Computing Bottleneck Structures at Scale for High-Precision Network Performance Analysis

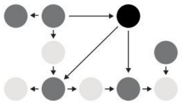
Noah Amsel, Jordi Ros-Giralt, Sruthi Yellamraju,
James Ezick, Brendan von Hofe, Alison Ryan, Richard Lethin

Reservoir Labs

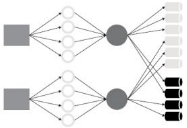
7th Annual International Workshop on Innovating the Network for Data-Intensive Science
November 2020

Reservoir Labs: Technology Expertise

Networking

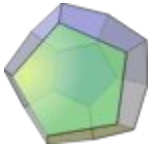


GradientGraph
Network Optimization



R-Core
Packet Path Accelerator

High-Performance Computing

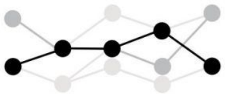


R-Stream
Automatic Parallelization and Mapping Through Polyhedral Model

LLVM
Customization for Advanced Supercomputers

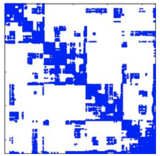
Cybersecurity

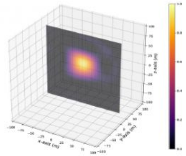
R-Scope
Network Sensor Visibility
Enterprise Security



ENSIGN: Cyber
Spectral Hypergraph Analytics

Algorithms





Asymptotic Improvements to Physical Simulation and Inverse Problems

We show how to scale bottleneck structure analysis to production networks.

Motivating Bottleneck Analysis

The Conventional View of Congestion Control

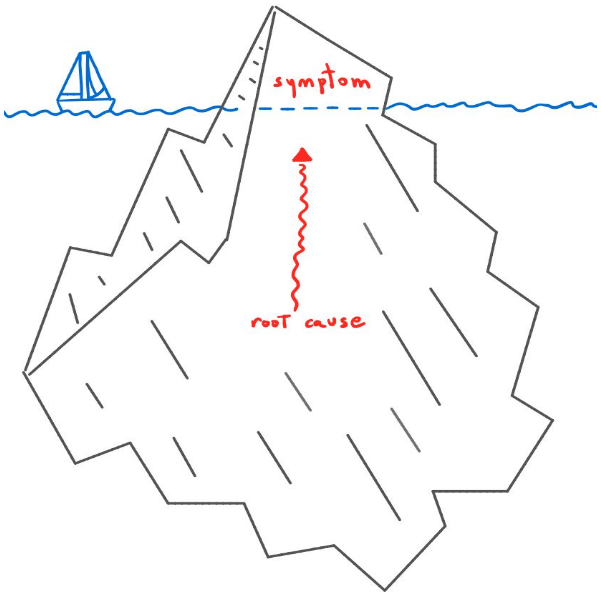
- Consider each flow individually
- Network conditions are a black box

Recent work from Google[1]:

> *“Regardless of how many links a connection traverses or what their individual speeds are, **from TCP's viewpoint an arbitrarily complex path behaves as a single link** with the same RTT [round-trip time] and bottleneck rate.*

[1] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, Van Jacobson, "BBR: Congestion-Based Congestion Control," ACM Queue, Dec 2016.

What are we missing?



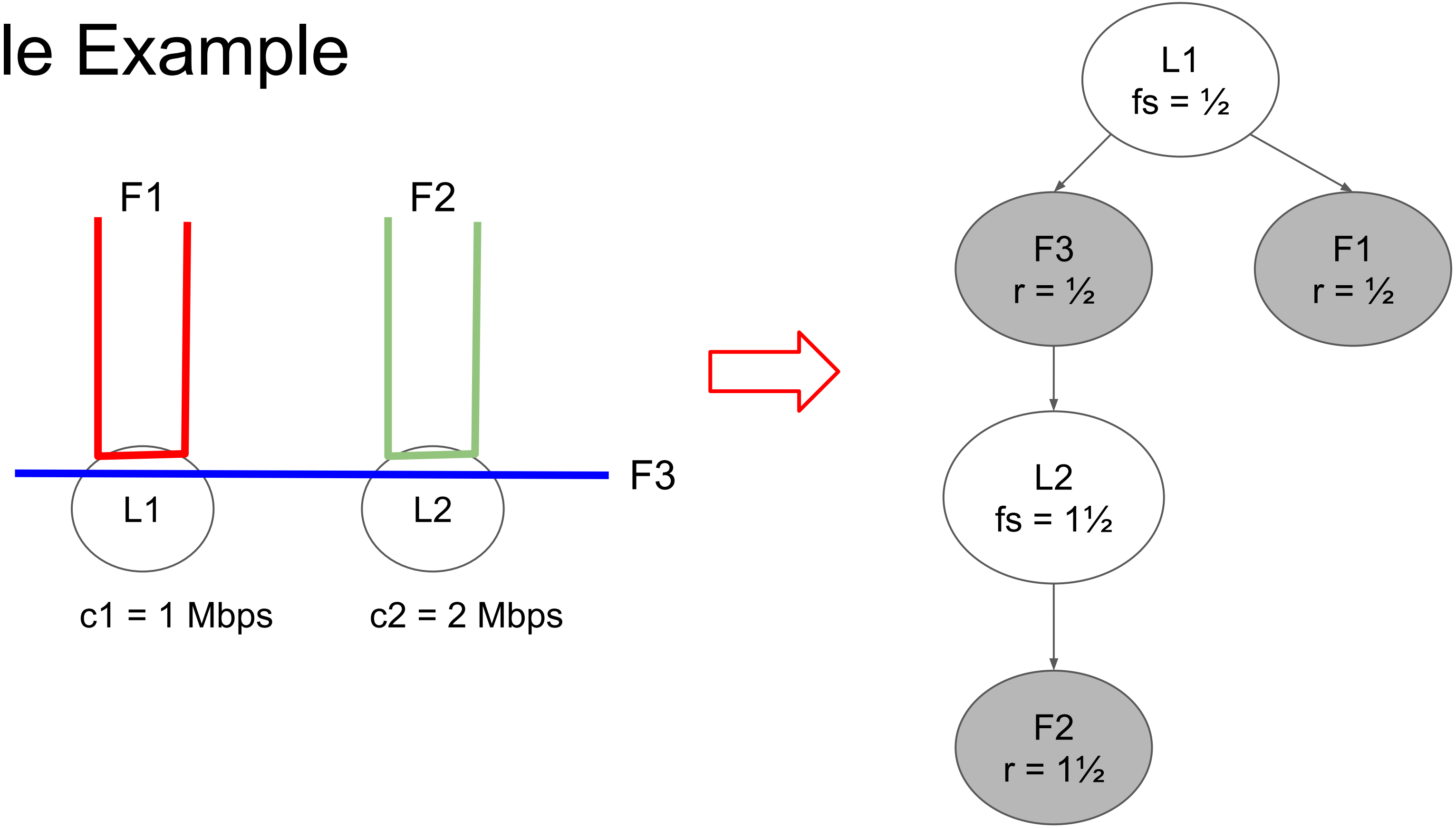
Single-bottleneck view

Hidden interdependency

The Theory of Bottleneck Structures

- Consider all the flows together
- Explain where the network conditions come from
  - “How does each element affect the performance of the other elements?”
- Model this latent dependency structure as a directed graph

Simple Example

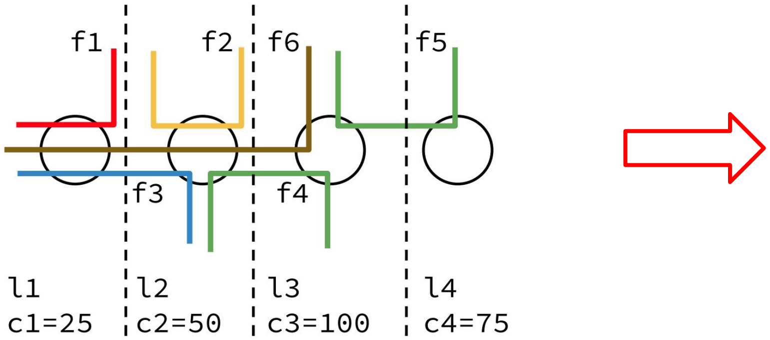


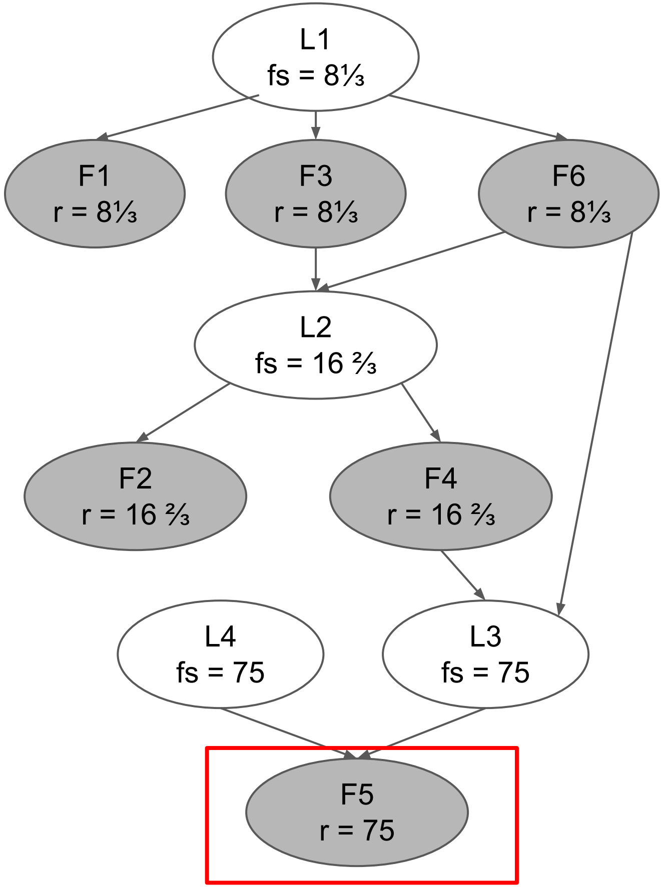
Insight from Bottleneck Structures

- Suppose there are six TCP flows in a network with the following rates (Mbps):

$r_1 = 8\frac{1}{3}$ $r_2 = 16\frac{2}{3}$ $r_3 = 8\frac{1}{3}$
$r_4 = 16\frac{2}{3}$ $r_5 = 75$ $r_6 = 8\frac{1}{3}$

- Which is the elephant flow? Flow 5?

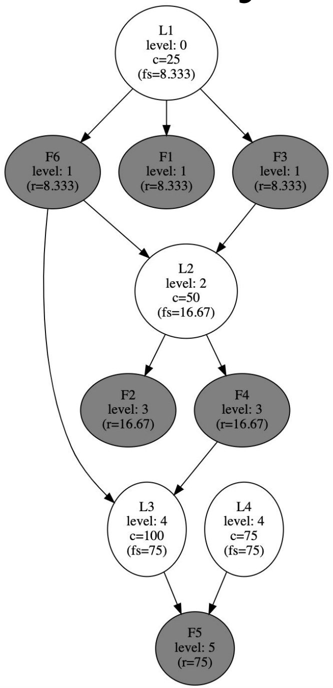




What if we artificially set $r_3 = 8$?

$r_1 = 8\frac{1}{3}$
$r_2 = 16\frac{2}{3}$
$r_3 = 8\frac{1}{3}$
$r_4 = 16\frac{2}{3}$
$r_5 = 75$
$r_6 = 8\frac{1}{3}$

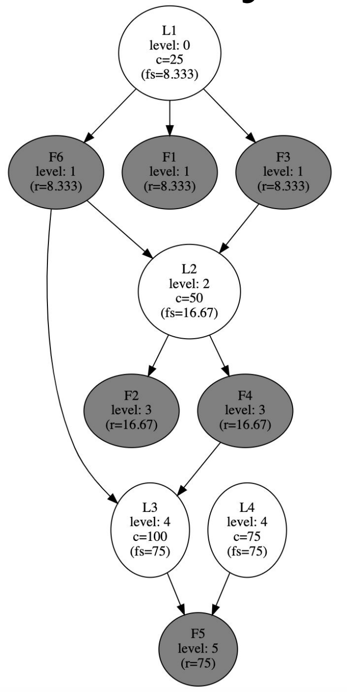
$T = 133\frac{1}{3}$



What if we artificially set $r_3 = 8$?

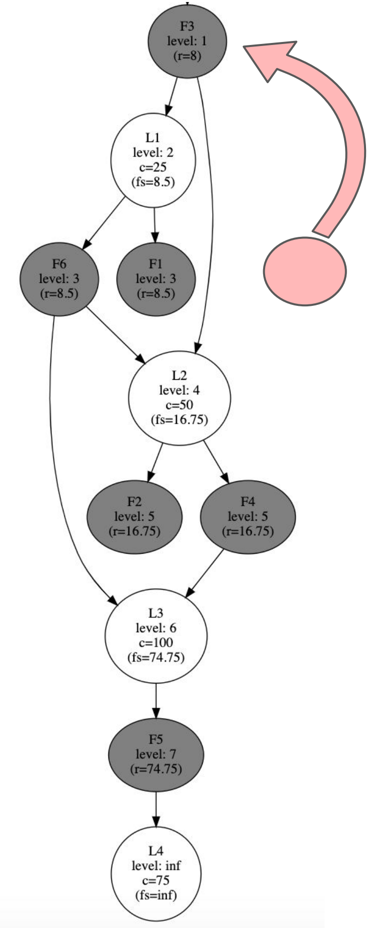
$r_1 = 8\frac{1}{3}$
$r_2 = 16\frac{2}{3}$
$r_3 = 8\frac{1}{3}$
$r_4 = 16\frac{2}{3}$
$r_5 = 75$
$r_6 = 8\frac{1}{3}$

$T = 133\frac{1}{3}$



$r_1 = 8\frac{1}{2}$
$r_2 = 16\frac{3}{4}$
$r_3 = 8$
$r_4 = 16\frac{3}{4}$
$r_5 = 74\frac{3}{4}$
$r_6 = 8\frac{1}{2}$

$T = 133\frac{1}{4}$



Flow Derivatives

$$r_3 = 8\frac{1}{3} \qquad T = 133\frac{1}{3}$$

$$r_3' = 8 \qquad T' = 133\frac{1}{4}$$

$$r_3' = 8\frac{1}{3} - \delta \qquad T' = 133\frac{1}{3} - \delta/4$$

$$\frac{dT}{dr_3} = \lim_{\delta \to 0} \frac{T' - T}{r_3' - r_3} = \frac{-\delta/4}{-\delta} = \frac{1}{4}$$

Flow Derivatives

$$\frac{dT}{dr_1} = 1$$

$$\frac{dT}{dr_4} = 0$$

$$\frac{dT}{dr_2} = 1$$

$$\frac{dT}{dr_5} = 1$$

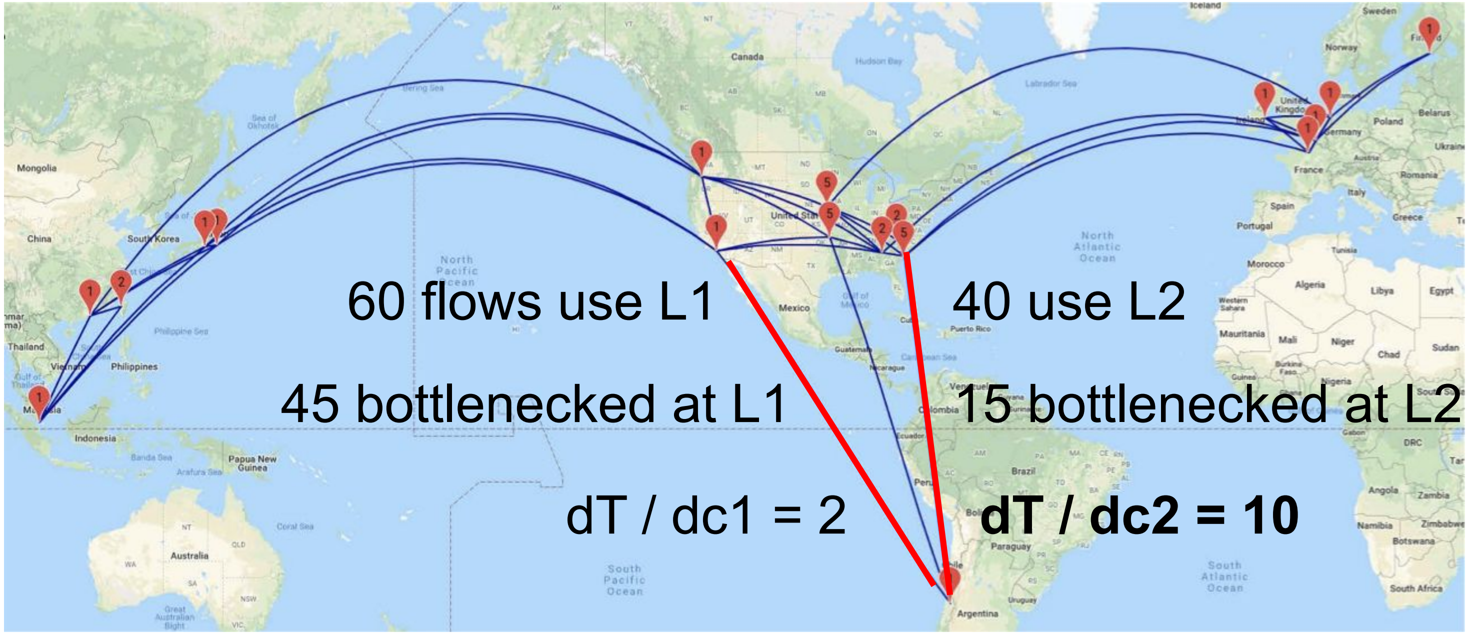
$$\frac{dT}{dr_3} = \frac{1}{4}$$

$$\frac{dT}{dr_6} = -\frac{1}{2}$$

Applications of Bottleneck Structure Analysis

Traffic Engineering	**Flow control optimization**
	Routing optimization
	Multi-path optimization
	Flow admission control
	Bandwidth steering
Network design	**Capacity planning**
	Topology design
	Resiliency analysis
	Robustness analysis
	Bandwidth tapering
Intent-based networking	Performance baselining
	Multi-resource modeling
	Performance troubleshooting
	SLA management

Use Case: Link Upgrades



Chi-yao Hong, et al. “B4 and After: Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in Google's Software-Defined WAN”, SIGCOMM'18 (2018).

Algorithms

Algorithms

	Construct the bottleneck structure graph (and calculate rate assignments)	**Calculate a flow or link derivative**
INDIS 2019	*ComputeBS* ● Modified Water-Filling Algorithm (Bertsekas & Gallager, 1992) ● $O(\|E\| \cdot \|\mathcal{L}\|)$	*BruteGrad* ● Change the rate, then recompute rate assignments from scratch ● Slow, numerical problems

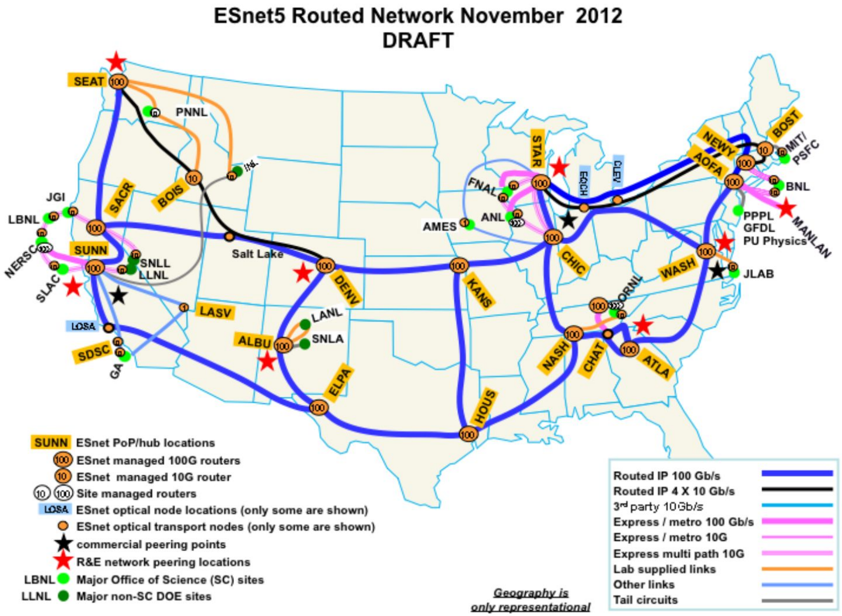
ESnet: A Practical Use Case

High-performance data network that services 50 DOE research sites

As of 2013:

- 28 routers
- 78 links
- > 100K flows

Constantly changing conditions!



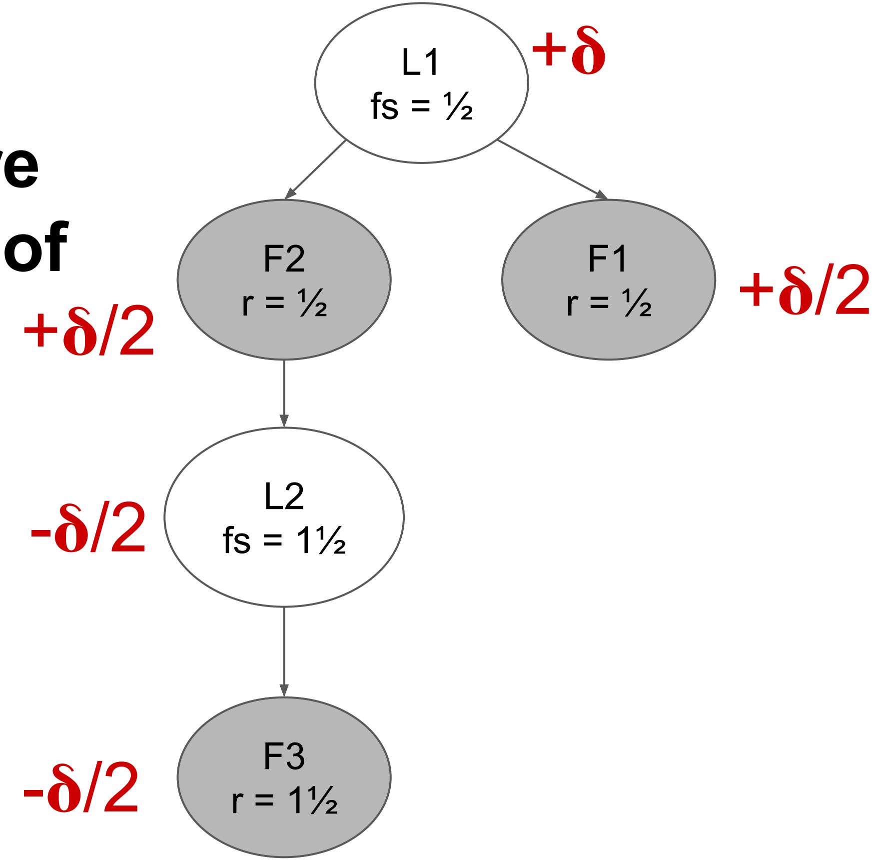
Algorithms

	Construct the bottleneck structure graph (and calculate rate assignments)	Calculate a flow or link derivative
INDIS 2019	*ComputeBS* • Modified Water-Filling Algorithm (CITE SIGMETRICS) • $O(\|E\| \cdot \|\mathcal{L}\|)$	*BruteGrad* • Change the rate, then recompute rate assignments from scratch • Slow, numerical problems
INDIS 2020	*FastComputeBS* • Improved Water-Filling Algorithm using a min-heap • $O(\|E\| \log \|\mathcal{L}\|)$	*ForwardGrad* • Based on forward prop automatic differentiation • Propagate perturbations along the bottleneck structure graph • Runtime is linear in number of affected elements

ForwardGrad

Use the bottleneck structure graph to speed calculation of the derivative

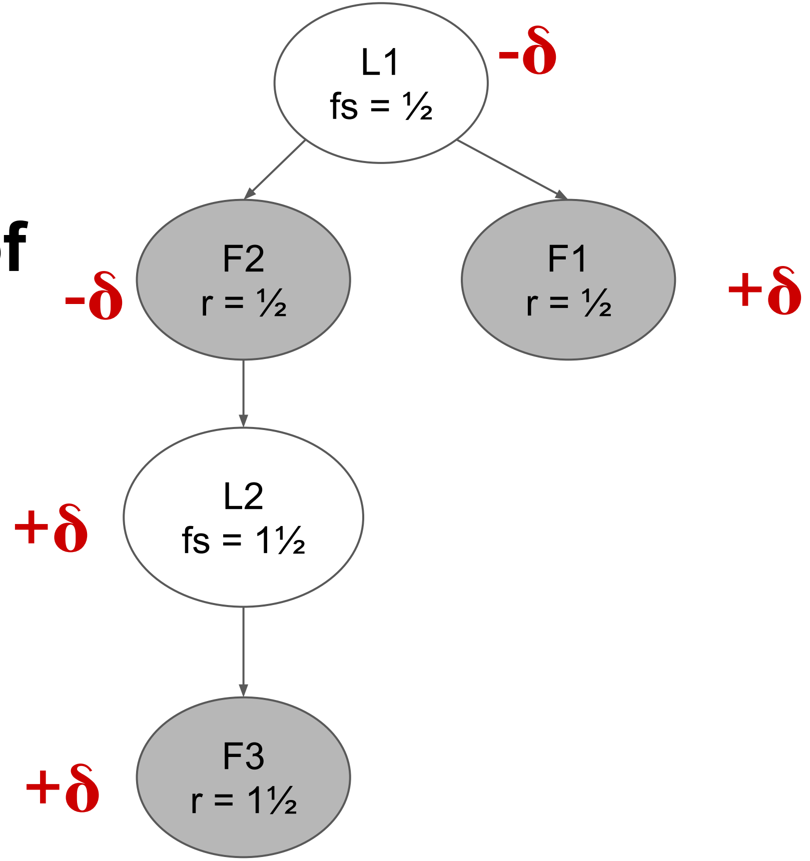
$$\frac{dr_3}{dc_1} = \frac{-\delta/2}{+\delta} = -\frac{1}{2}$$



ForwardGrad

Use the bottleneck structure graph to speed calculation of the derivative

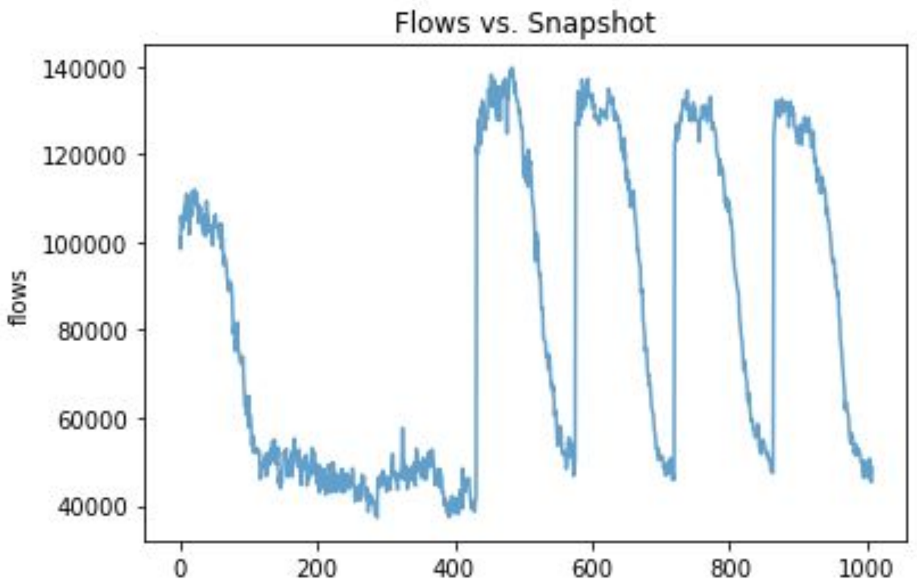
$$\frac{dr_3}{dr_1} = \frac{+\delta}{+\delta} = 1$$



Benchmarking

Benchmarking: Dataset

- NetFlow logs from ESnet
  - 28 routers
- Feb. 1st, 2013 – Feb. 7th, 2013 (Friday – Saturday)
- Sampled every 5 minutes from 8 am – 8 pm
  - 1008 logs per router
  - 28224 logs total
- 78 links



Benchmarking: Software

Tasks:

1. Compute the bottleneck structure graph for each of the 1008 snapshots
2. Compute derivative of total throughput with respect to all 78 link derivatives for each of 12 snapshots

Python Package (INDIS 2019)

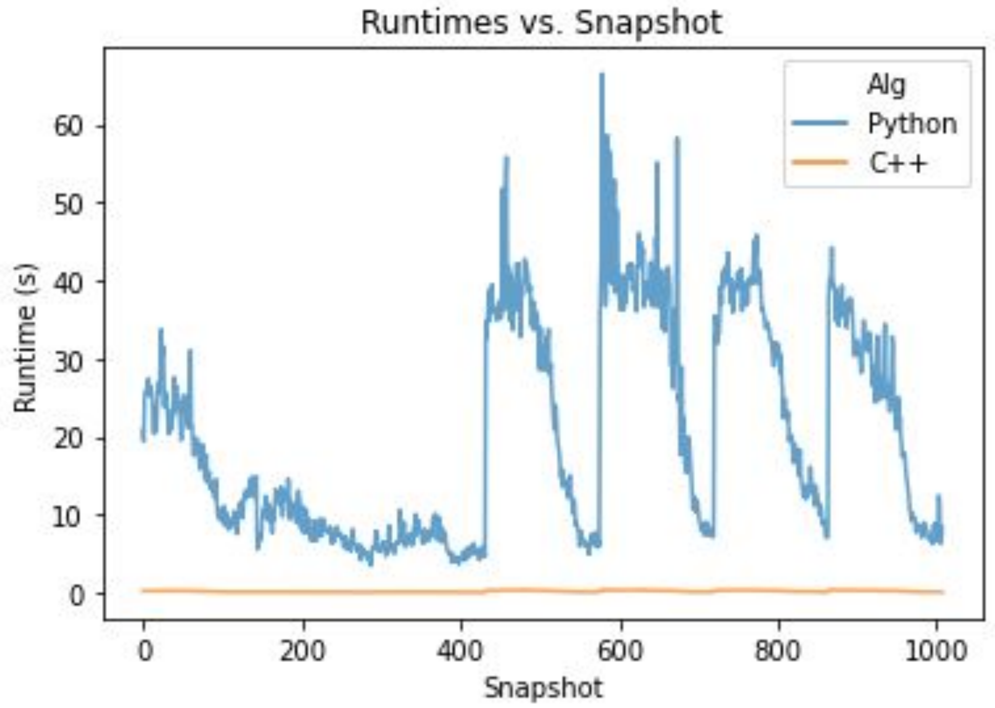
- *ConstructBS*
- *BruteGrad*

C++ Package (INDIS 2020)

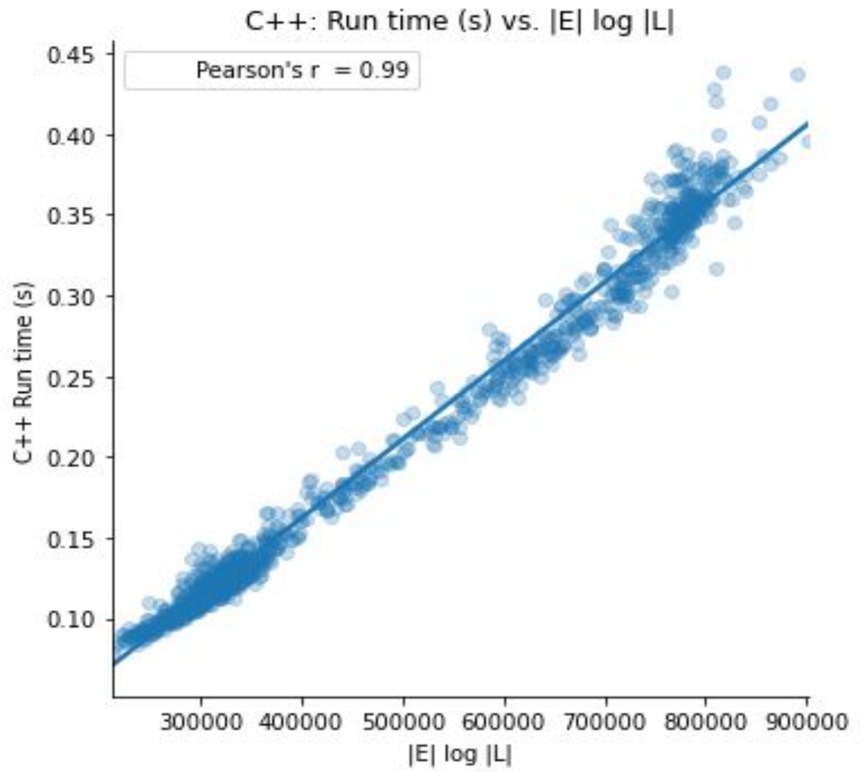
- *FastConstructBS*
- *BruteGrad*
- *ForwardGrad*

Computing Bottleneck Structures: Runtime

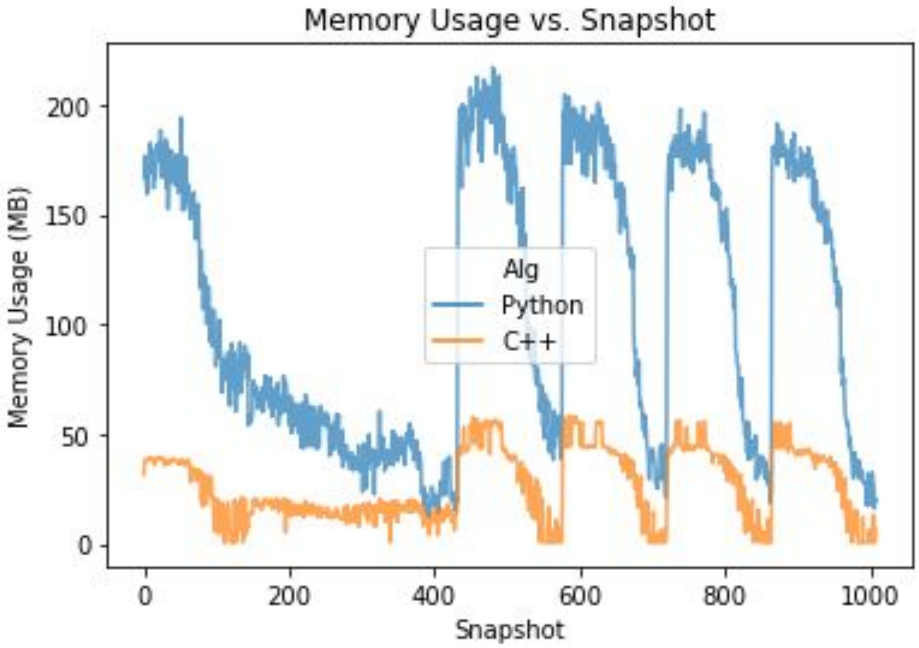
- 87x improvement
- Max: 0.44 s
- Avg: 0.21 s

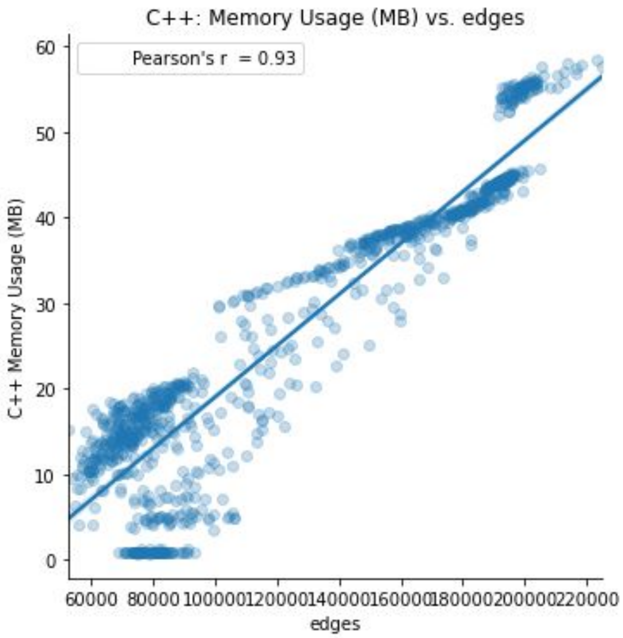


Computing Bottleneck Structures: C++ Runtime

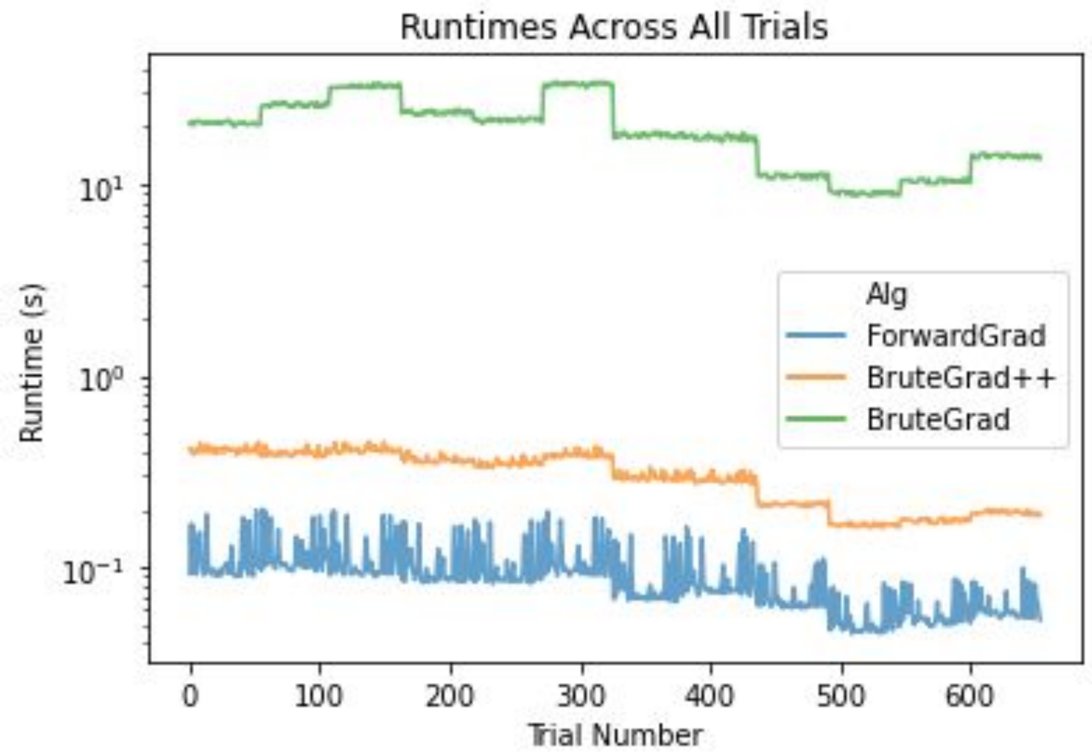


Computing Bottleneck Structures: Memory Usage (4x)





Computing Link Derivatives: Runtime

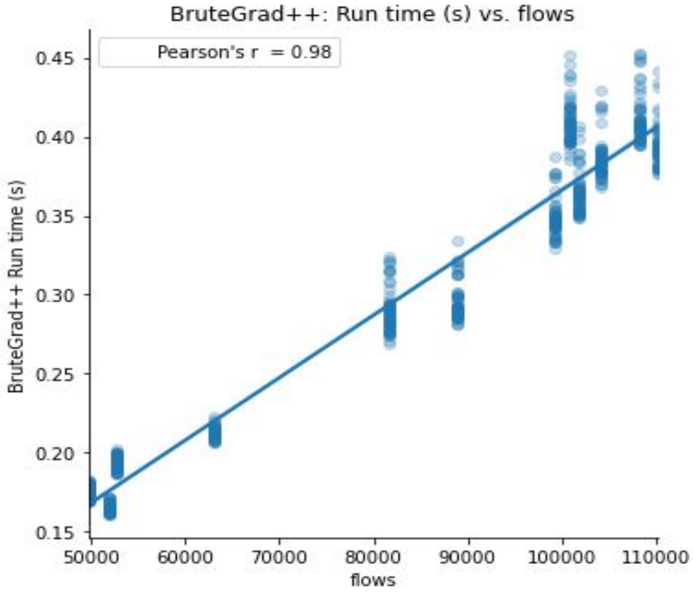


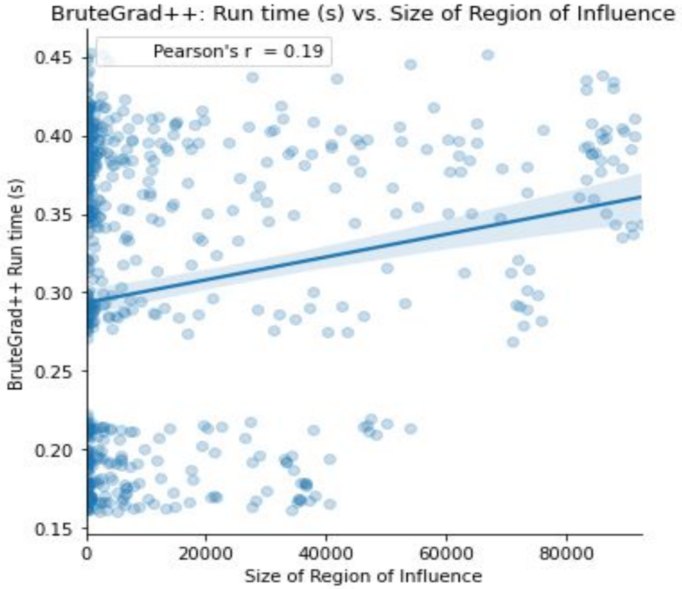
Python vs. C++: 66 x

New algorithm: 3.5 x

ForwardGrad avg: 0.09 s

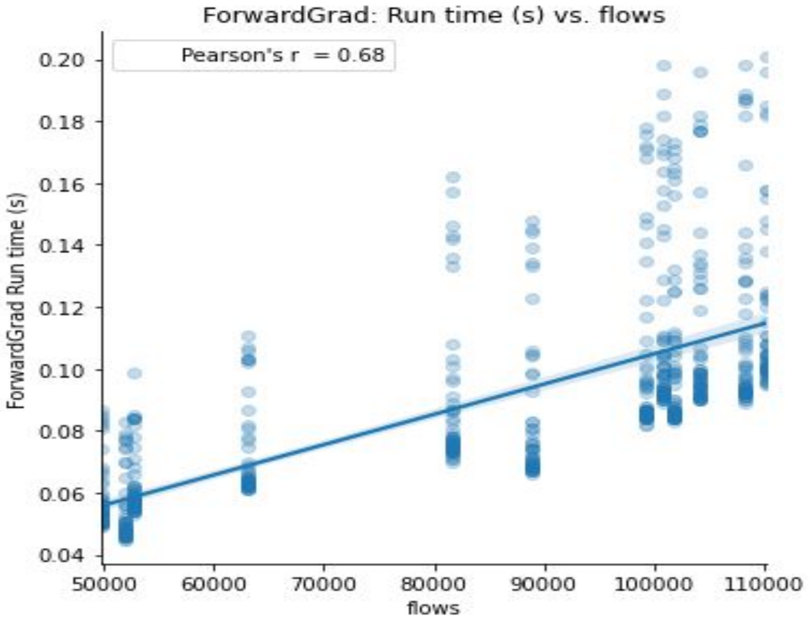
Computing Link Derivatives: BruteGrad++ Runtime

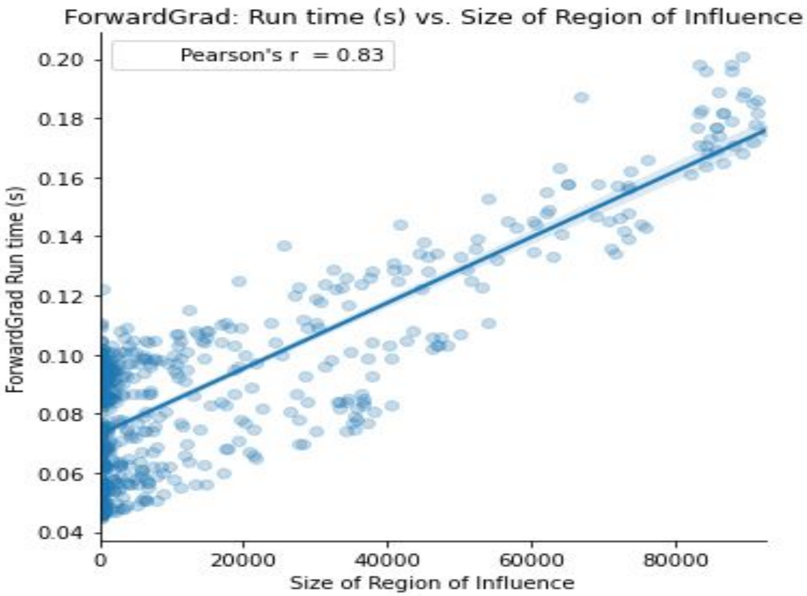




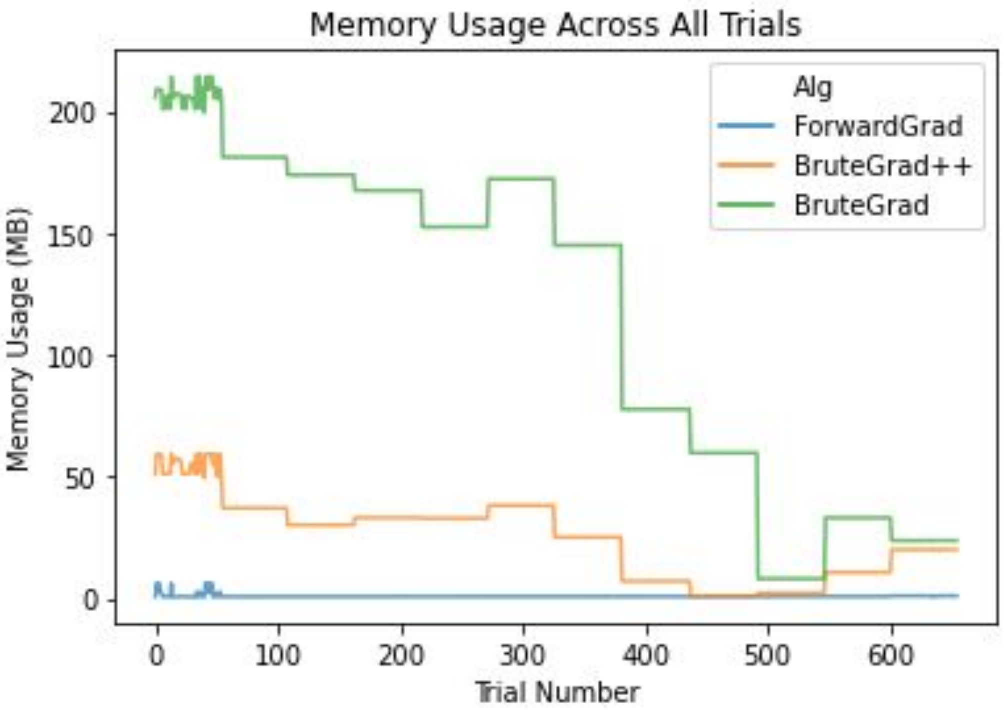
✔

Computing Link Derivatives: ForwardGrad Runtime





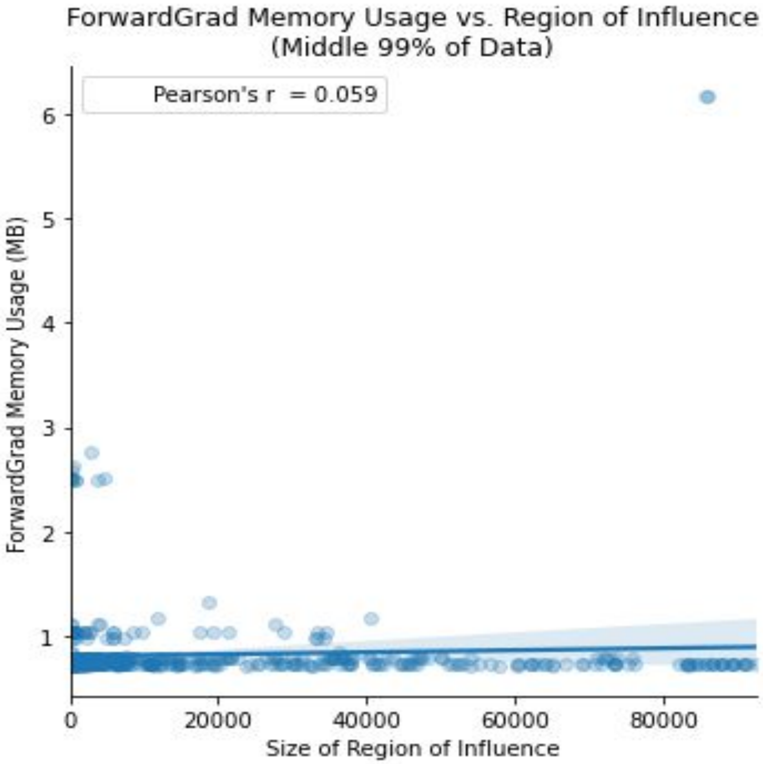
Computing Link Derivatives: ForwardGrad Memory



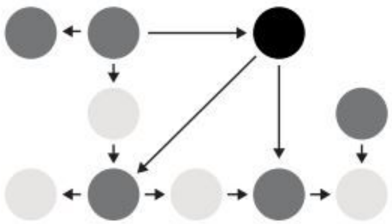
Python vs. C++
10 x

BruteGrad vs. ForwardGrad
30 x

Computing Link Derivatives: ForwardGrad Memory



Our new algorithms and software…



- Are fast enough to analyze changing networks in real time
- Have highly scalable asymptotic complexity curves in time and space
- Unlock myriad potential applications of bottleneck analysis

Thank You!

Reservoir Labs

Contact Us

- Dr. Jordi Ros-Giralt: giralt@reservoir.com
- Noah Amsel (me): amsel@reservoir.com

Related Publications

- Giralt et. al., “On the Bottleneck Structure of Congestion-Controlled Networks,” ACM SIGMETRICS, Boston, June 2020.
- Giralt et. al., “G2: A Network Optimization Framework for High-Precision Analysis of Bottleneck and Flow Performance,” INDIS, Nov. 2019.
- Gudibanda, et al., “Fast Detection of Elephant Flows with Dirichlet-Categorical Inference,” INDIS November, 2018.