Dual Algorithmic Reasoning

ICLR 2023

Danilo Numeroso, Davide Bacciu, Petar Veličković

Stanford CS/MS&E 331

Plan for today

1. **Overview of neural algorithmic reasoning**
2. Ford-Fulkerson refresher
3. Quick paper overview

Neural algorithmic reasoning

Goal: train GNN to imitate classical algorithms
- Typically for polynomial-time solvable problems

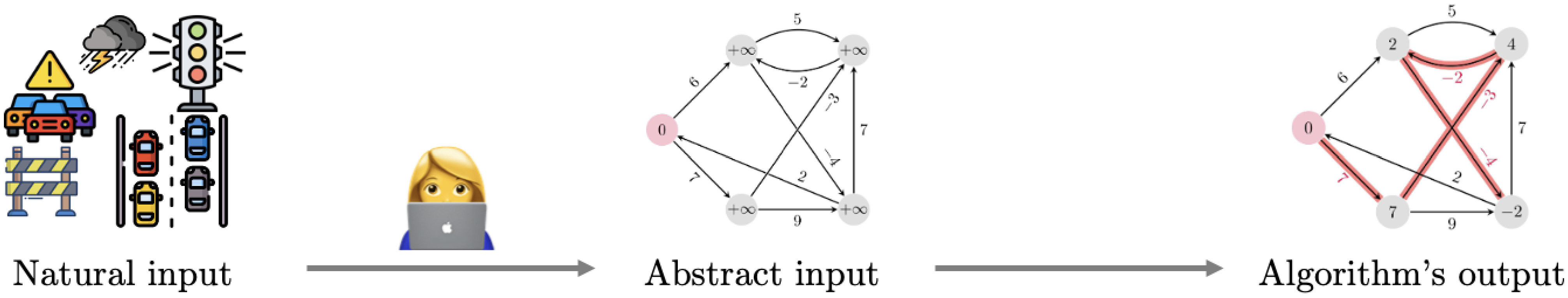
Important question:

If we already have an efficient algorithm for the problem…
why train a GNN?

Classical algorithms are designed with abstraction in mind
- Enforce their inputs to conform to stringent preconditions
- E.g., in routing, that we know traffic patterns perfectly, *a priori*

Neural algorithmic reasoning

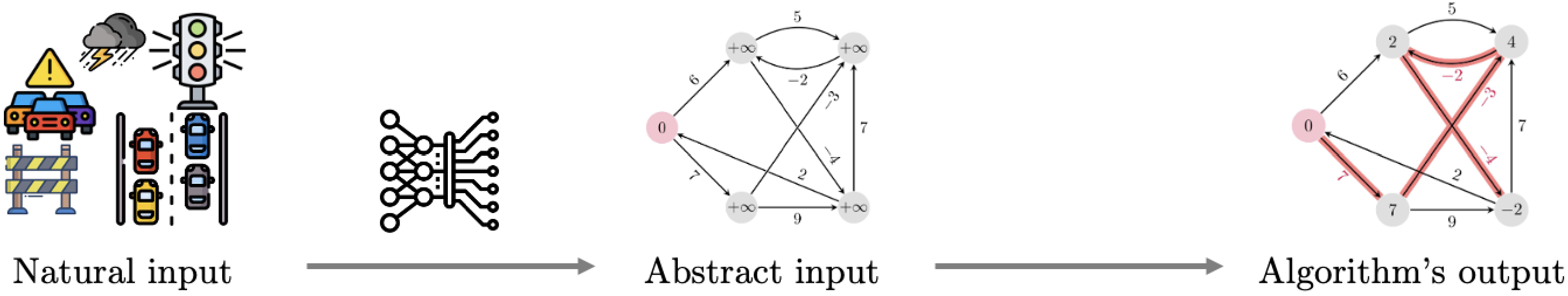
- Assume we have real-world inputs
  ...but algorithm only admits abstract inputs
- First try: Manually convert from one input to another
  - Issue: Not an easy task, so prone to human error



[Stanford CS/MS&E 331](#)

Neural algorithmic reasoning

- Assume we have real-world inputs
  …but algorithm only admits abstract inputs
- Second try: replace human with NN and apply same algorithm
  - Issue: algorithms typically perform discrete optimization
    - Doesn't play nicely w/ gradient-based optimization of NNs



Neural algorithmic reasoning

- Second (more fundamental) issue: **data efficiency**
  - Real-world data is often **incredibly rich**
  - We still have to **compress** it down to scalar values
  - Algorithm commits to using this scalar, assuming it's perfect
- **Goal of neural algorithmic reasoning:**
  Seamless, differentiable pipeline: natural inputs → outputs
- Use existing algorithm:
  - Guide selection of learnable modules
  - Intermediate supervision (end-to-end learning rarely works)

Dual algorithmic reasoning

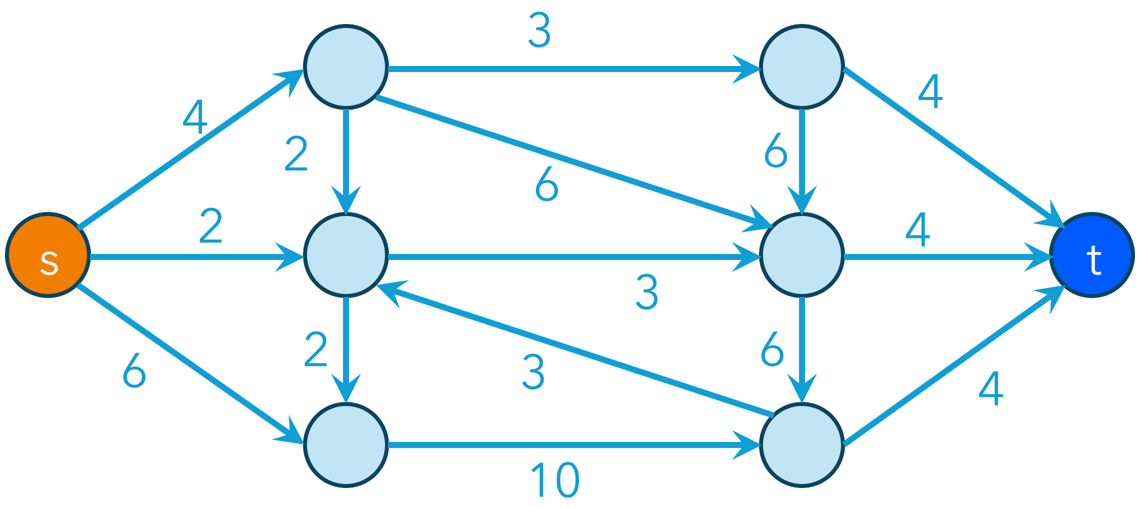
- Prior work: Multi-task learning on similar algorithms helps
  - Joint training improves learning & transfer across related algorithms
  - Many algorithms reuse primitives like Bellman–Ford and BFS
- **Key idea: use duality information**
  - Many problems admit primal and dual formulations
  - Solving one often reveals the solution to the other
  - Train on primal and dual optimization simultaneously
- **Main example:** max-flow, min-cut
- Results: gains on synthetic algorithmic and real graph tasks

Plan for today

1. Overview of neural algorithmic reasoning
2. **Ford-Fulkerson refresher**
3. Quick paper overview

Min cut

- Graphs are directed and edges have “capacities” (weights)
- We have a special “source” vertex s and “sink” vertex t
  - s has only outgoing edges
  - t has only incoming edges

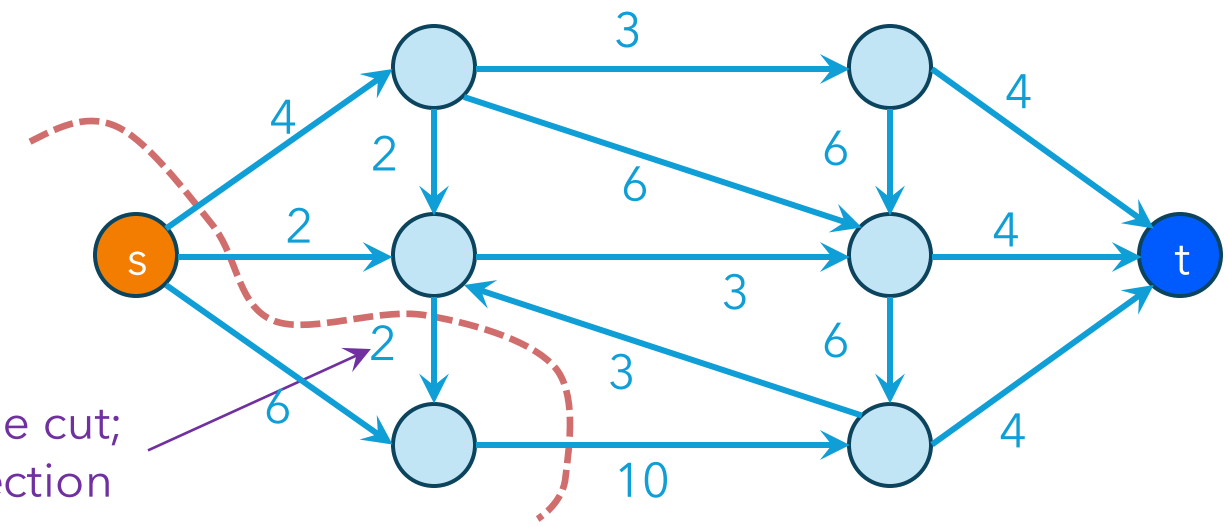


[Stanford CS 161]

Min cut

- An **s-t cut** is a cut which separates s from t
- An edge **crosses the cut** if it goes from s's side to t's side

This cut has cost 4 + 2 + 10 = 16

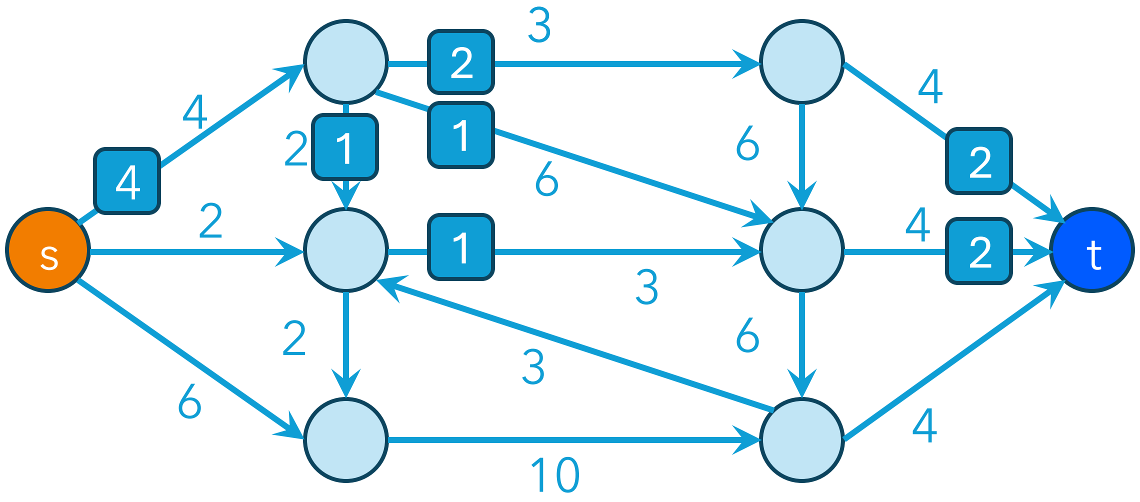


Edge does not cross the cut;
it's going in wrong direction

Stanford CS 161

Max flow

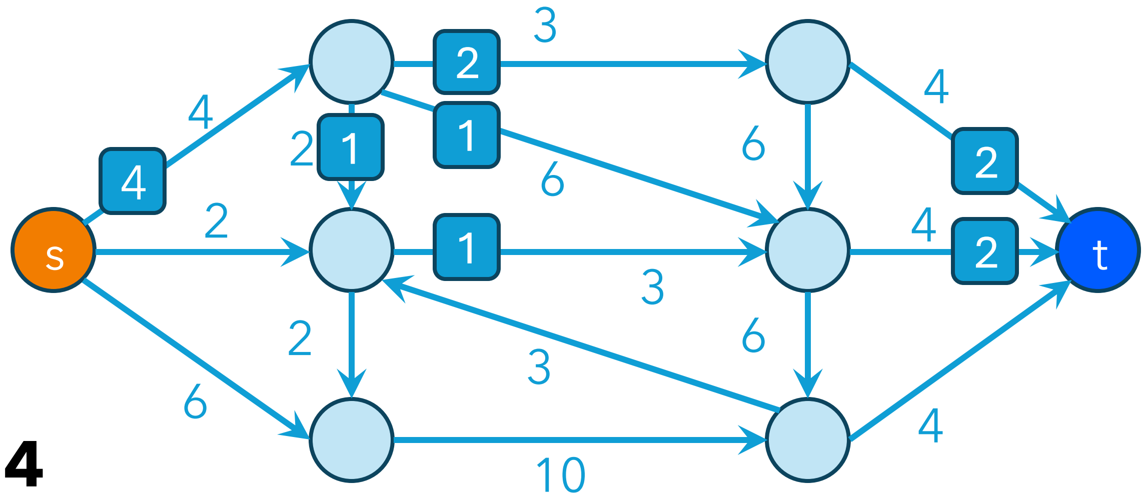
- In addition to a capacity, each edge has a flow
  - Unmarked edges in the picture below have flow 0
- Flow on an edge must be less than its capacity
- At each vertex (other than s,t) incoming flow = outgoing flow



Stanford CS 161

Max flow

- The value of a flow is:
  - The amount of flow going out of s
  - Which is equal to the amount of flow going into t

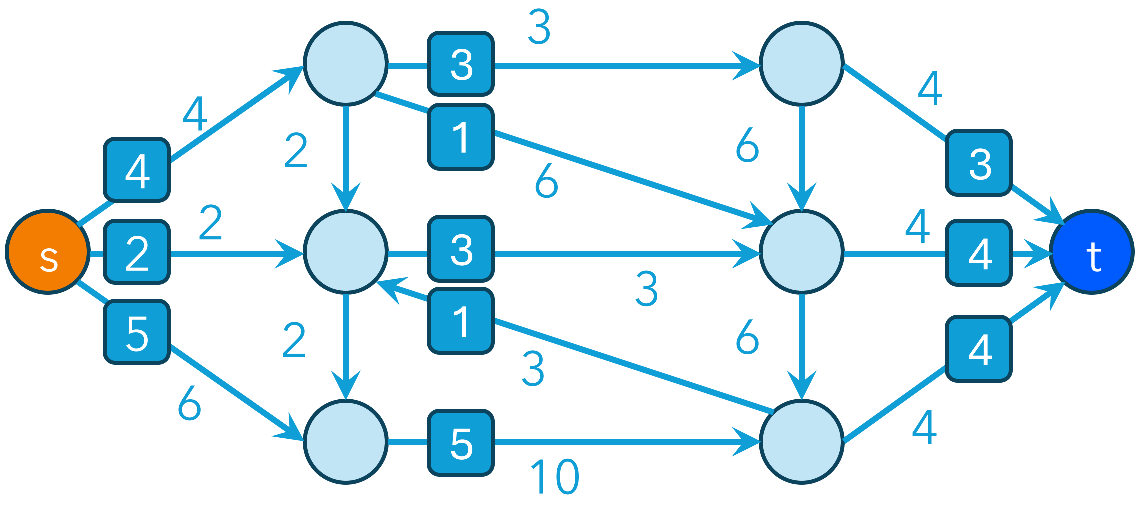


Value of this flow is 4

Stanford CS 161

Max flow

- The value of a flow is:
  - The amount of flow going out of s
  - Which is equal to the amount of flow going into t

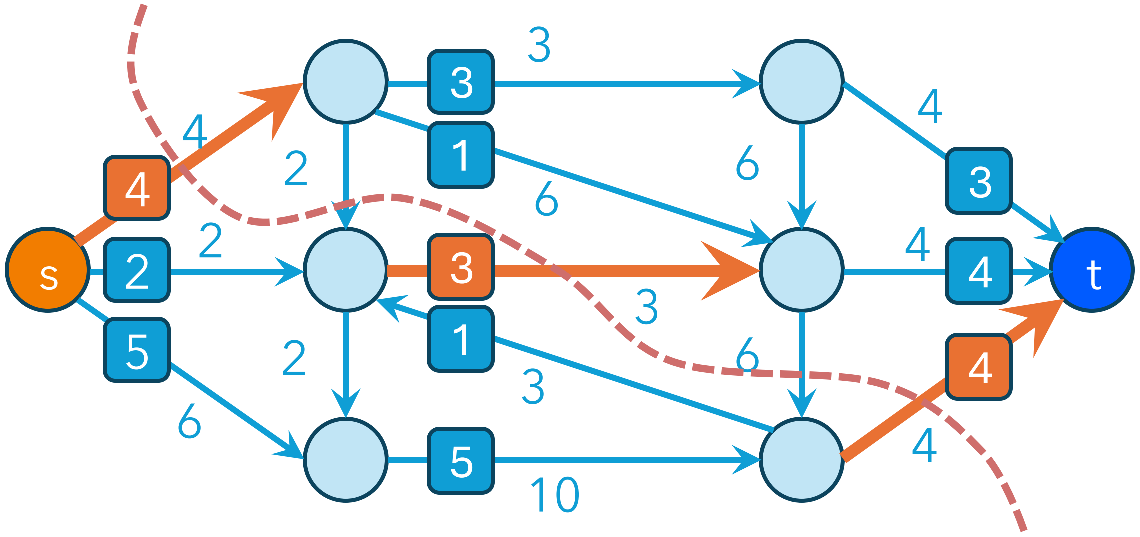


Max flow is 11

Max-flow min-cut theorem

Value of a max flow from s to t = cost of a min s-t cut

Intuition: in max flow, min cut better fill up; this is the bottleneck



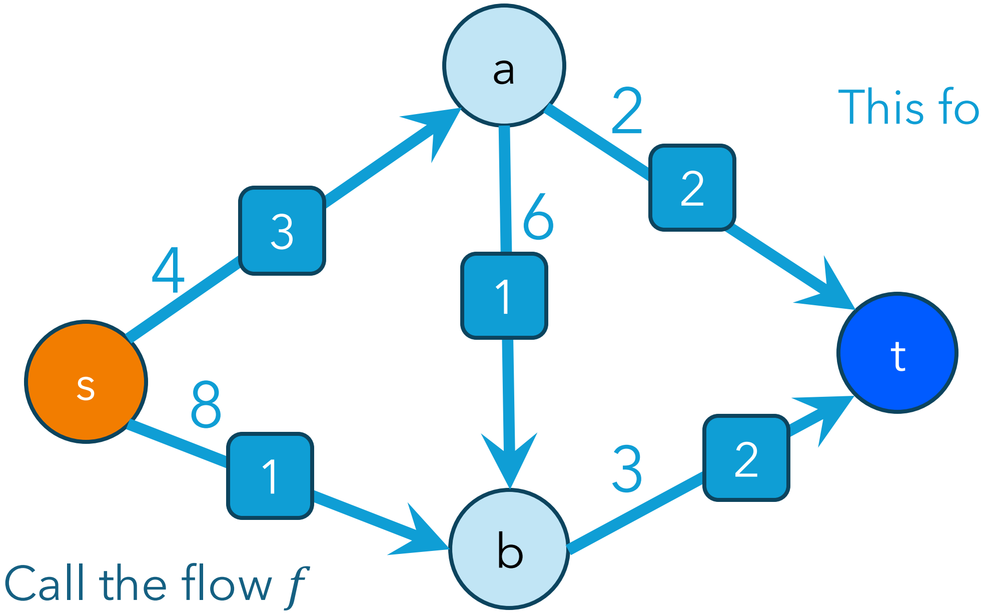
Stanford CS 161

Ford-Fulkerson algorithm

Outline of algorithm:

- Start with zero flow
- We will maintain a “residual graph” G_f
- Path from s to t in G_f will give us a way to improve our flow
- Continue until there are no s-t paths left

Tool: Residual networks

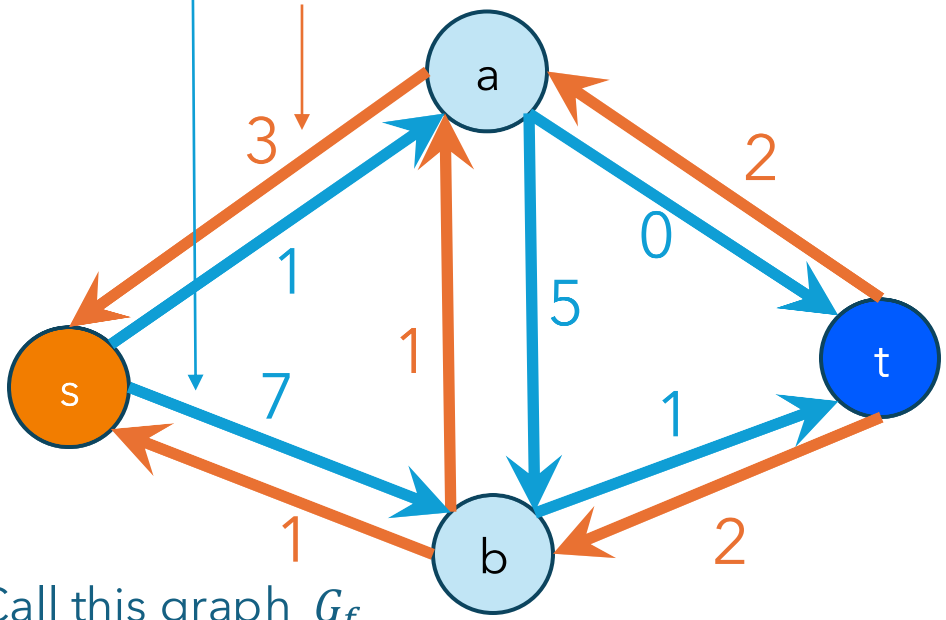


Call the flow f
Call the graph G

Create a new **residual network** from this flow:

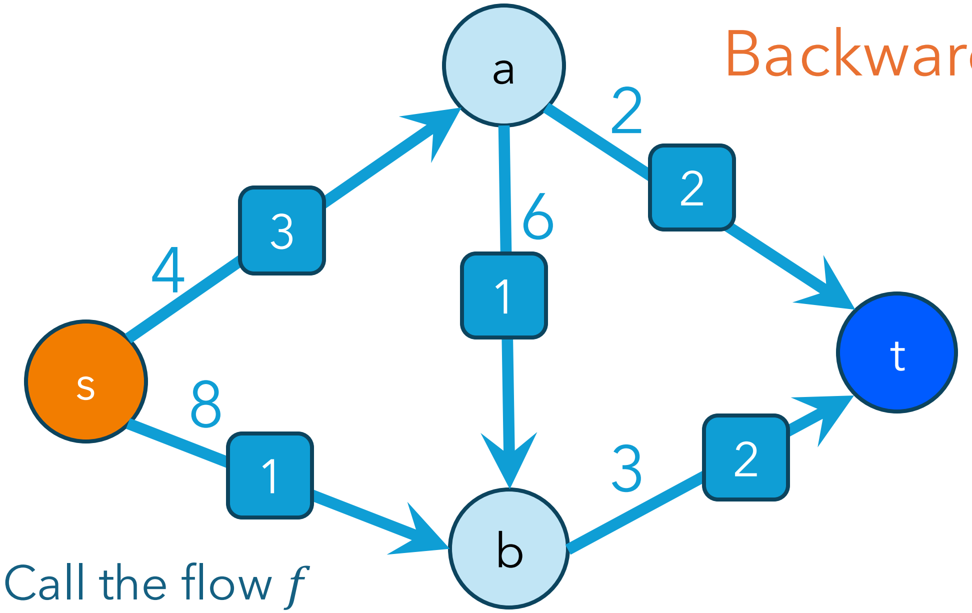
This forward edge has weight "capacity – flow"

This backward edge has weight "flow"



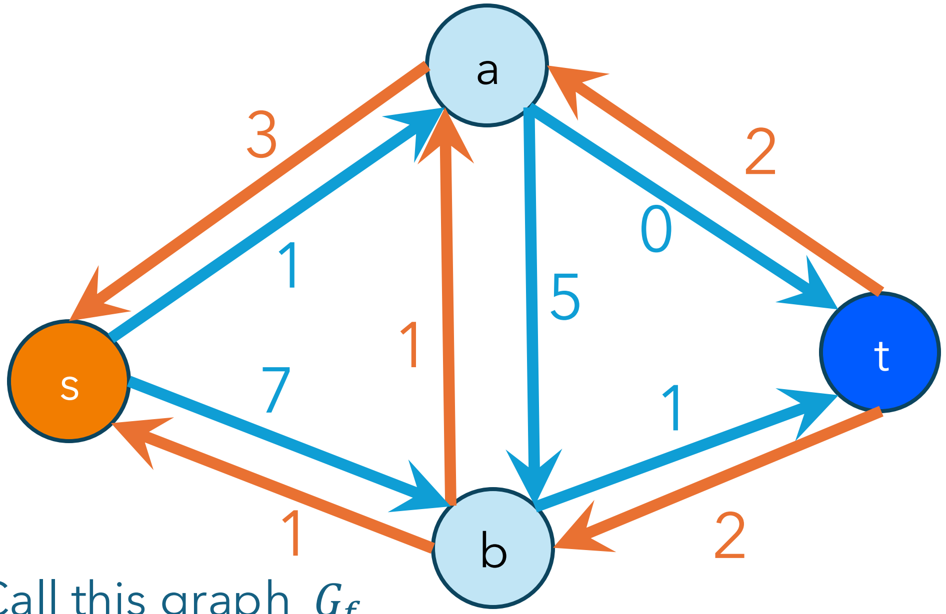
Call this graph G_f

Tool: Residual networks



Call the flow f
Call the graph G

Backwards edges are the amount that's been used
Forward edges are the amount that's left

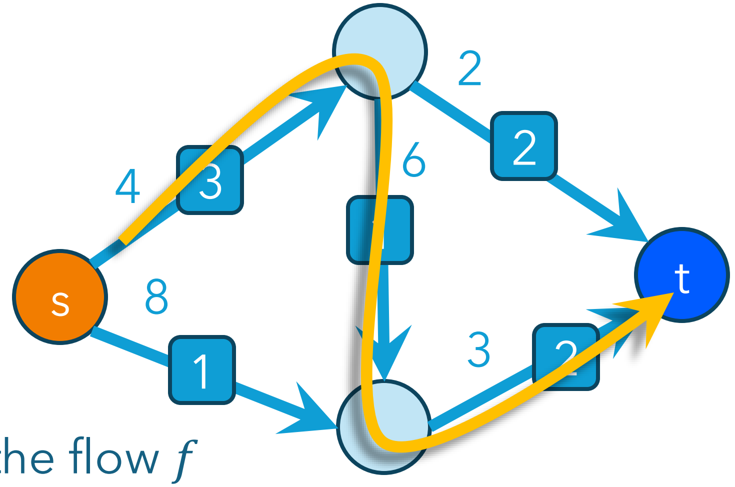


Call this graph G_f

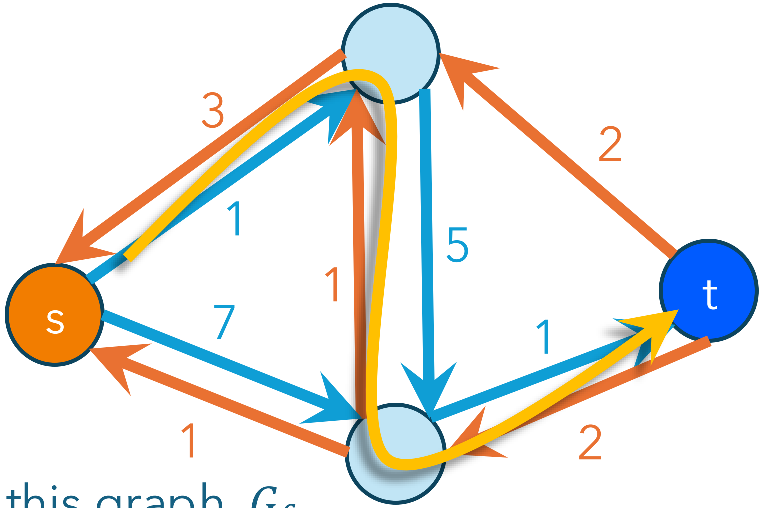
Create a new **residual network** from this flow:

Tool: Augmenting paths

- Path s → t in residual network is called an *augmenting path*
- If there's an augmenting path, can increase flow along path



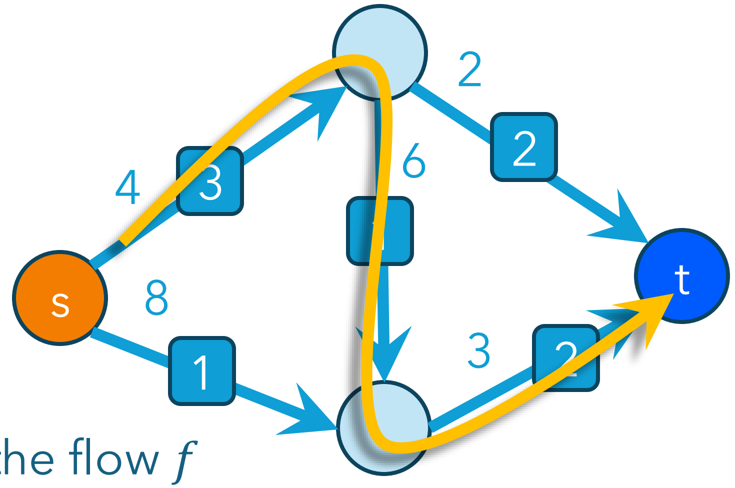
Call the flow f
Call the graph G



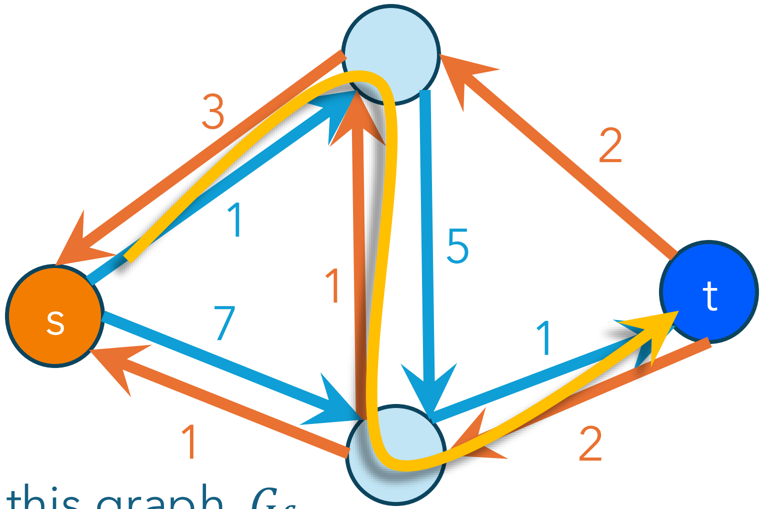
Call this graph G_f

Tool: Augmenting paths

- Easy case: every edge on the path in G_f is a **forward edge**
  - Just increase the flow on all the edges!



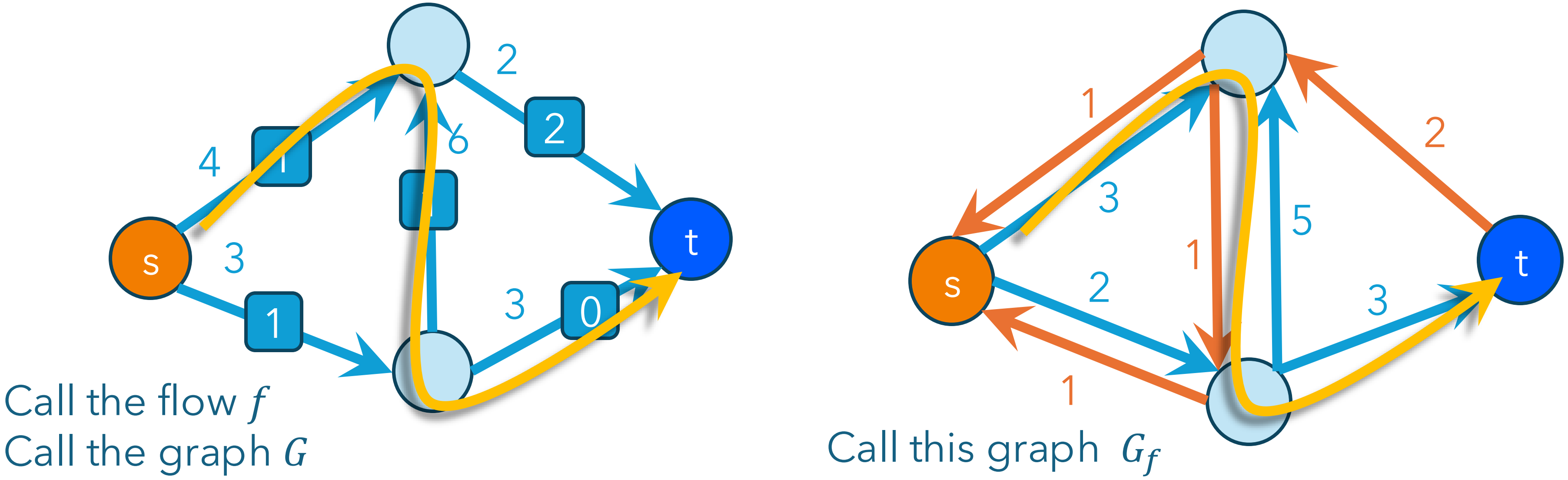
Call the flow f
Call the graph G



Call this graph G_f

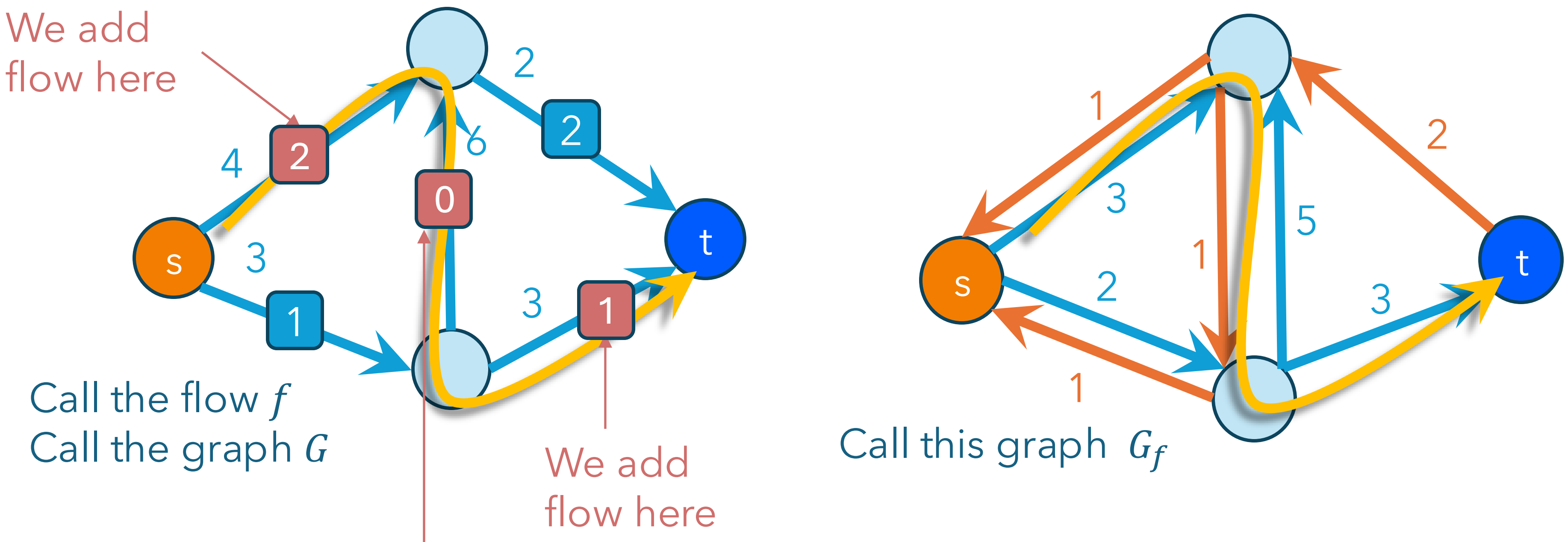
Tool: Augmenting paths

- Harder case: there are **backward edges** in the path
  - Here's a slightly different example of a flow:



Tool: Augmenting paths

- Harder case: there are **backward edges** in the path
  - Here's a slightly different example of a flow:



We remove flow here, since augmenting path is going backwards along this edge

Ford-Fulkerson Algorithm

1. f ← all zero flow
2. $G_f \leftarrow G$
3. while t is reachable from s in G_f
   1. Find a path P from s to t in G_f // e.g., use DFS or BFS
   2. f ← increaseFlow(P, f)
   3. update G_f
4. return f

Correctness follows from max-flow min-cut theorem

E.g., see lecture notes on course webpage

Plan for today

1. Overview of neural algorithmic reasoning
2. Ford-Fulkerson refresher
3. **Quick paper overview**

Dual algorithmic reasoning (DAR)

Encode–Process–Decode neural execution [Veličković, Blundell '21]

1. **Encoding network:** Node/edge features → latent space
2. **Processor networks:** Learn Ford–Fulkerson w/ 2 processors
    - **Processor 1:** Learns to find augmenting paths
    - **Processor 2:** Performs flow updates and predicts min s–t cut
3. **Decoding network:** Convert latent states to path, flow, cut

Training with hints:

- Supervise each intermediate state (augmenting paths, flows)
- Provides step-wise signals to reduce error propagation

Real-world experiments

- **Goal:** Test if DAR transfers to real-world data
- Apply pretrained DAR models to **brain vessel graphs**
  - Task: classify vessel types
- **Method:** Reuse synthetic-trained processor networks
  - Retrain encoders on physical features
- Learned flow dynamics act as meaningful graph embeddings
  - Dual DAR embeddings outperform baselines
- **Take-away:**

  Dual reasoning yields richer, flow-aware representations