Combinatorial optimization and reasoning with GNNs: **Algorithmic reasoning**

Cappart, Chételat, Khalil, Lodi, Morris, Veličković

Guide to reading

Overview of current research in **neural algorithm reasoning**

Don’t worry about understanding each and every direction 😊

- The goal should be to get a broad sense of the field

Neural-algorithmic alignment

Many ways to attack combinatorial optimization with DL:

Why GNNS?

GNNs can execute poly-time **dynamic programming** algs

E.g. “Graph Neural Networks are Dynamic Programmers,” Dudzik and Veličković, NeurIPS’22

Paradigm from which many poly-time algs can be constructed

Extrapolation

(G)NNs are traditionally powerful at ***interpolation***

- i.e., strong performance when test distribution ≈ training distribution

(G)NNs typically struggle at ***extrapolation***

- i.e., evaluated out of distribution
- E.g., increasing number of nodes in input graph

Neural algorithmic alignment:

How to construct algorithmic reasoners that extrapolate?

Extrapolation

Key aspects of Veličković et al. [ICLR'20] enabling extrapolation:

1. Using the **encode-process-decode** framework
2. Favoring the **max-aggregation** function
3. Supervising with ground-truth alg's **execution traces**
4. Executing **multiple** related algorithms

Executing multiple related algorithms

The CLRS Algorithmic Reasoning Benchmark

Petar Veličković [1] Adrià Puigdomènech Badia [1] David Budden [1]
Razvan Pascanu [1] Andrea Banino [1] Misha Dashevskiy [1] Raia Hadsell [1] Charles Blundell [1]

Includes:

- Sorting
- Searching
- Dynamic programming
- Graph algorithms
- String algorithms
- Geometric algorithms

Outline

1. Introduction
2. **Understanding max-aggregation**
3. Reasoning on natural inputs

Extrapolation error

[Xu et al., ICLR'21]

- $f: \mathcal{X} \to \mathbb{R}$ is a model trained on $\{(x_i, y_i)\}_{i=1}^{n} \subset \mathcal{D}$
  $y_i = g(x_i)$ for some ground-truth function g
- $\mathcal{P}$ is a **distribution over $\mathcal{X} \setminus \mathcal{D}$**
- $\ell: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a loss function
- **Extrapolation error:** $\mathbb{E}_{x \sim \mathcal{P}}\left[\ell\left(f(x), g(x)\right)\right]$

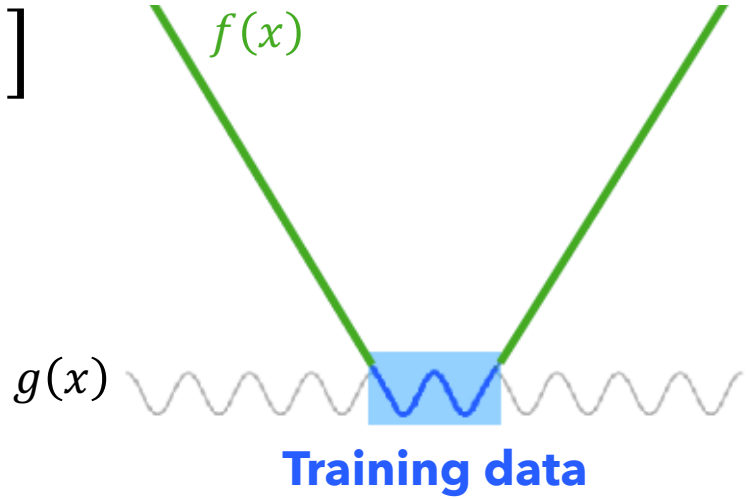
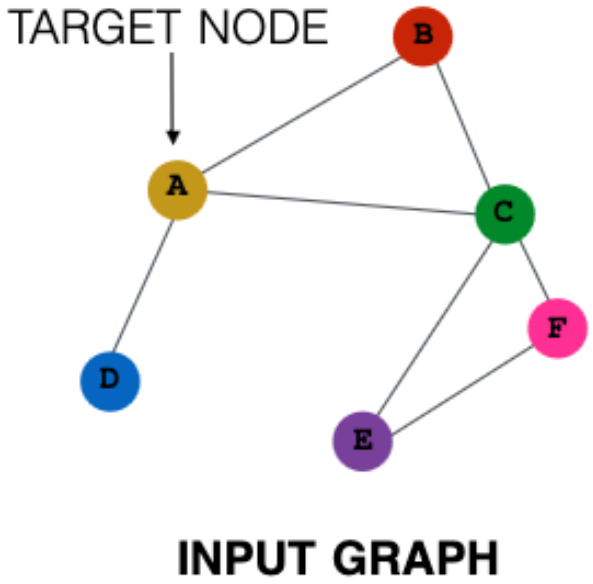


Figure by Xu et al., ICLR'21

Aggregation functions



$$\boldsymbol{h}_A^{(t)} = \sum_{j \in N(A)} \mathrm{MLP}^{(t)}\left(\boldsymbol{h}_A^{(t-1)}, \boldsymbol{h}_j^{(t-1)}\right)$$

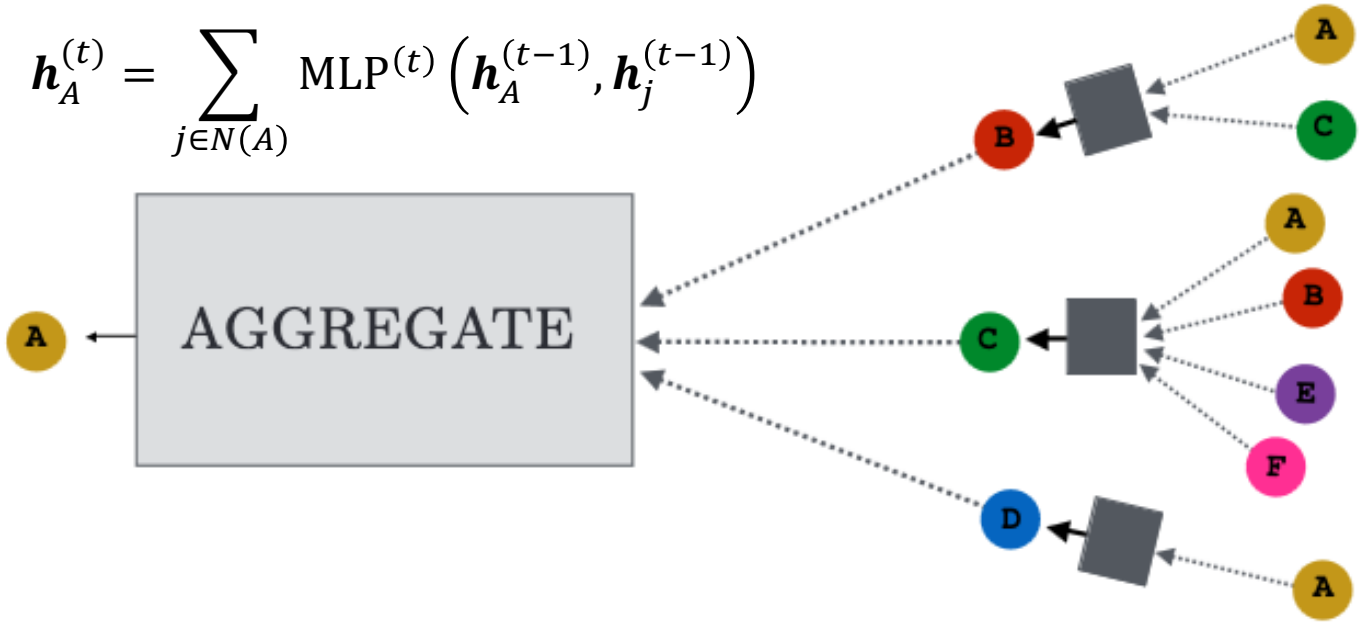
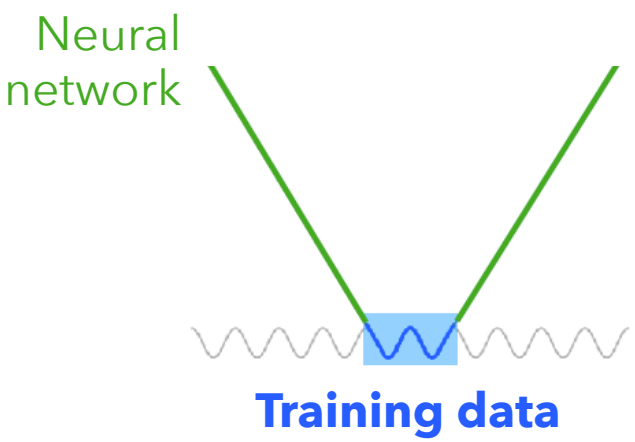


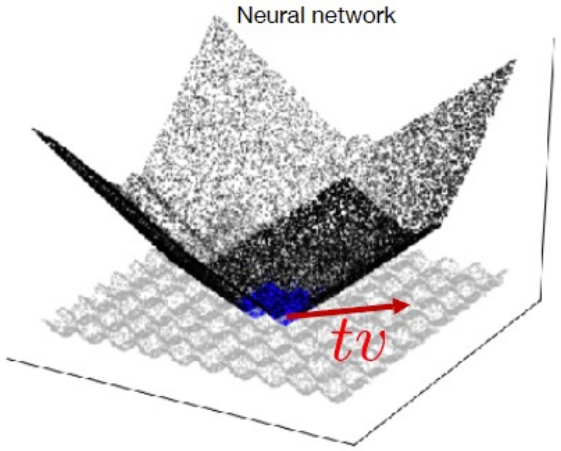
Figure by Hamilton

ReLU MLP extrapolate linearly

Theorem [Xu et al., ICLR'21, informal]:

- Let f be a 2-layer ReLU MLP trained with gradient descent
- Along any direction $\boldsymbol{v} \in \mathbb{R}^d$, f approaches a **linear** function
- I.e., let $\boldsymbol{x} = t\boldsymbol{v}$. Then $f(\boldsymbol{x} + h\boldsymbol{v}) - f(\boldsymbol{x}) \to \beta_{\boldsymbol{v}} h$ at a rate $O\left(\frac{1}{t}\right)$





Figures by Xu et al., ICLR'21

Implications for GNNs

Shortest path: $x_i^{(t+1)} = \min\left\{x_i^{(t+1)}, \min_{(j,i)\in E} x_j^{(t)} + e_{ji}^{(t)}\right\}$

GNN: $\boldsymbol{h}_i^{(t+1)} = \sum_{j\in N(i)} \mathrm{MLP}^{(t)}\left(\boldsymbol{h}_i^{(t-1)}, \boldsymbol{h}_j^{(t-1)}\right)$

MLP must learn a **non-linearity**

Implications for GNNs

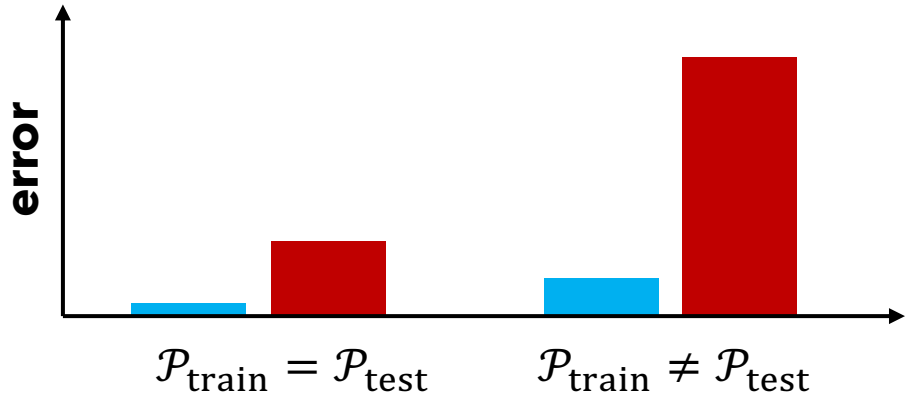
Shortest path: $x_i^{(t+1)} = \min\left\{x_i^{(t+1)}, \min_{(j,i)\in E} x_j^{(t)} + e_{ji}^{(t)}\right\}$

GNN: $\boldsymbol{h}_i^{(t+1)} = \sum_{j\in N(i)} \mathrm{MLP}^{(t)}\left(\boldsymbol{h}_i^{(t-1)}, \boldsymbol{h}_j^{(t-1)}\right)$

GNN 2: $\boldsymbol{h}_i^{(t+1)} = \max_{j\in N(i)} \mathrm{MLP}^{(t)}\left(\boldsymbol{h}_i^{(t-1)}, \boldsymbol{h}_j^{(t-1)}\right)$

Predicting shortest path predecessor:

[Veličković et al. ICLR'20]



Outline

1. Introduction
2. Understanding max-aggregation
3. **Reasoning on natural inputs**

Key question

Key question in neural algorithmic alignment:

If we're just teaching a NN to **imitate** a classical algorithm…

Why not just run that algorithm?

Why use GNNs for algorithm design?

Classical algorithms are designed with **abstraction** in mind

Enforce their inputs to conform to stringent preconditions

However, we design algorithms to solve **real-world** problems!

Our **goals** can be at odds with our **methods**

*Example: Harris and Ross's ['55] study of the **max flow** problem*

Slide inspired by Petar Veličković

Original max flow study

SECRET

U.S. AIR FORCE

PROJECT RAND

RESEARCH MEMORANDUM

FUNDAMENTALS OF A METHOD FOR EVALUATING
RAIL NET CAPACITIES (U)

T. E. Harris
F. S. Ross

RM-1573

October 24, 1955 Copy No. 137

SUMMARY

Air power is an effective means of interdicting an enemy's rail system, and such usage is a logical and important mission for this Arm.

As in many military operations, however, the success of interdiction depends largely on how complete, accurate, and timely is the commander's information, particularly concerning the effect of his interdiction-program efforts on the enemy's capability to move men and supplies. This information should be available at the time the results are being achieved.

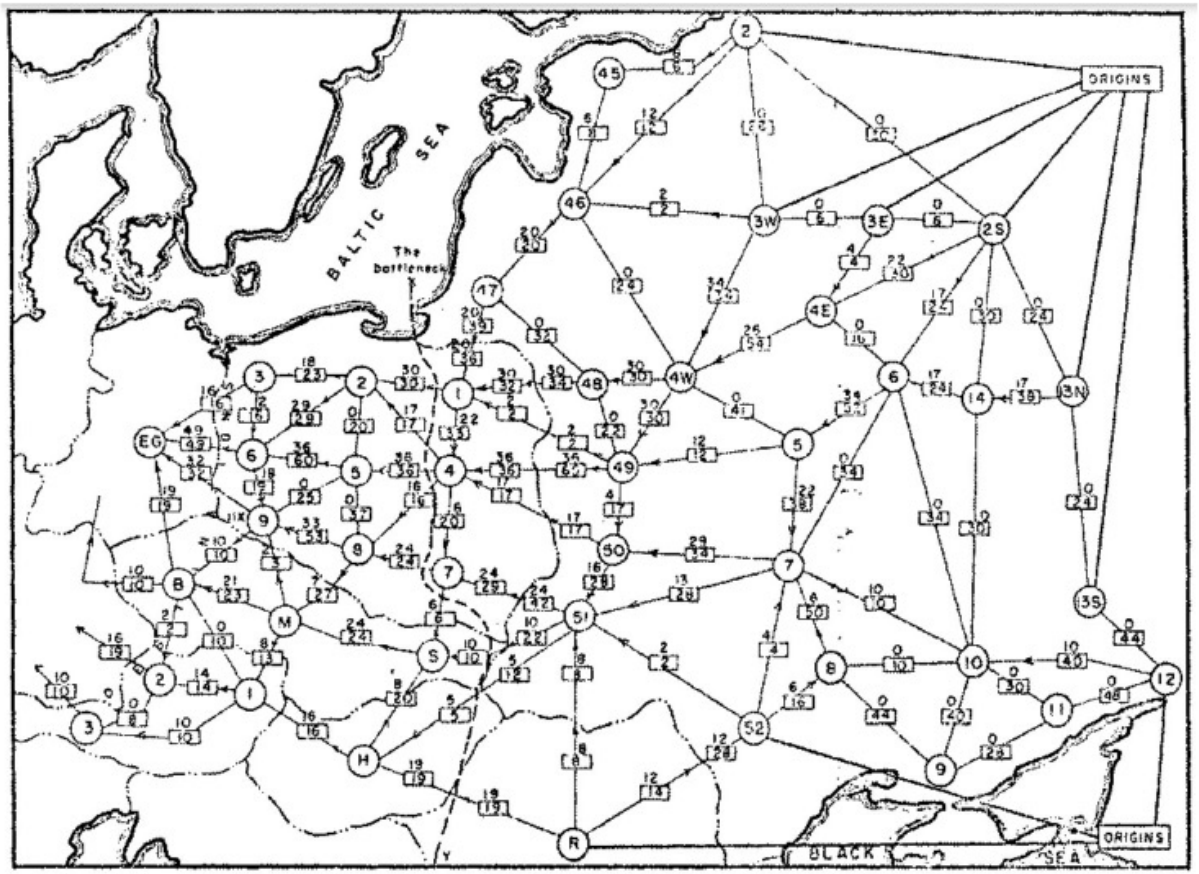
Slide by Veličković

The Warsaw Pact railway network

Goal: Find the "bottleneck"

I.e., the **minimum cut**

Equivalent to **max flow**



Slide by Veličković

Abstractification ⇒ information loss

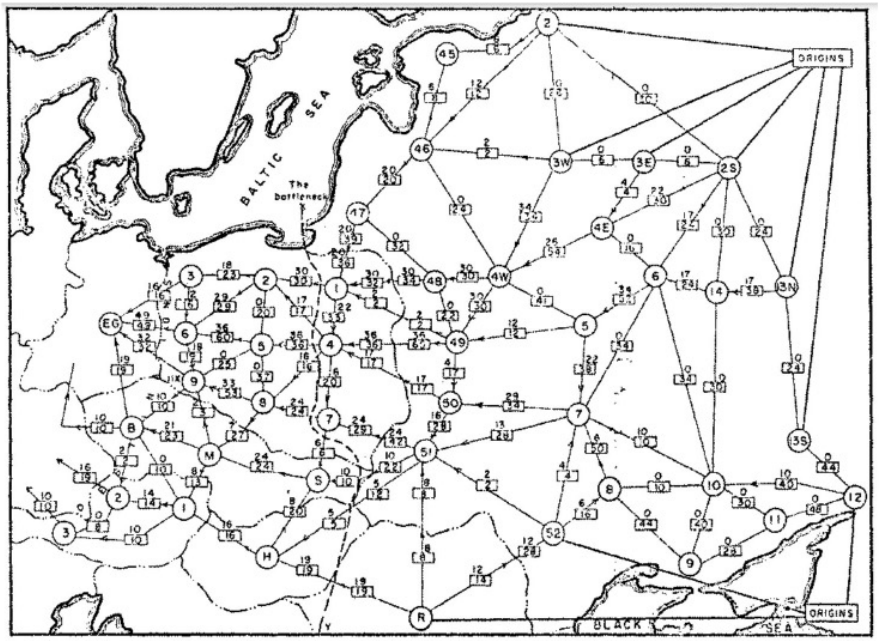
II. THE ESTIMATING OF RAILWAY CAPACITIES

The evaluation of both railway system and individual track capacities is, to a considerable extent, an art. The authors know of no tested mathematical model or formula that includes all of the variations and imponderables that must be weighed.* Even when the individual has been closely associated with the particular territory he is evaluating, the final answer, however accurate, is largely one of judgment and experience.

Slide by Veličković

Abstractification ⇒ information loss

- Attaches a **single, scalar capacity** to an entire railway system
- Ignores a **wealth of information** from the underlying system



Amazon Last Mile Routing Challenge

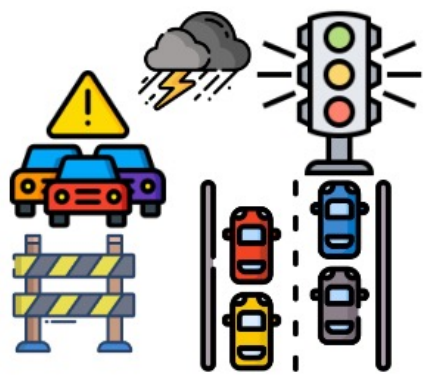
"Important gap between **theoretical** route planning and **real-life** route execution"

…"In real-life operations, the quality of a route is not exclusively defined by its **theoretical length, duration, or cost**"

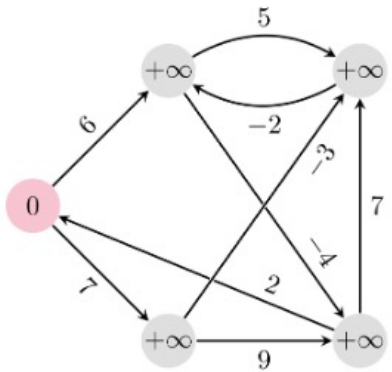
Many factors affect whether a driver "can **effectively, safely and conveniently** execute the planned route under real-life conditions."

Abstractifying the core problem

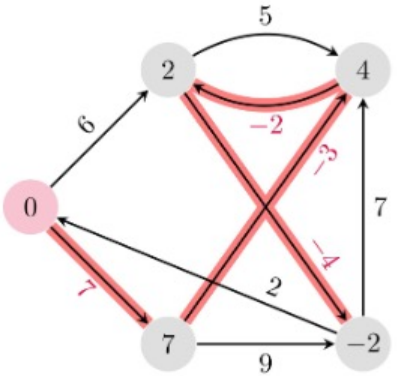
- Assume we have real-world inputs
  …but algorithm only admits abstract inputs
- Could try **manually** converting from one input to another



Natural inputs



Abstract inputs

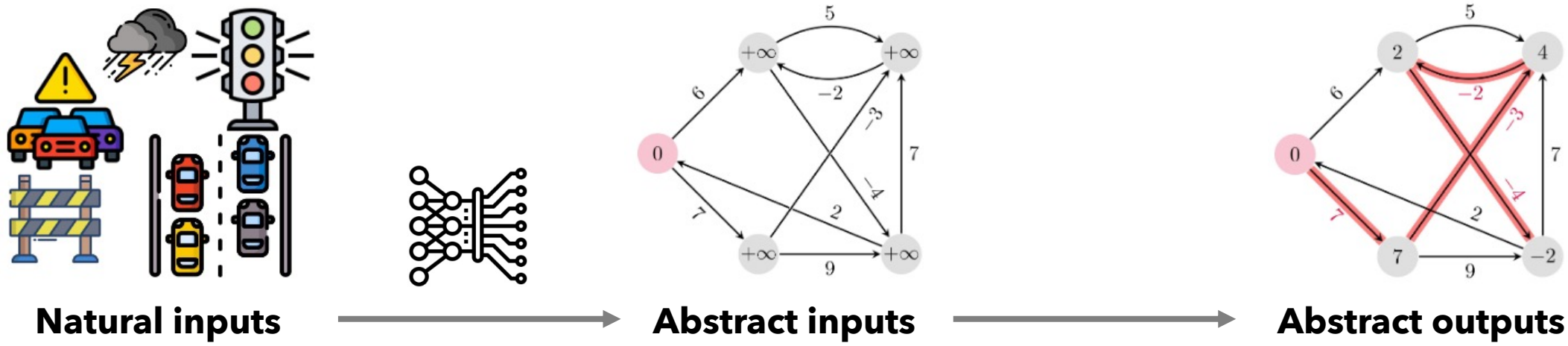


Abstract outputs

Slide by Veličković

Attacking the core problem

- Alternatively, **replace** human feature extractor with NN
  - Still apply same combinatorial algorithm
- Issue: algorithms typically perform **discrete optimization**
  - Doesn't play nicely with **gradient-based** optimization of NNs



Slide by Veličković

Algorithmic bottleneck

Second (more fundamental) issue: **data efficiency**

- Real-world data is often incredibly rich
- We still have to compress it down to scalar values

The algorithmic solver commits to using this scalar

Assumes it is perfect!

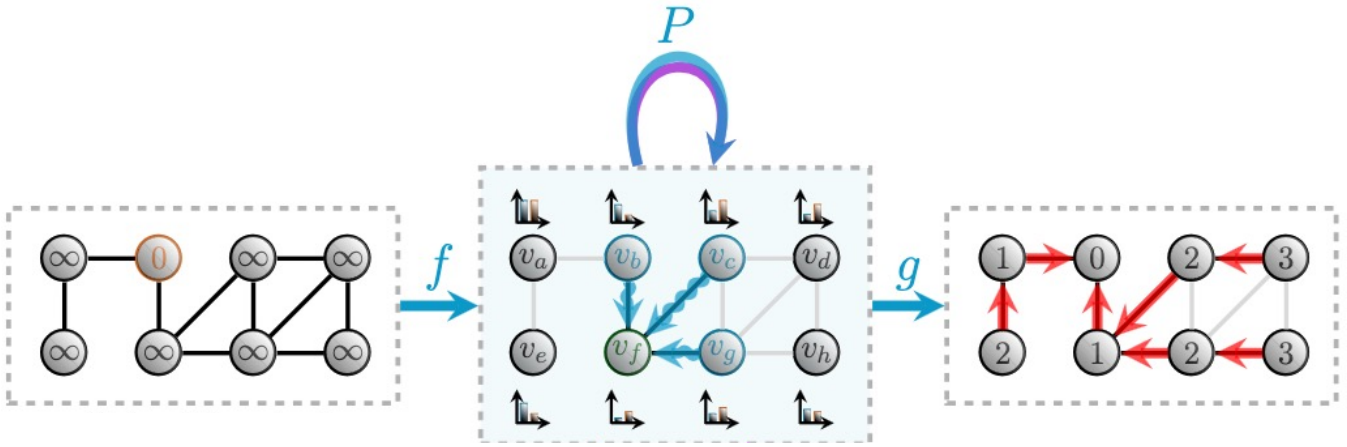
If there's insufficient training data to estimate the scalars:

- Alg will give a **perfect solution**
- …but in a **suboptimal environment**

Slide by Veličković

Neural algorithmic pipeline

Abstract inputs $\bar{x}$



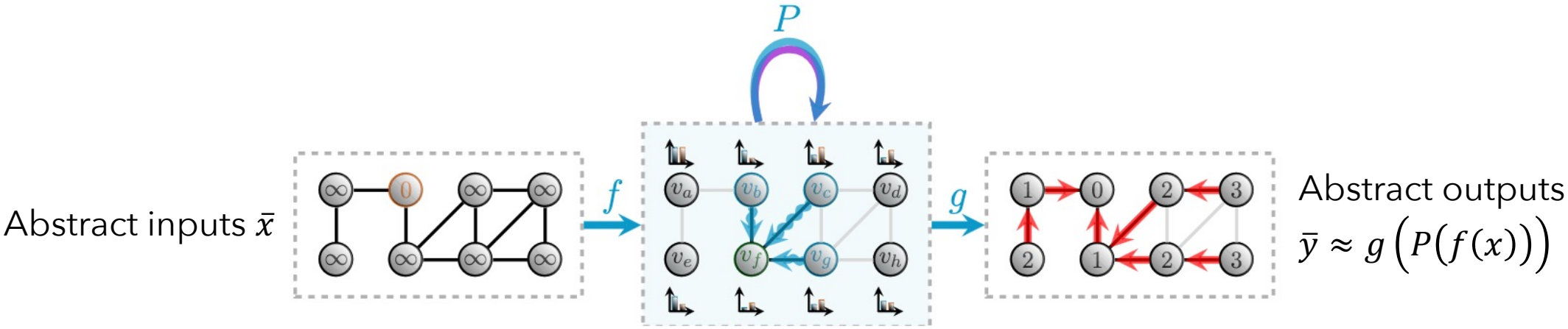
Abstract outputs

$\bar{y} \approx g\left(P(f(x))\right)$

1. On abstract inputs, learn encode-process-decode functions

Figure by Cappart et al.

Neural algorithmic pipeline

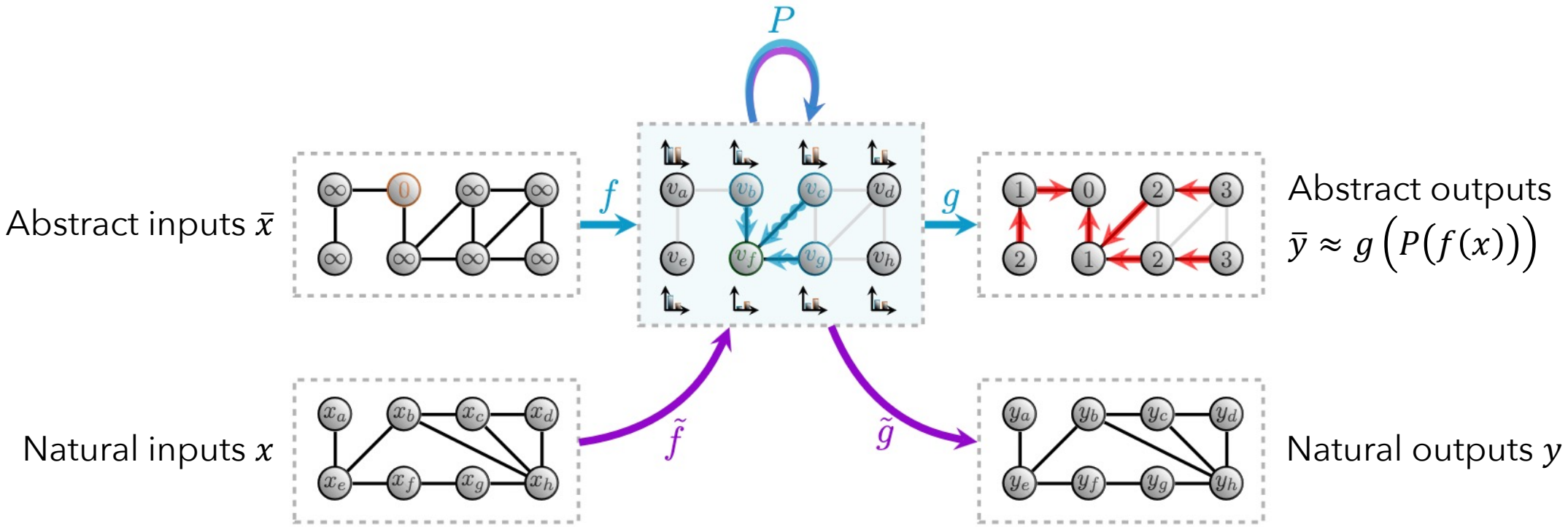


Processor P:

1. Is aligned with computations of target algorithm
2. Admits useful gradients
3. Operates over high-dim latent space (better use of data)

Figure by Cappart et al.

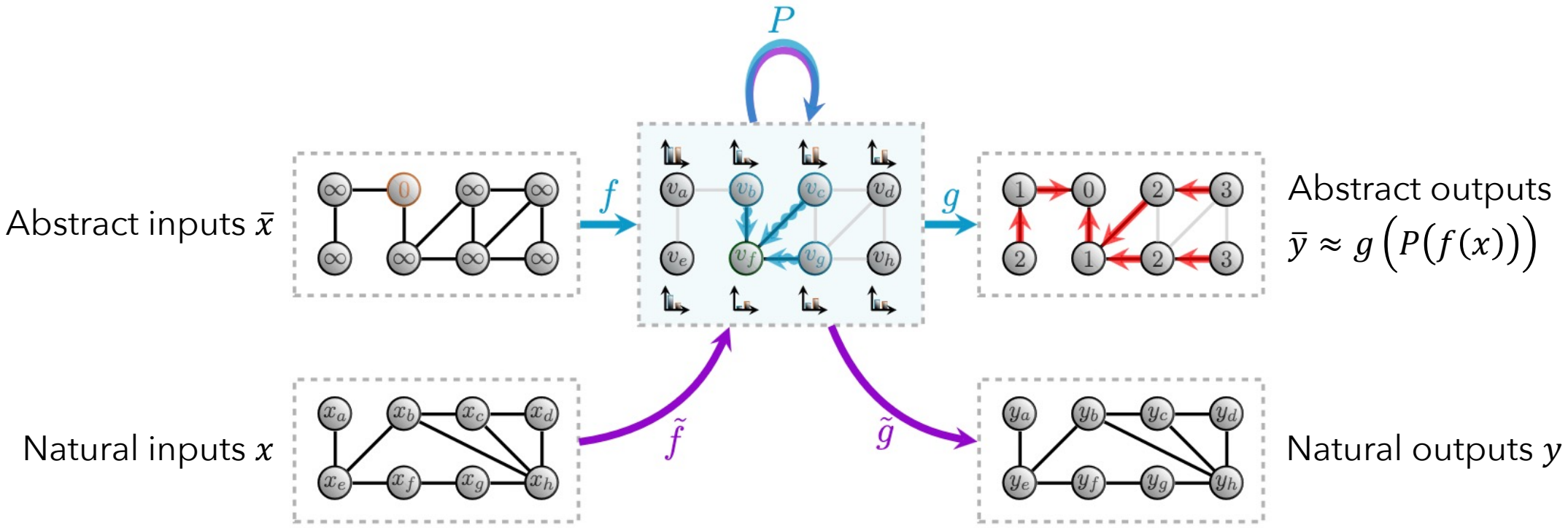
Neural algorithmic pipeline



2. Set up encode-decode functions for natural inputs/outputs

Figure by Cappart et al.

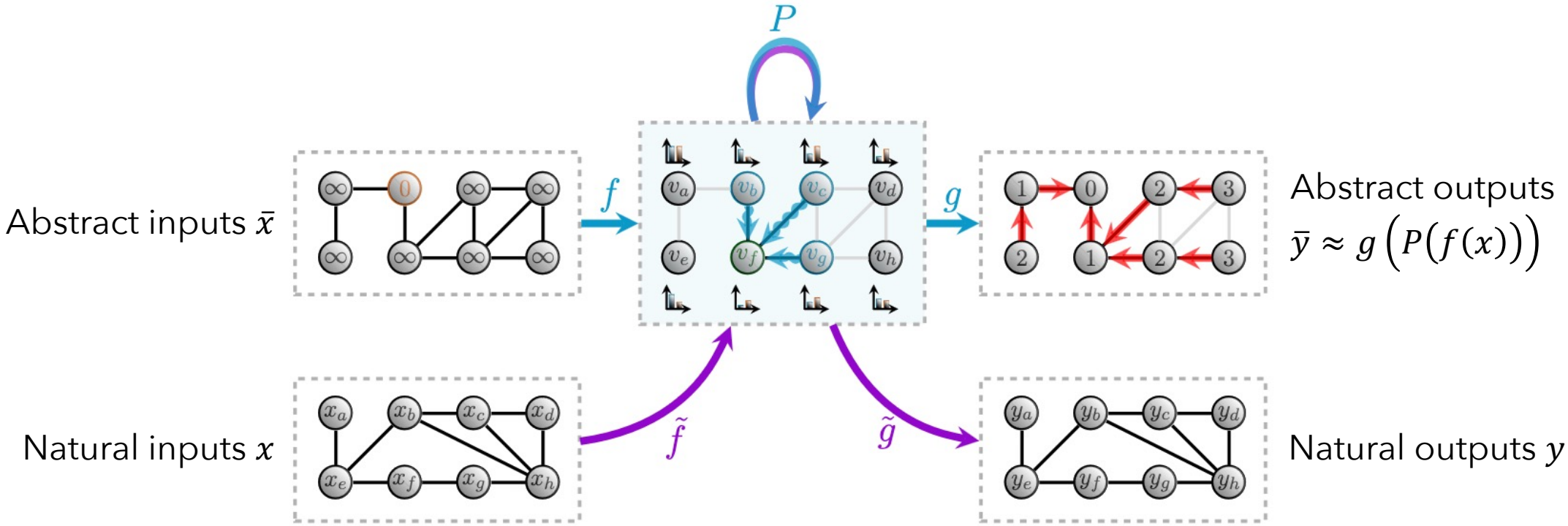
Neural algorithmic pipeline



3. Learn parameters using loss that compares $\tilde{g}\left(P\left(\tilde{f}(x)\right)\right)$ to y

Figure by Cappart et al.

Neural algorithmic pipeline



Note: Keep P frozen while learning $\tilde{f}$ and $\tilde{g}$

Figure by Cappart et al.

Overview

Survey: broad overview of **neural algorithmic reasoning**

Among other topics, covers:

- Why **max-aggregation** allows for extrapolation
  - MLPs extrapolate linearly
  - Must hard-code algorithmic non-linearities
- The **neural algorithmic pipeline**
  - Allows us to solve natural, real-world instances