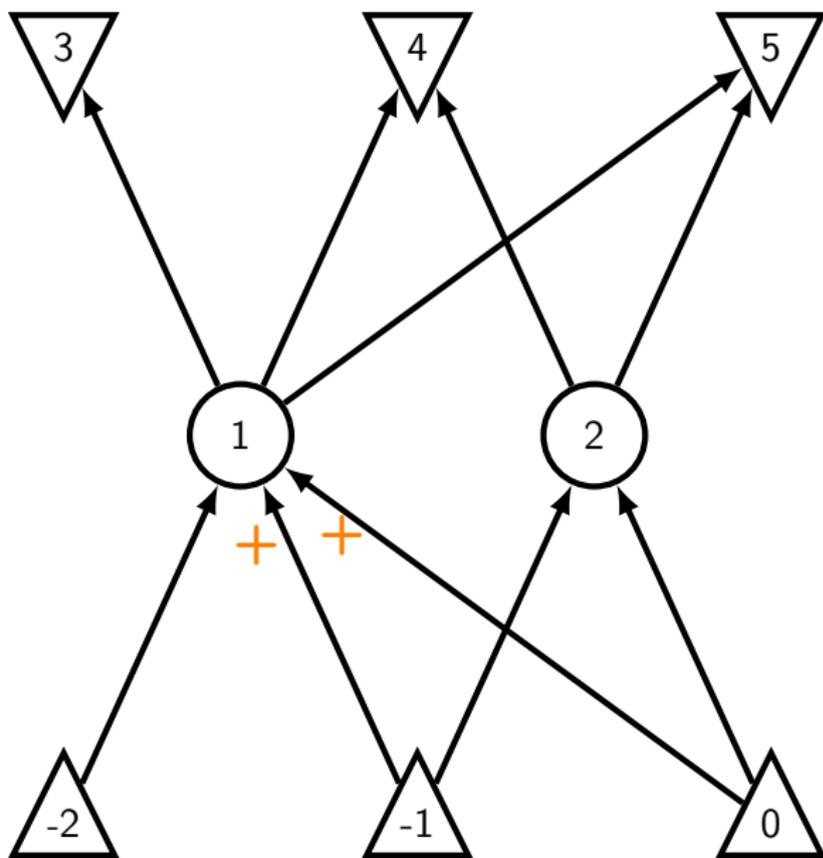
- ▶ Postroute $(-2, 2)$
- ▶ Front Eliminate $(1, 2)$

Example



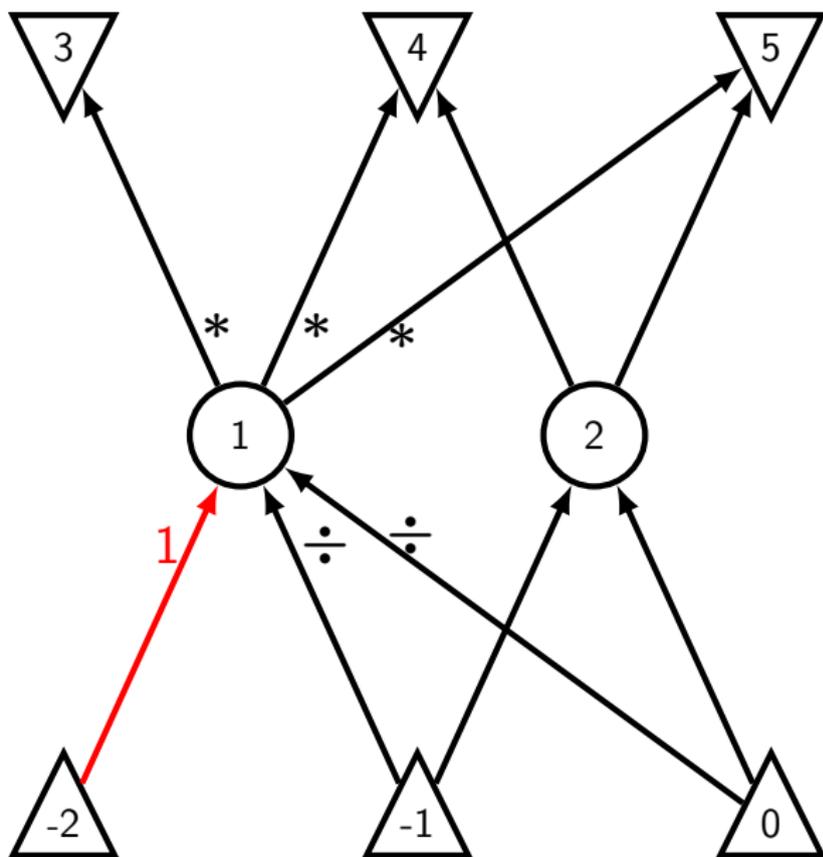
- ▶ Postroute $(-2, 2)$
- ▶ Front Eliminate $(1, 2)$
- ▶ Preroute $(2, 3)$

Example



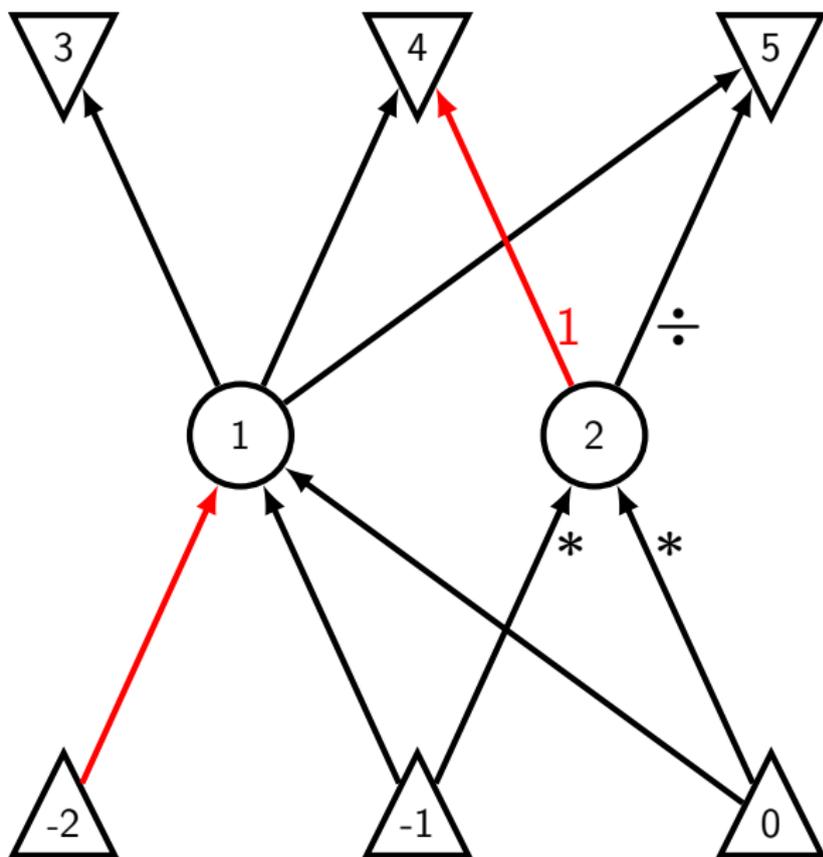
- ▶ Postroute $(-2, 2)$
- ▶ Front Eliminate $(1, 2)$
- ▶ Preroute $(2, 3)$
- ▶ Back Eliminate $(2, 1)$

Example



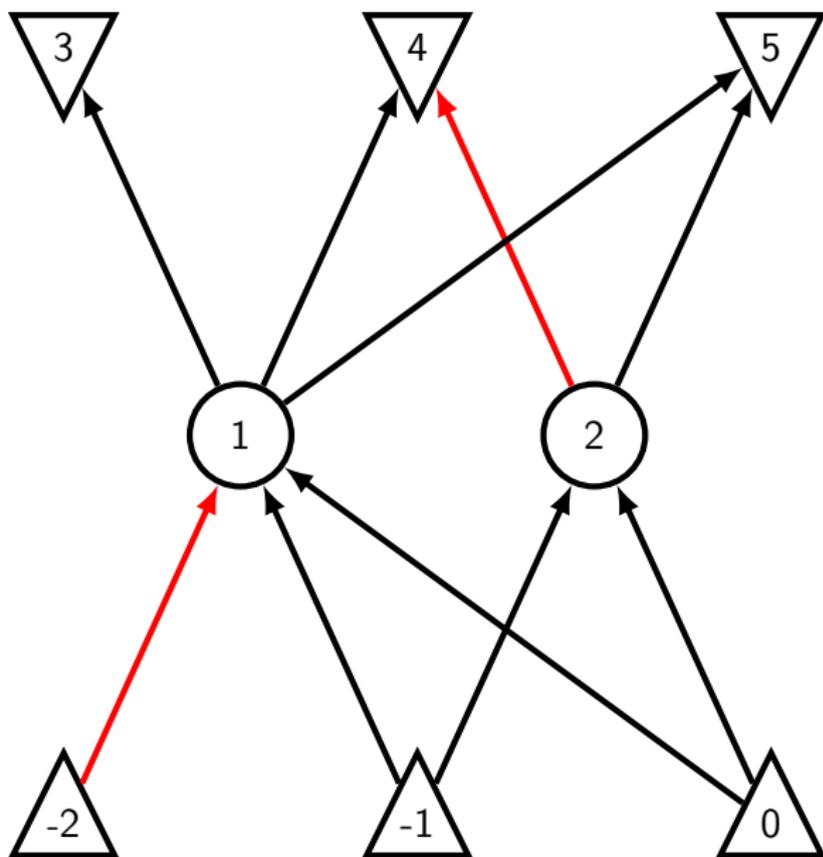
- ▶ Postroute $(-2, 2)$
- ▶ Front Eliminate $(1, 2)$
- ▶ Preroute $(2, 3)$
- ▶ Back Eliminate $(2, 1)$
- ▶ Normalize $(-2, 1)$

Example



- ▶ Postroute $(-2, 2)$
- ▶ Front Eliminate $(1, 2)$
- ▶ Preroute $(2, 3)$
- ▶ Back Eliminate $(2, 1)$
- ▶ Normalize $(-2, 1)$
- ▶ Normalize $(2, 4)$

Example



- ▶ Postroute $(-2, 2)$
- ▶ Front Eliminate $(1, 2)$
- ▶ Preroute $(2, 3)$
- ▶ Back Eliminate $(2, 1)$
- ▶ Normalize $(-2, 1)$
- ▶ Normalize $(2, 4)$

\Rightarrow 8 nonunit edges

Are divisions Useful?

Possibly, but **we don't know how to use them**

Greedy Algorithm - reroutings lead to *small* improvement at great cost (still not as good as randomized algorithm)

Randomized algorithm - reroutings don't appear to help

Outline

Introduction and Motivation

- Linearization

- Vector Propagation

- Preaccumulation

Jacobian Scarcity

- Structural Properties of Jacobians

- Randomized Algorithms

- Results

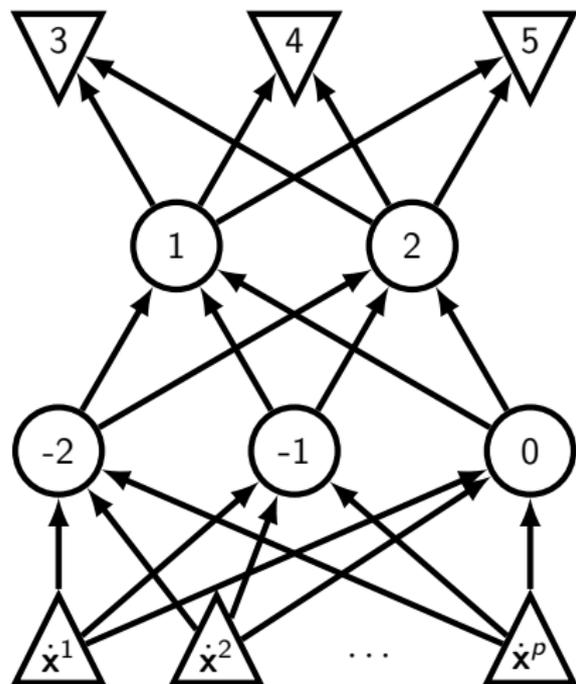
- Rerouting and Normalization

Future Work

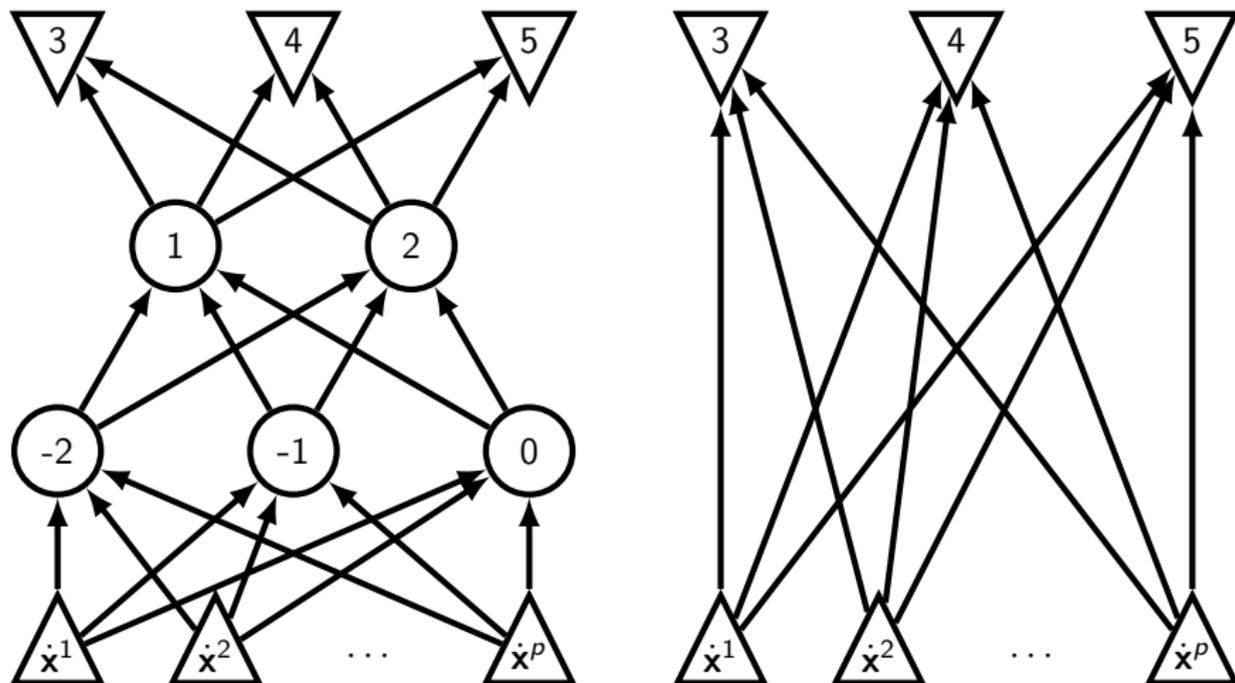
- Connections to Optimal Jacobian Accumulation

- Leveraging Face Elimination

Scarcity \leftrightarrow Optimal Jacobian Accumulation

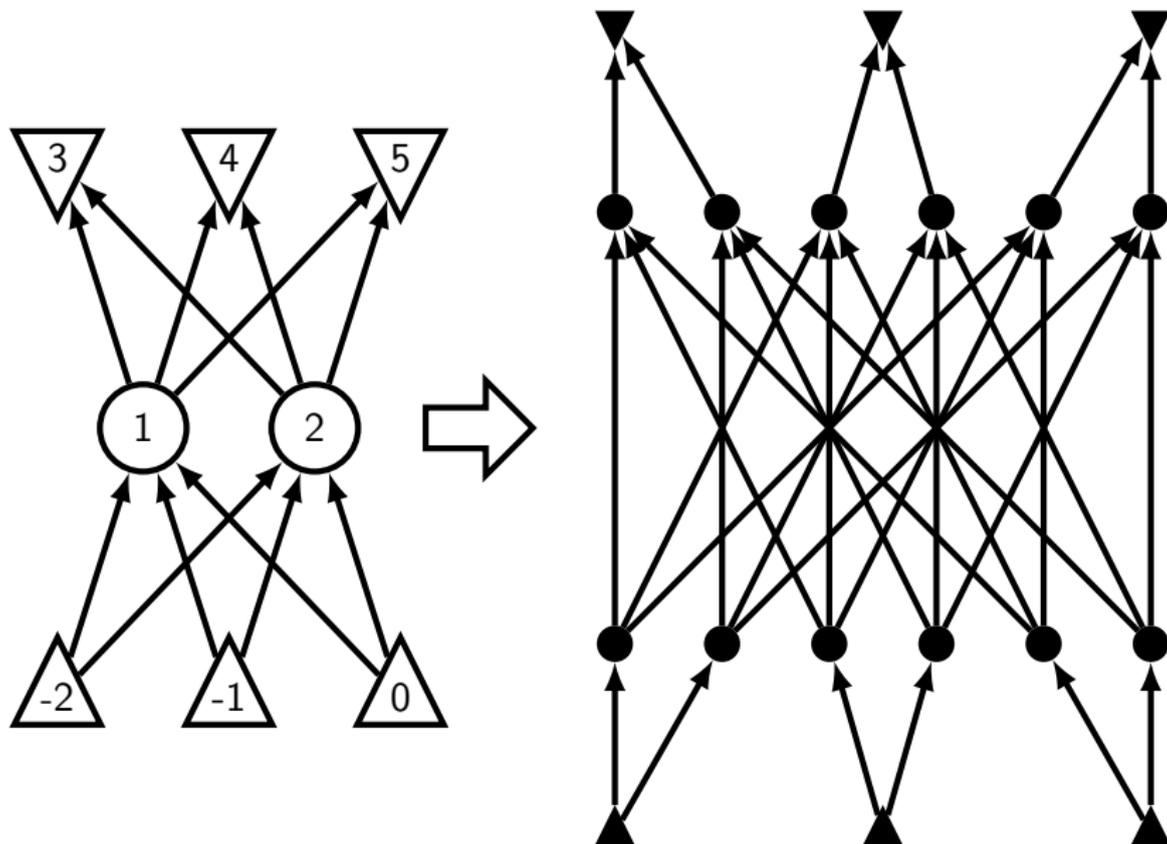


Scarcity \leftrightarrow Optimal Jacobian Accumulation

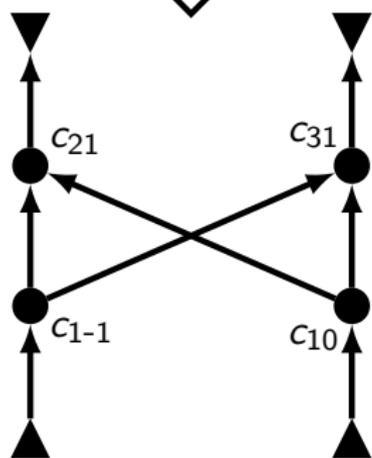
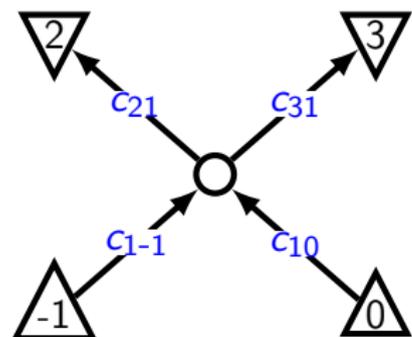


- Combined cost metric: preaccumulation followed by forward vertex elim.
- Results for OJA apply (modulo the fact that p tends to infinity)
- Implication: divisions are useful for OJA (or not? face elim.?)

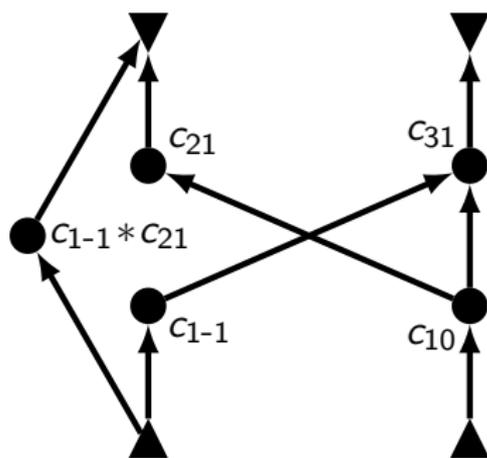
Exploiting Scarcity with Face Elimination



Exploiting Scarcity with Face Elimination



elim. face
(c_{1-1}, c_{21})



Exploiting Scarcity with Face Elimination

