

DeepMind

# Neural Algorithmic Reasoning

Petar Veličković

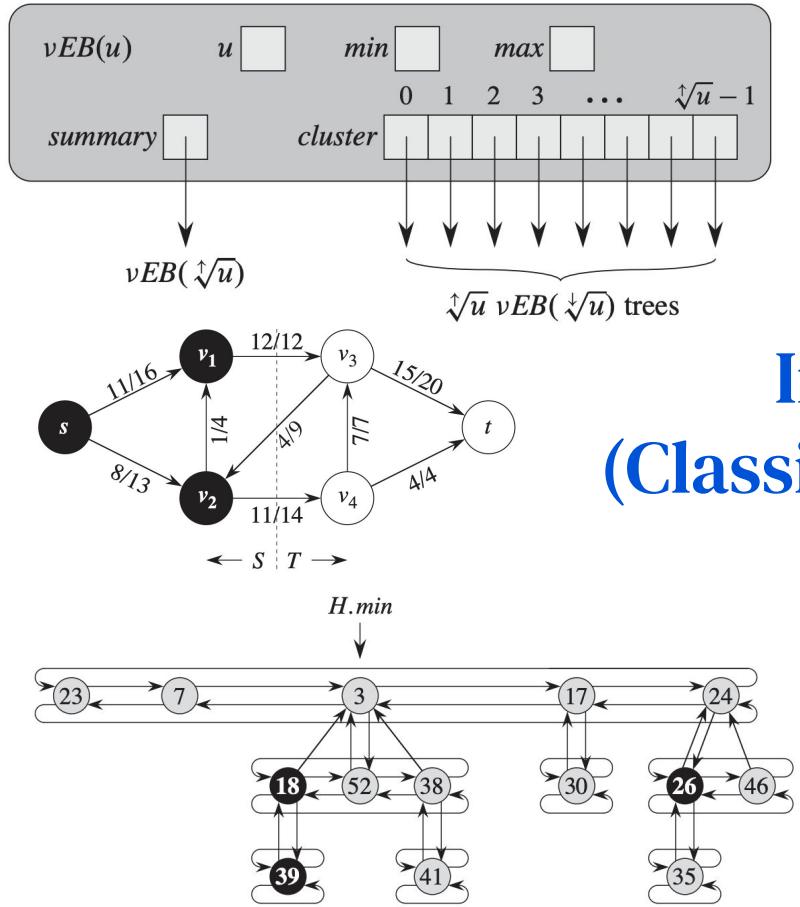
DLG-KDD'21  
15 August 2021



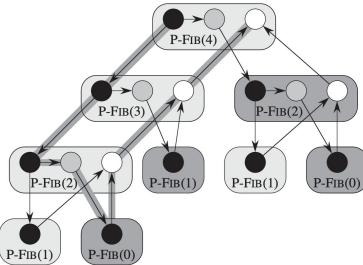
DeepMind

In this talk:  
**(Classical) Algorithms**





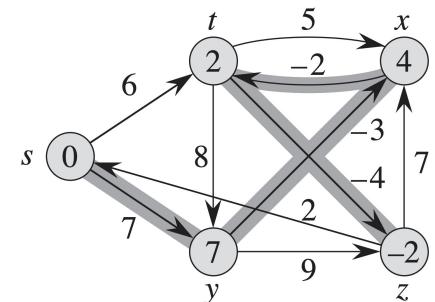
DeepMind

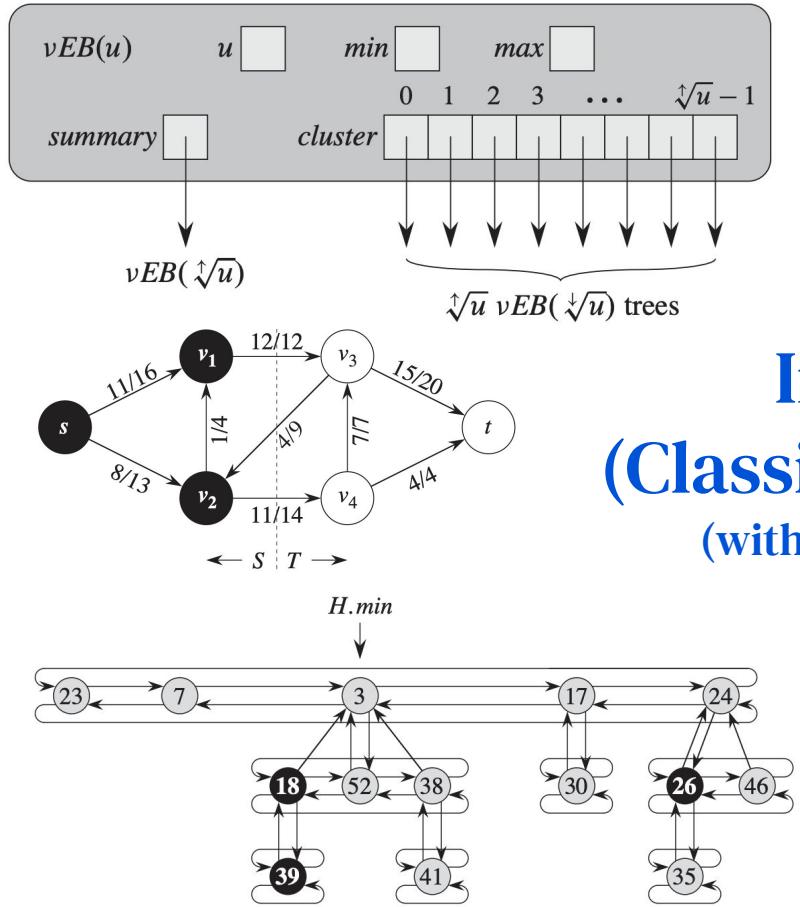


## In this talk: (Classical) Algorithms

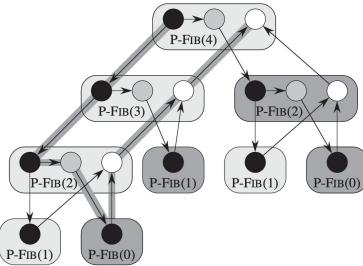
```
MERGE-SORT( $A, p, r$ )
1  if  $p < r$ 
2     $q = \lfloor (p + r)/2 \rfloor$ 
3    MERGE-SORT( $A, p, q$ )
4    MERGE-SORT( $A, q + 1, r$ )
5    MERGE( $A, p, q, r$ )
```

$j$	0	1	2	3	4	5	6
$i$	$y_j$	$B$	$D$	$C$	$A$	$B$	$A$
0	$x_i$	0	0	0	0	0	0
1	$A$	0	0	0	0	1	1
2	$B$	0	1	1	1	2	2
3	$C$	0	1	1	2	2	2
4	$B$	0	1	1	2	2	3
5	$D$	0	1	2	2	3	3
6	$A$	0	1	2	3	3	4
7	$B$	0	1	2	3	4	4





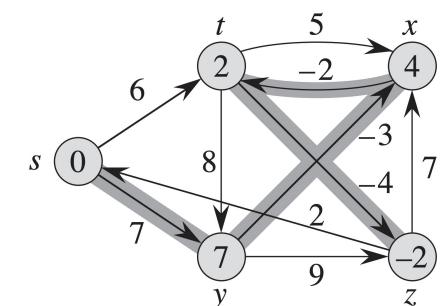
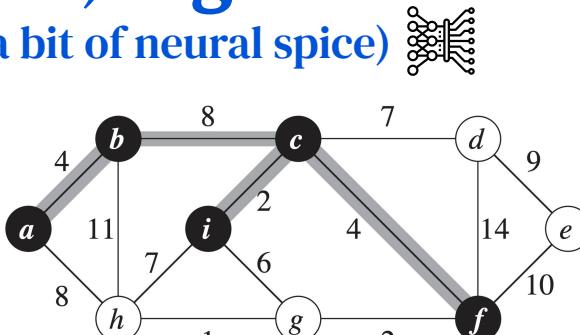
DeepMind



## In this talk: (Classical) Algorithms (with a bit of neural spice)

```
MERGE-SORT( $A, p, r$ )
1   if  $p < r$ 
2      $q = \lfloor (p + r)/2 \rfloor$ 
3     MERGE-SORT( $A, p, q$ )
4     MERGE-SORT( $A, q + 1, r$ )
5     MERGE( $A, p, q, r$ )
```

$j$	0	1	2	3	4	5	6
$i$	$y_j$	$B$	$D$	$C$	$A$	$B$	$A$
0	$x_i$	0	0	0	0	0	0
1	$A$	0	0	0	0	1	1
2	$B$	0	1	-1	-1	1	2
3	$C$	0	1	1	2	2	2
4	$B$	0	1	1	2	2	3
5	$D$	0	1	2	2	3	3
6	$A$	0	1	2	3	3	4
7	$B$	0	1	2	3	4	4



# Overview

Our aim is to address **three** key questions: (roughly ~10min for each)

- Why should we, as deep learning practitioners, study **algorithms**?
  - Further, why might it be beneficial to make '*algorithm-inspired*' neural networks?
- How to **build** neural networks that behave algorithmically?
  - And why am I even telling you this in a "*Graph Machine Learning*" context?
- Do algorithmic neural networks actually **work** when deployed?
  - If so, how are they *actually* being used?

Hopefully, also some ideas on **where** you might be able to **apply** the ideas above :)



1

# Motivation for studying algorithms



# Why algorithms?

- Essential “**pure**” forms of combinatorial **reasoning**
  - ‘Timeless’ principles that will remain regardless of the model of computation
  - Completely decoupled from any form of **perception**\*

*\*though perception itself may also be expressed in the language of algorithms*



# Why algorithms?

- Essential “**pure**” forms of combinatorial **reasoning**
  - ‘Timeless’ principles that will remain regardless of the model of computation
  - Completely decoupled from any form of **perception**\*
- **Favourable** properties
  - Trivial **strong** generalisation
  - **Compositionality** via *subroutines*
  - Provable **correctness** and **performance** guarantees
  - Interpretable **operations** / *pseudocode*



# Why algorithms?

- Essential “**pure**” forms of combinatorial **reasoning**
  - ‘Timeless’ principles that will remain regardless of the model of computation
  - Completely decoupled from any form of **perception**\*
- **Favourable** properties
  - Trivial **strong** generalisation
  - **Compositionality** via *subroutines*
  - Provable **correctness** and **performance** guarantees
  - Interpretable **operations** / *pseudocode*
- Hits *close to home*
  - Algorithms and competitive programming are how I got into Computer Science



2

# Maximum flow and the Ford-Fulkerson algorithm



# Maximum flow problem

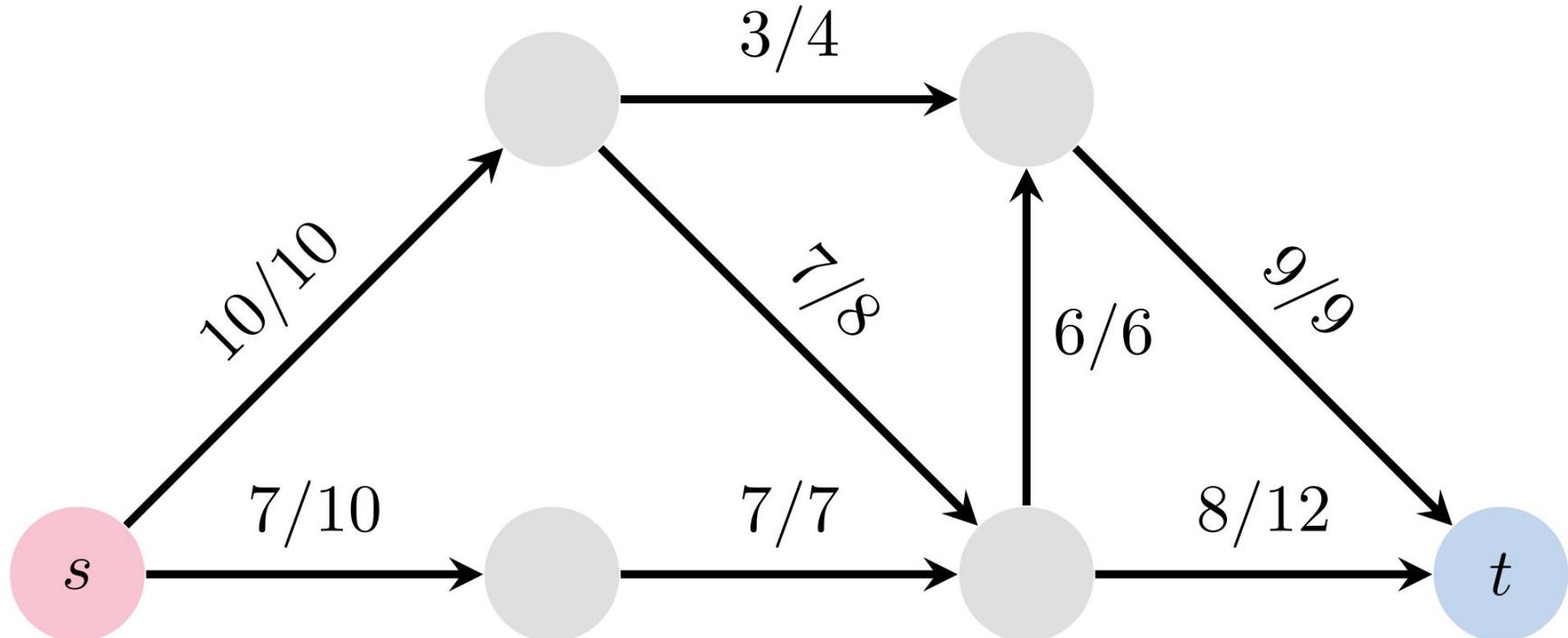
- **Flow network:** graph  $G = (V, E)$ , augmented with a *capacity function*,  $c: V \times V \rightarrow \mathbb{R}^+$ 
  - Capacity  $c_{uv}$  denotes how much **flow** is allowed on  $(u, v)$  edge
- Two special nodes: source,  $s$ , and sink,  $t$ 
  - Source unleashes “infinite” capacity, sink receives “infinite” capacity
- A **flow** in  $G$  is any mapping  $f: V \times V \rightarrow \mathbb{R}^+$ , such that:

$$\begin{aligned}\forall u, v \in V \quad f_{u,v} &\leq c_{u,v} \\ \forall u \in V \setminus \{s, t\} \quad \sum_{v \in V} f_{v,u} &= \sum_{v \in V} f_{u,v}\end{aligned}$$

- The **value** of a flow is the total flow emanating from the source:
$$\sum_{v \in V} f_{s,v} - \sum_{v \in V} f_{v,s}$$
  - We are interested in **maximising** it!



## Max-flow example ( $f=17$ )



# Ford-Fulkerson's Algorithm

- Such a **rigorously** defined problem often admits remarkably **elegant** and **provably correct** algorithm blueprint!

FORD-FULKERSON-METHOD( $G, s, t$ )

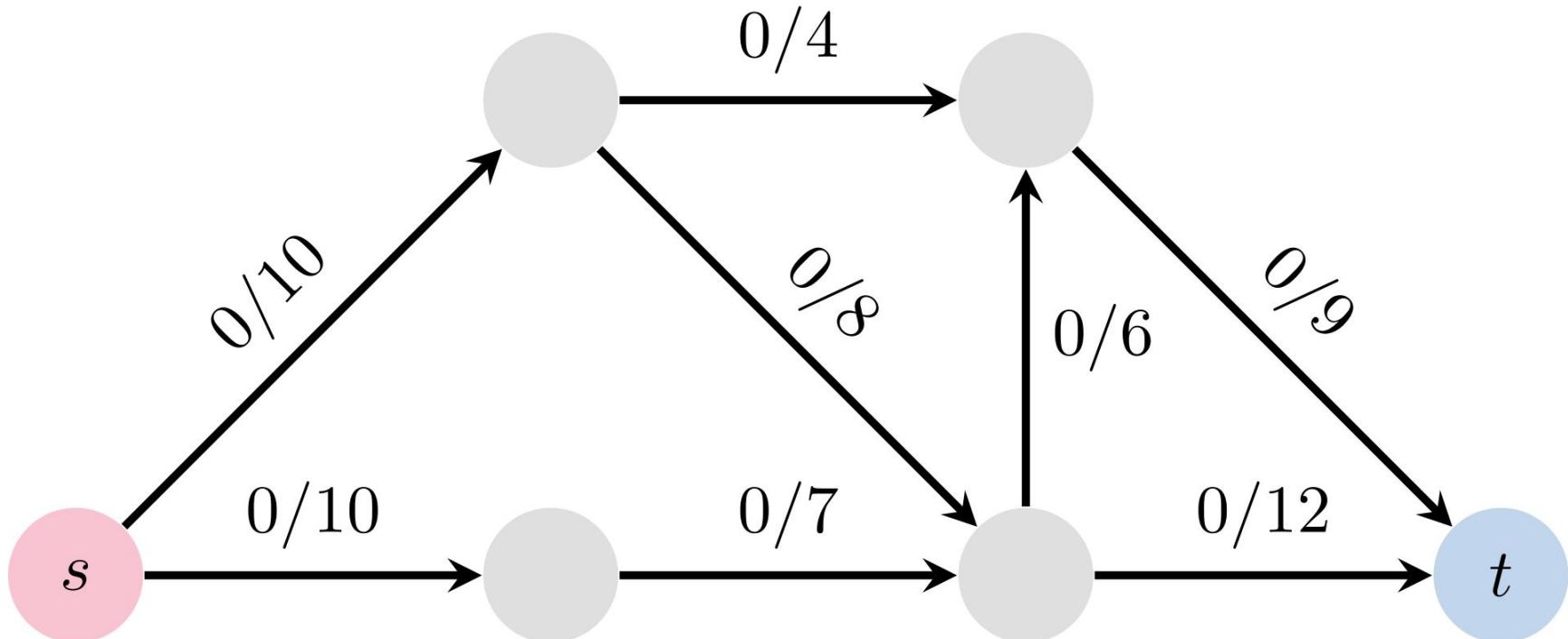
- 1 initialize flow  $f$  to 0
- 2 **while** there exists an augmenting path  $p$  in the residual network  $G_f$ 
  - 3     augment flow  $f$  along  $p$
  - 4 **return**  $f$

\*representing the capacities that remain after applying f

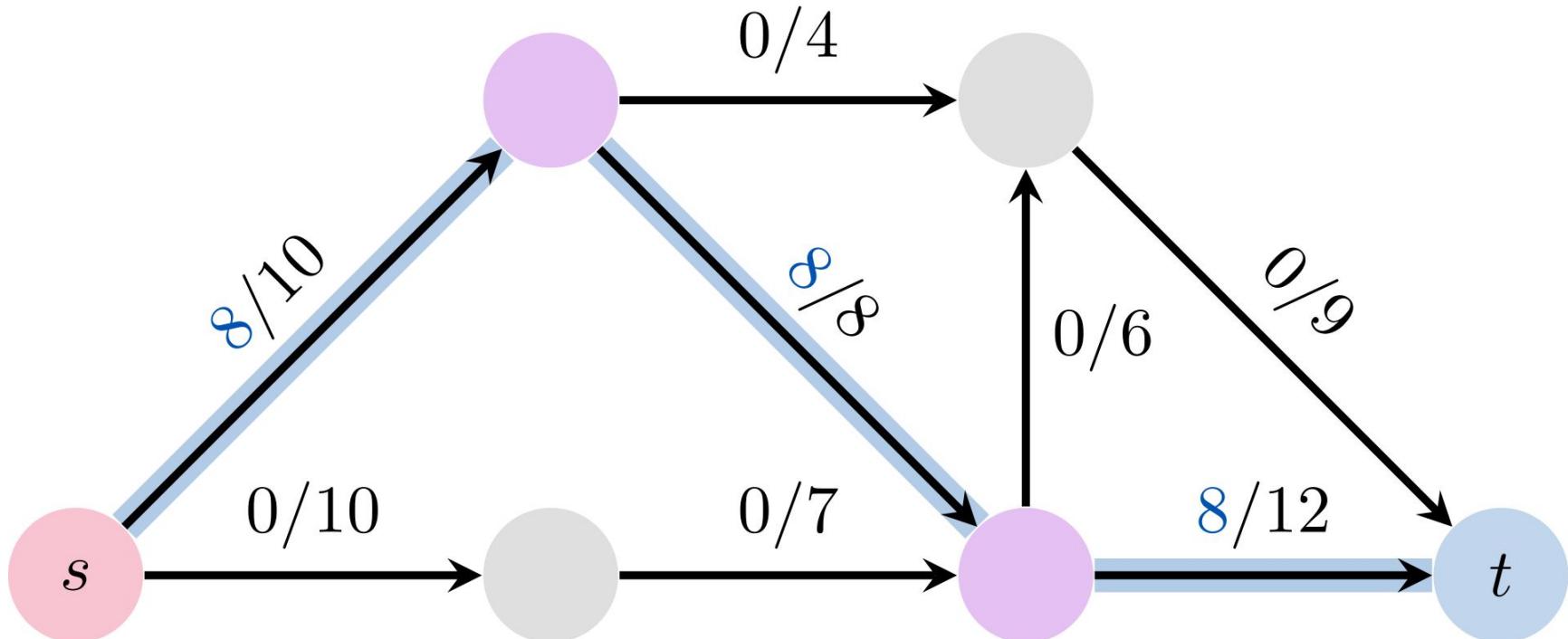
- Many specific ways to find  $p$  yield different algorithms (e.g. Edmonds–Karp, Dinitz, etc...)
  - This can be proven to terminate with correct solution



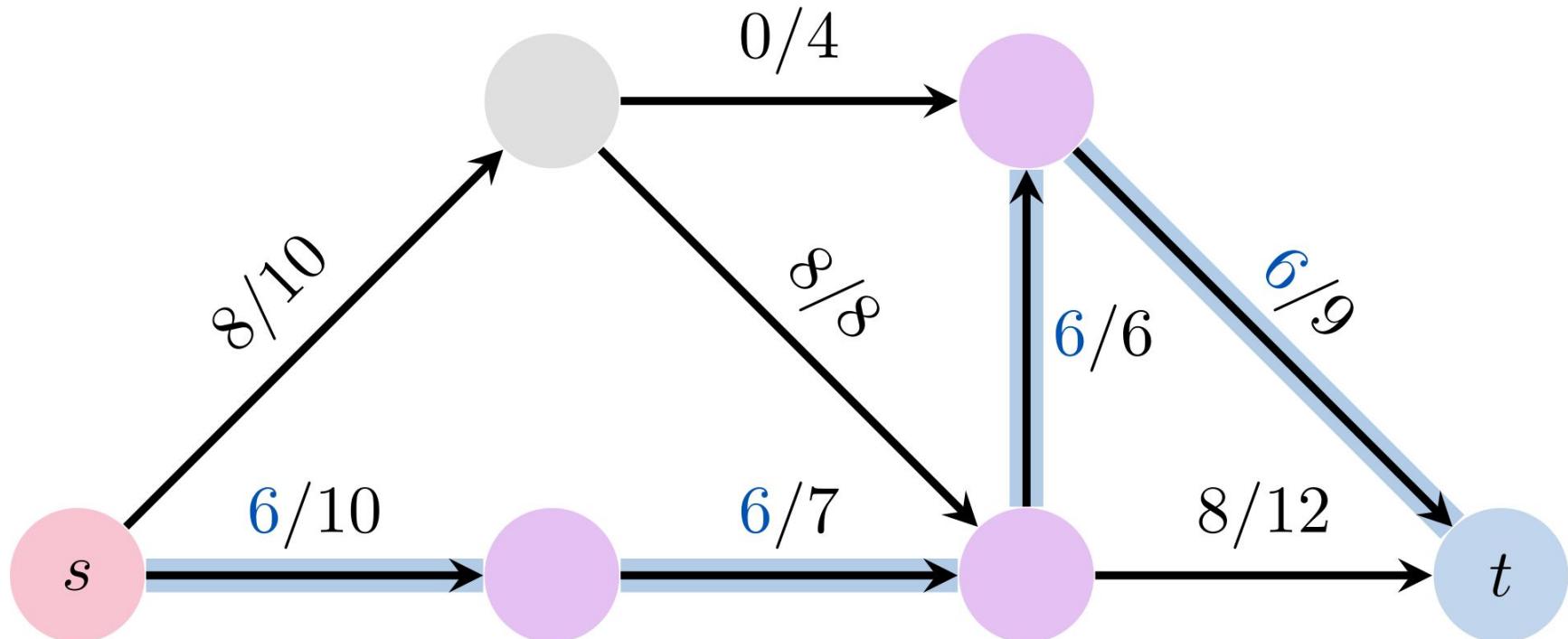
## Ford-Fulkerson in action



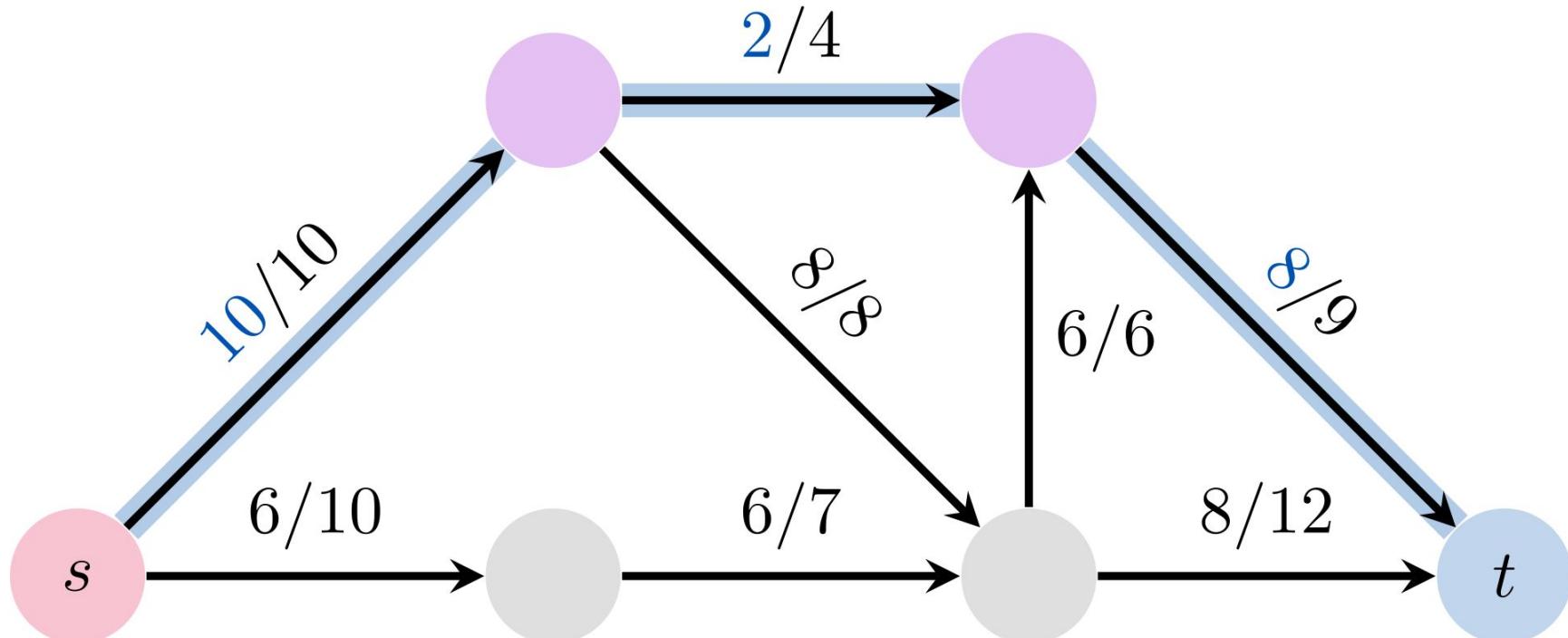
## Ford-Fulkerson in action



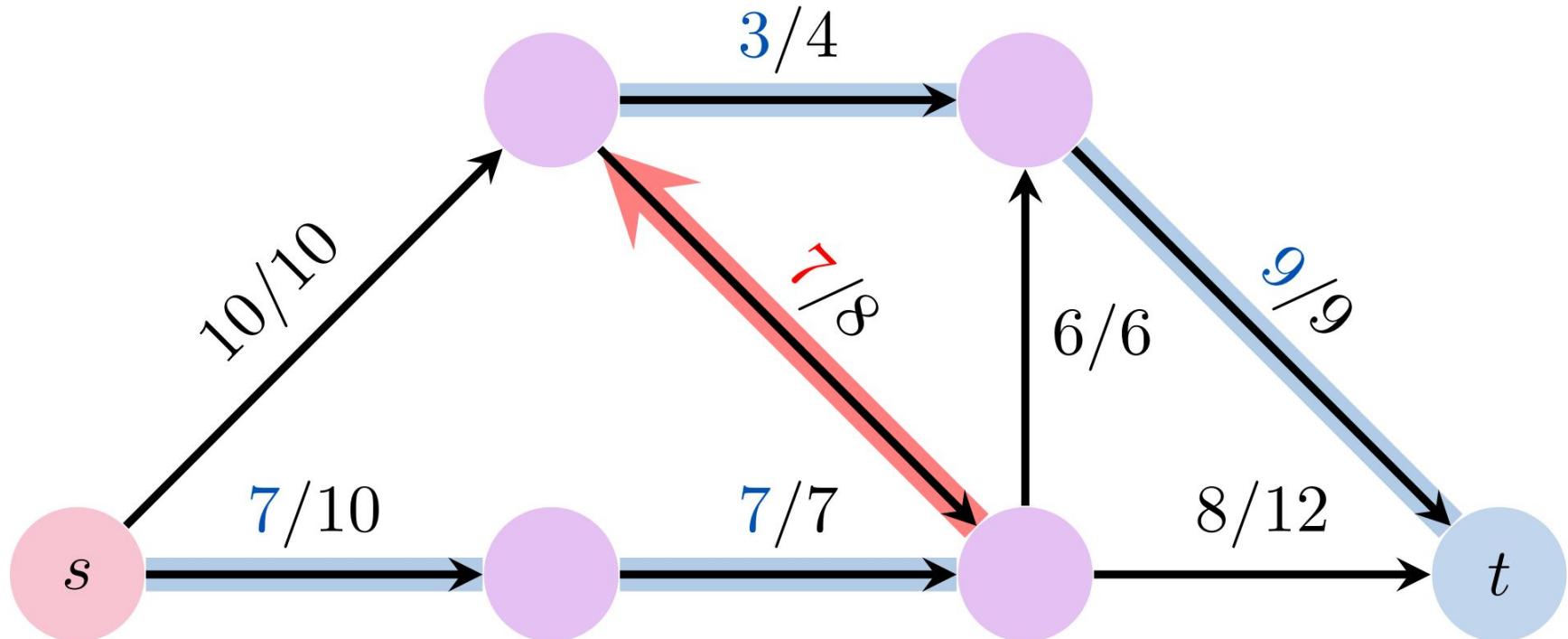
## Ford-Fulkerson in action



## Ford-Fulkerson in action



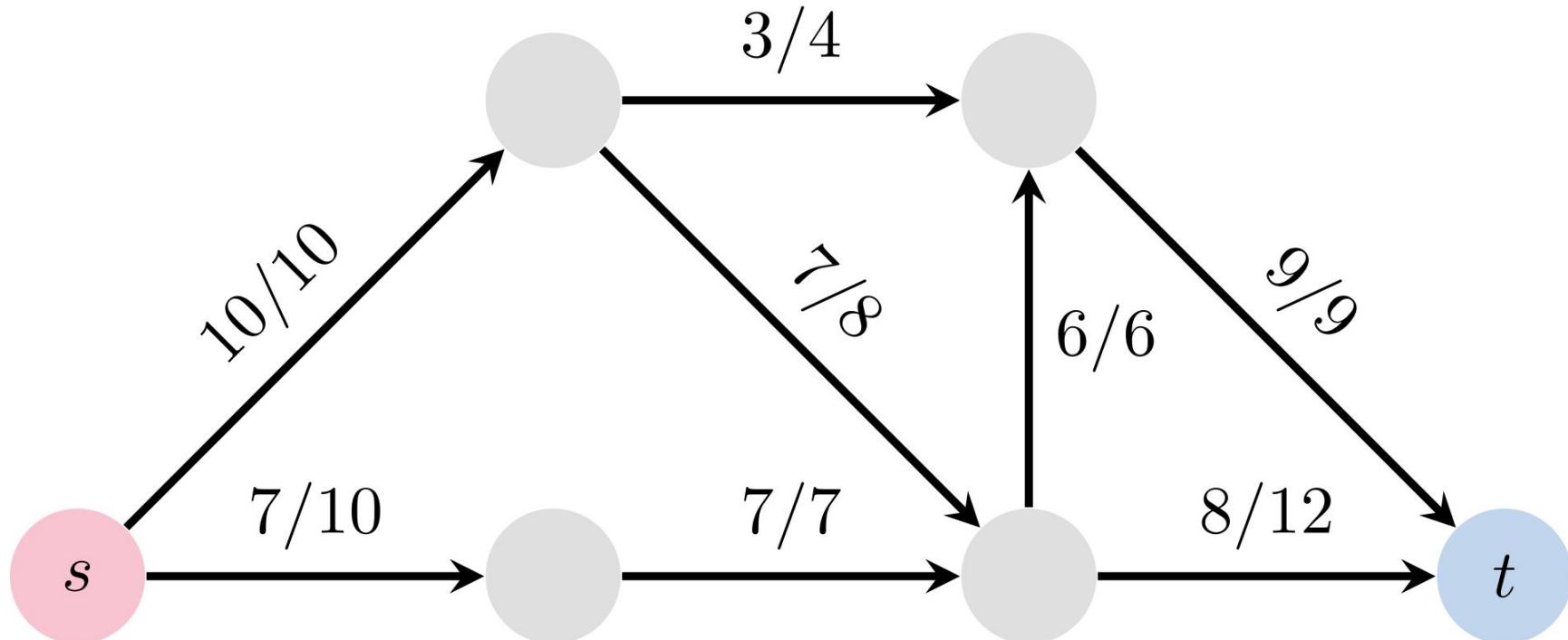
## Ford-Fulkerson in action



(the flow may also be **returned!**)



# Final solution!



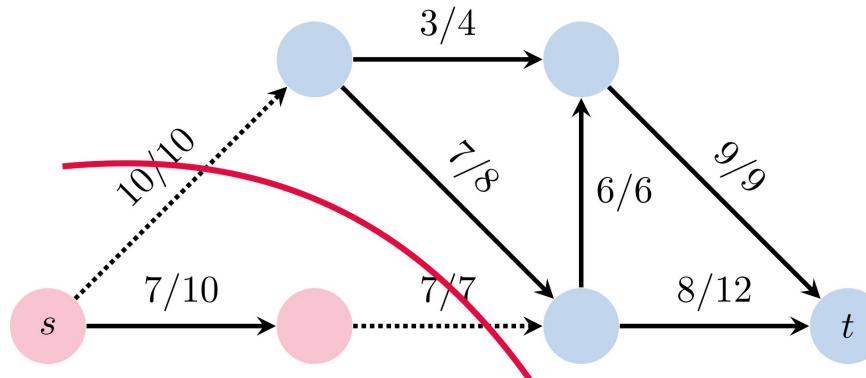
# Max-flow Min-cut theorem

Observing data in this way, also yields easy observation of **connections**, hence **theorems!**

## **Theorem 26.6 (Max-flow min-cut theorem)**

If  $f$  is a flow in a flow network  $G = (V, E)$  with source  $s$  and sink  $t$ , then the following conditions are equivalent:

1.  $f$  is a maximum flow in  $G$ .
2. The residual network  $G_f$  contains no augmenting paths.
3.  $|f| = c(S, T)$  for some cut  $(S, T)$  of  $G$ .



3

“Fundamentals  
of a method for  
evaluating rail  
net capacities”

(Harris & Ross, 1955)



# The core problem

- Classical algorithms are designed with *abstraction* in mind, enforcing their inputs to conform to stringent **preconditions**.
  - Keeping the inputs constrained enables an uninterrupted focus on “reasoning”
  - Easily certify the resulting procedure’s correctness, i.e., stringent **postconditions**
- However, we must never forget **why** we design algorithms!
- Unfortunately, this is at **timeless odds** with the way they are designed
  - Let’s study an example from the 1950s.



# Original interest in flows

SECRET

SUMMARY

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. 37

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.

<https://apps.dtic.mil/dtic/tr/fulltext/u2/093458.pdf>

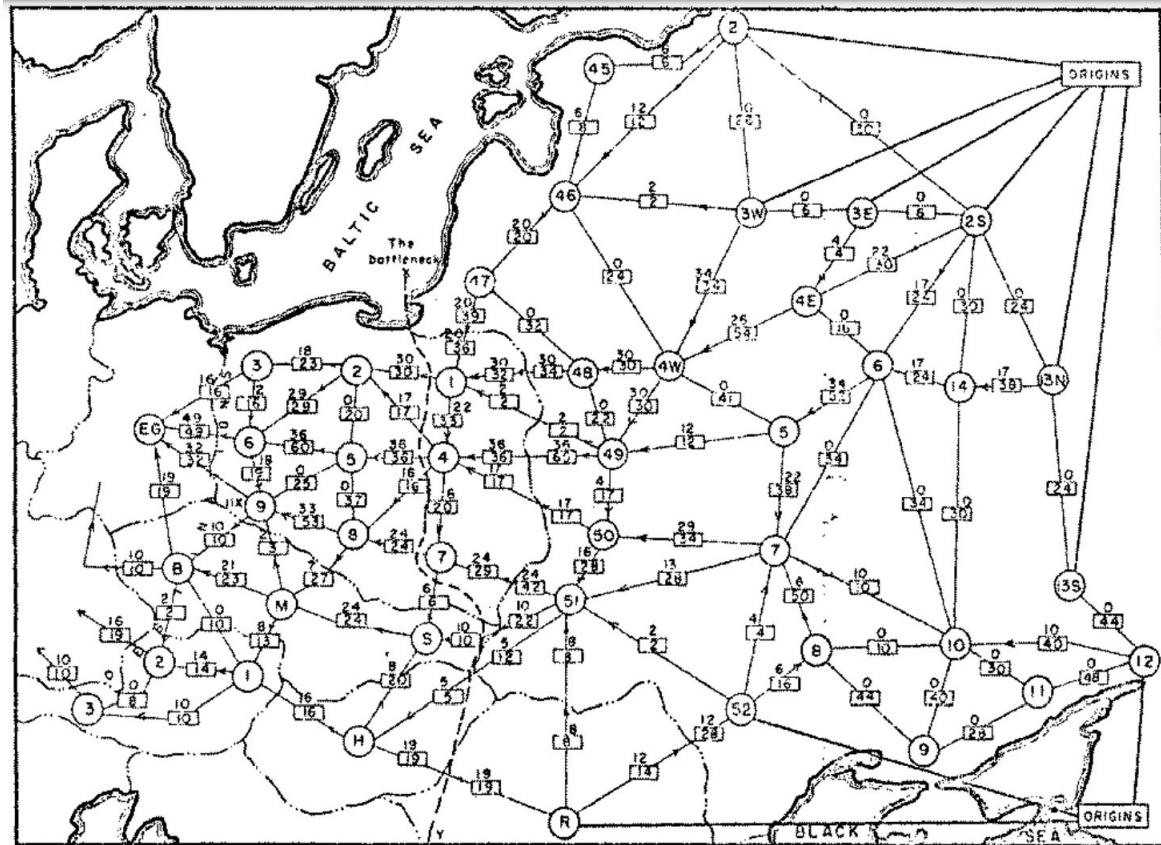


# The Warsaw Pact railway network

Find “the bottleneck”, i.e.  
the **minimum cut**.

As we know, this is directly  
related to computing the  
**maximum flow**.

(this was *intuitively* assumed by  
Harris & Ross as well)



# The core problem, as seen in 1955

## 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.



# An important issue for the community

- The “core problem” plagues applications of classical combinatorial algorithms to this day!
- Satisfying their preconditions necessitates converting inputs into an **abstractified** form
- If done manually, this often implies *drastic information loss*
  - Combinatorial problem no longer accurately portrays the dynamics of the real world.
  - Algorithm will give a **perfect** solution, but in a **useless** environment
- The data we need to apply the algorithm may be only **partially** observable
  - This can often render the algorithm completely inapplicable.
- An issue of high interest for *both* combinatorial and operations research communities.



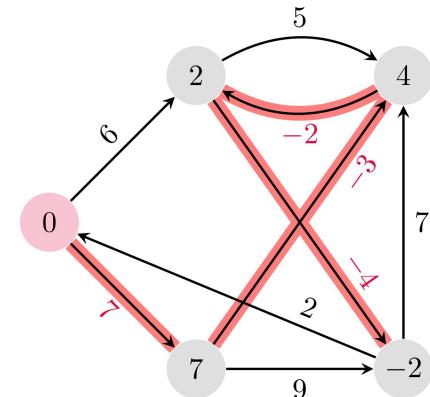
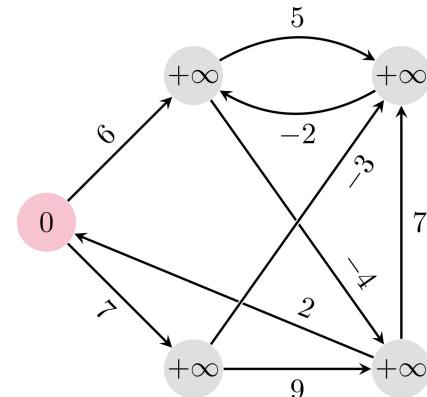
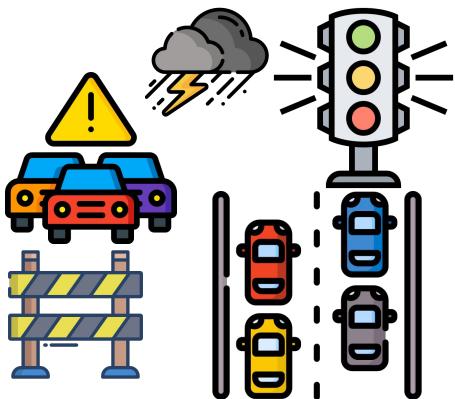
4

Towards a  
*neurally spiced*  
solution



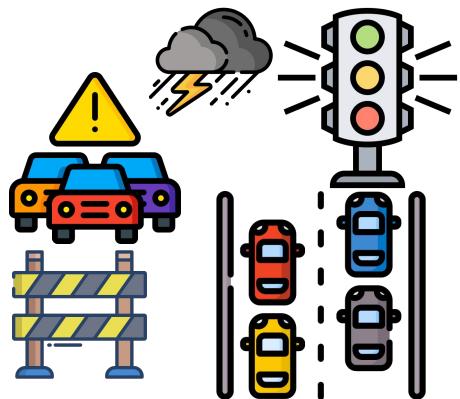
# Abstractifying the core problem

- Assume we have *real-world* inputs, but our algorithm only admits *abstract* inputs
  - For now, we assumed **manually** converting from one input to another

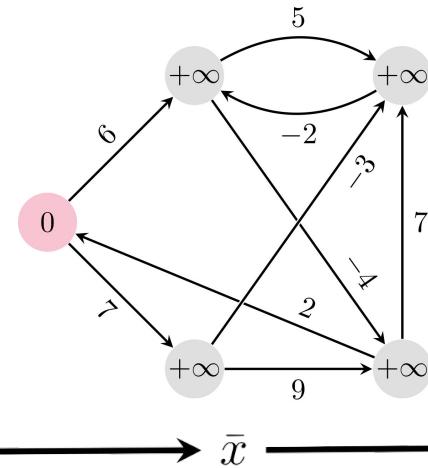


# Abstractifying the core problem

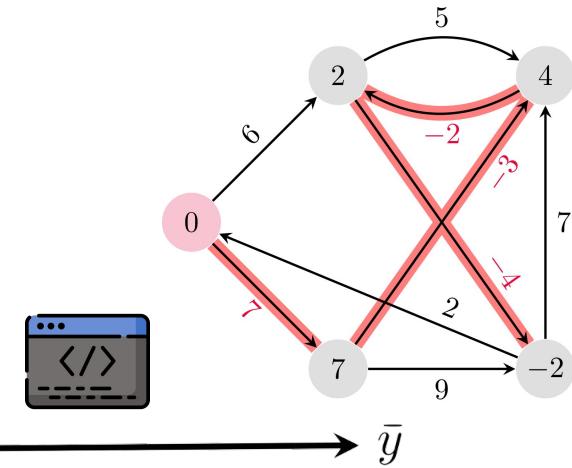
- Assume we have *real-world* inputs, but our algorithm only admits *abstract* inputs
  - For now, we assumed **manually** converting from one input to another



Natural inputs



Abstract inputs



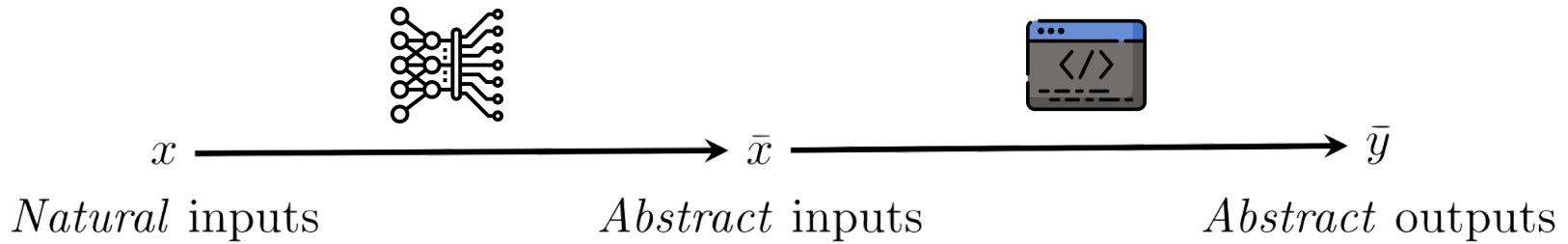
Abstract outputs

- Whenever we have **manual** feature engineering of **raw data**, **neural nets** are attractive!



# Attacking the core problem

- First point of attack: “good old deep learning”
  - Replace human feature extractor with **neural network**
  - Still apply the same combinatorial algorithm



- **First issue:** algorithms typically perform **discrete optimisation**
  - This does not play nicely with gradient-based optimisation that neural nets require.

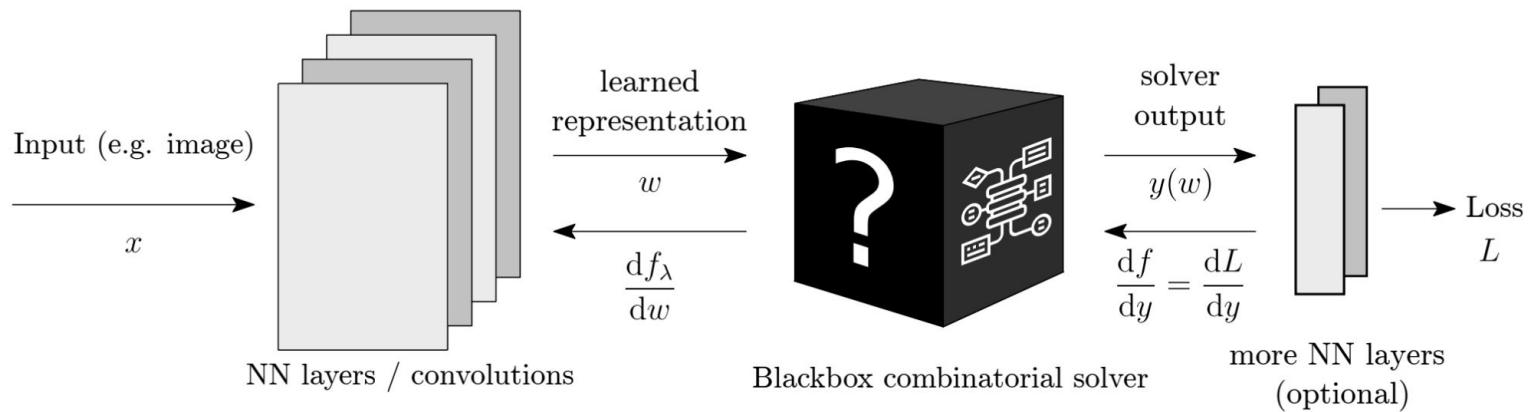


# Backpropagating through classical algorithms

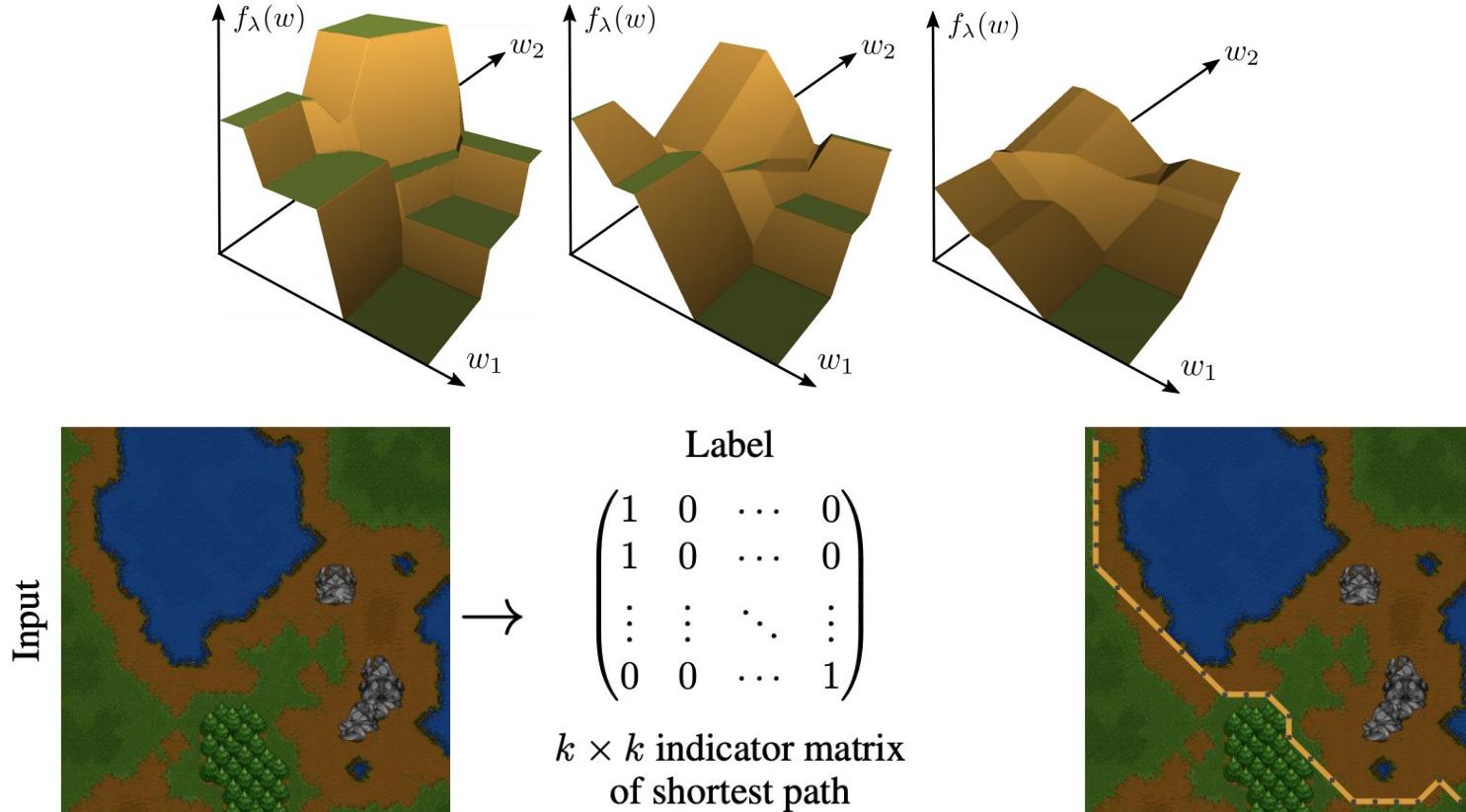
Vlastelica et al. (ICLR'20) provide a great approach for differentiating CO solver outputs

## DIFFERENTIATION OF BLACKBOX COMBINATORIAL SOLVERS

Marin Vlastelica<sup>1\*</sup>, Anselm Paulus<sup>1\*</sup>, Vít Musil<sup>2</sup>, Georg Martius<sup>1</sup>, Michal Rolínek<sup>1</sup>



# Black-box backprop



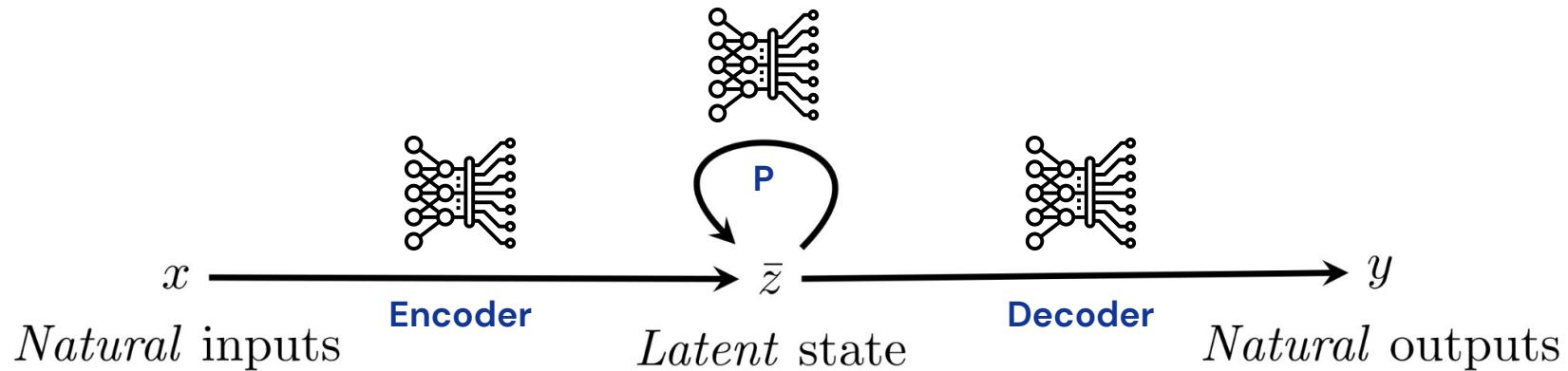
# Algorithmic **bottleneck**

- 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 are insufficient training data to properly estimate the scalars, we hit same issues!
  - Algorithm will give a **perfect** solution, but in a **suboptimal** environment



# Breaking the bottleneck

- Neural networks derive great flexibility from their **latent** representations
  - They are inherently *high-dimensional*
  - If any component is poorly predicted, others can step in and compensate!
- To break the bottleneck, we replace the algorithm with a **neural network**!

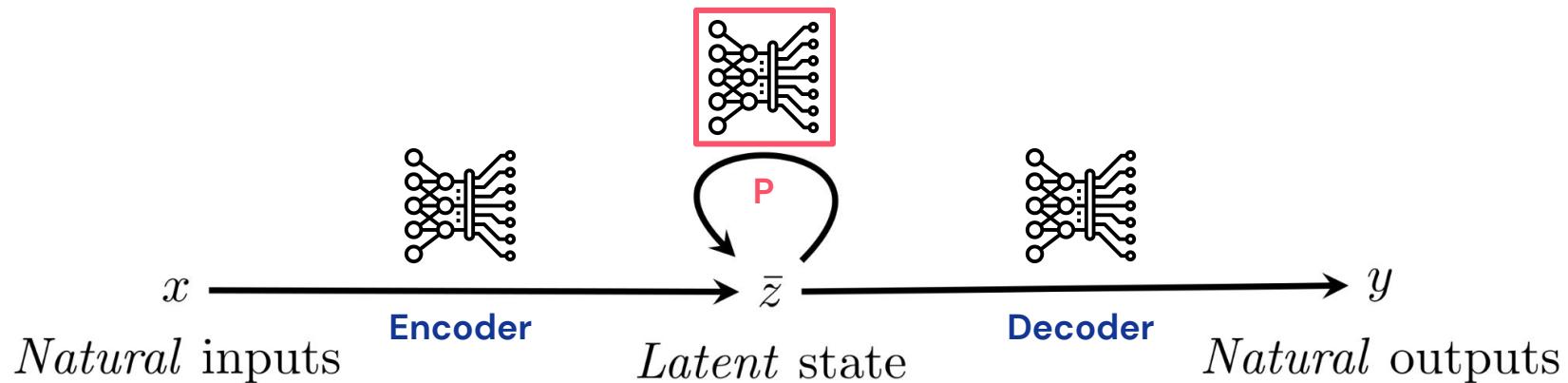


(The setting naturally aligns with *encode-process-decode* (Hamrick et al., CSS'18))



# Properties of this construction

- Assuming our **latent-state NN** aligns with the steps of an algorithm, we now have:
  - An **end-to-end** neural pipeline which is fully differentiable
  - No scalar-based bottlenecks, hence higher data efficiency.
- How do we obtain **latent-state neural networks** that **align** with algorithms?



5

# Algorithmic reasoning



# Algorithmic reasoning

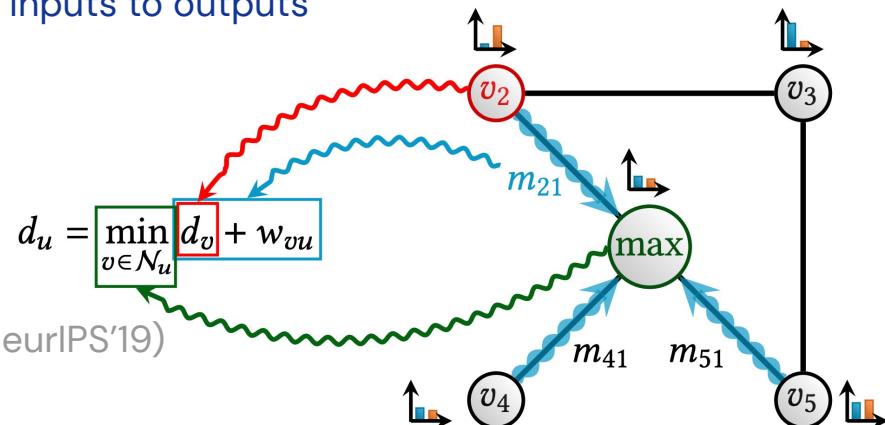
- The desiderata for our processor network  $P$  are slightly different than usual:
  - They are required to imitate the steps of the algorithm *faithfully*
  - This means they must **extrapolate!**
  - (*Related: how to best decide the **weights** of  $P$  to **robustly** match the algorithm?*)
- Neural networks typically **struggle** in the extrapolation regime!
- **Algorithmic reasoning** is an emerging area that seeks to ameliorate this issue
  - Primarily through theoretical and empirical prescriptions
  - These guide the neural architectures, inductive biases and featurisations that are useful for extrapolating combinatorially
- This is a **very** active research area, with many key papers published only last year!



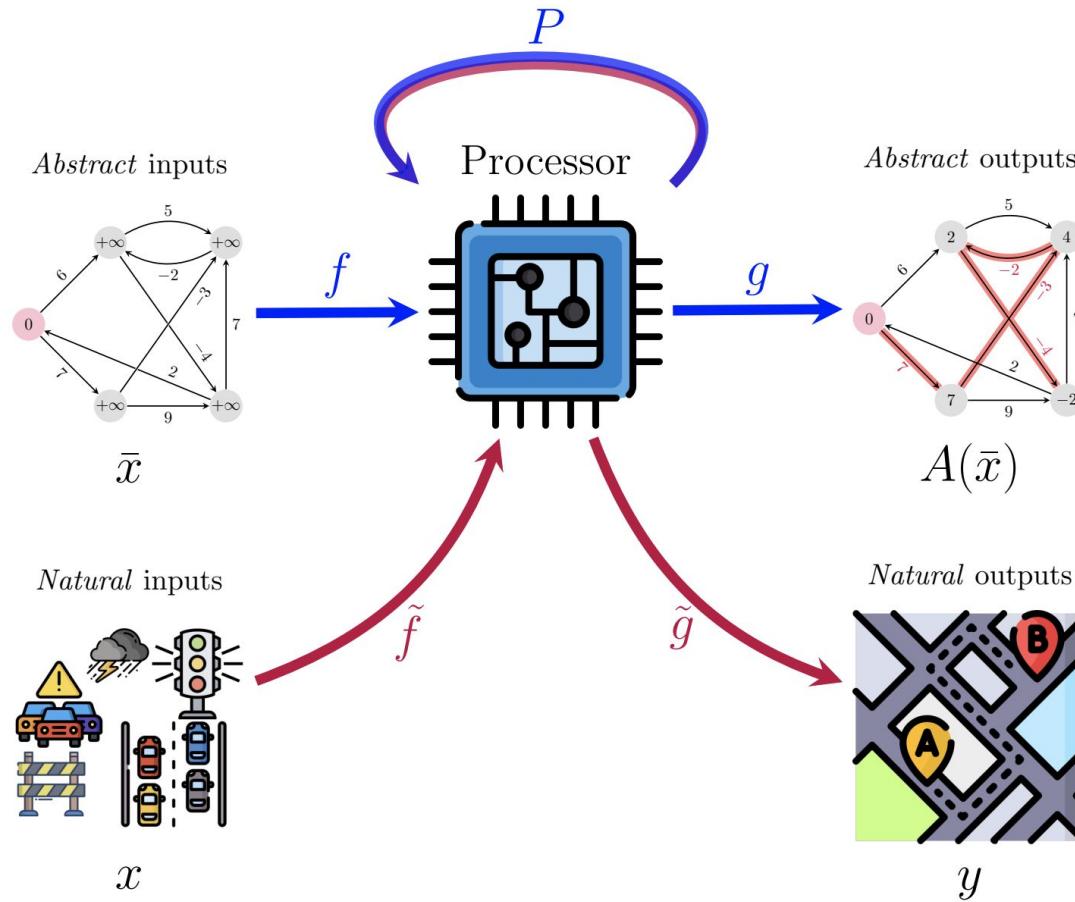
# tl;dr of algorithmic reasoning

- Graph neural networks (GNNs) align well with **dynamic programming** (Xu et al., ICLR'20)
  - Interesting **inductive biases** explored by Veličković et al. (ICLR'20):
    - Encode-**process**-decode from abstract inputs to outputs
    - Favour the **max** aggregation
    - **Strong** supervision on trajectories
  - Further interesting work:
    - IterGNNs (Tang et al., NeurIPS'20)
    - Shuffle-exchange nets (Freivalds et al., NeurIPS'19)
    - PGN (Veličković et al., NeurIPS'20)
    - PMP (Strathmann et al., ICLR'21 SimDL)
  - Latest insights: **linear algorithmic alignment** is highly beneficial (Xu et al., ICLR'21)

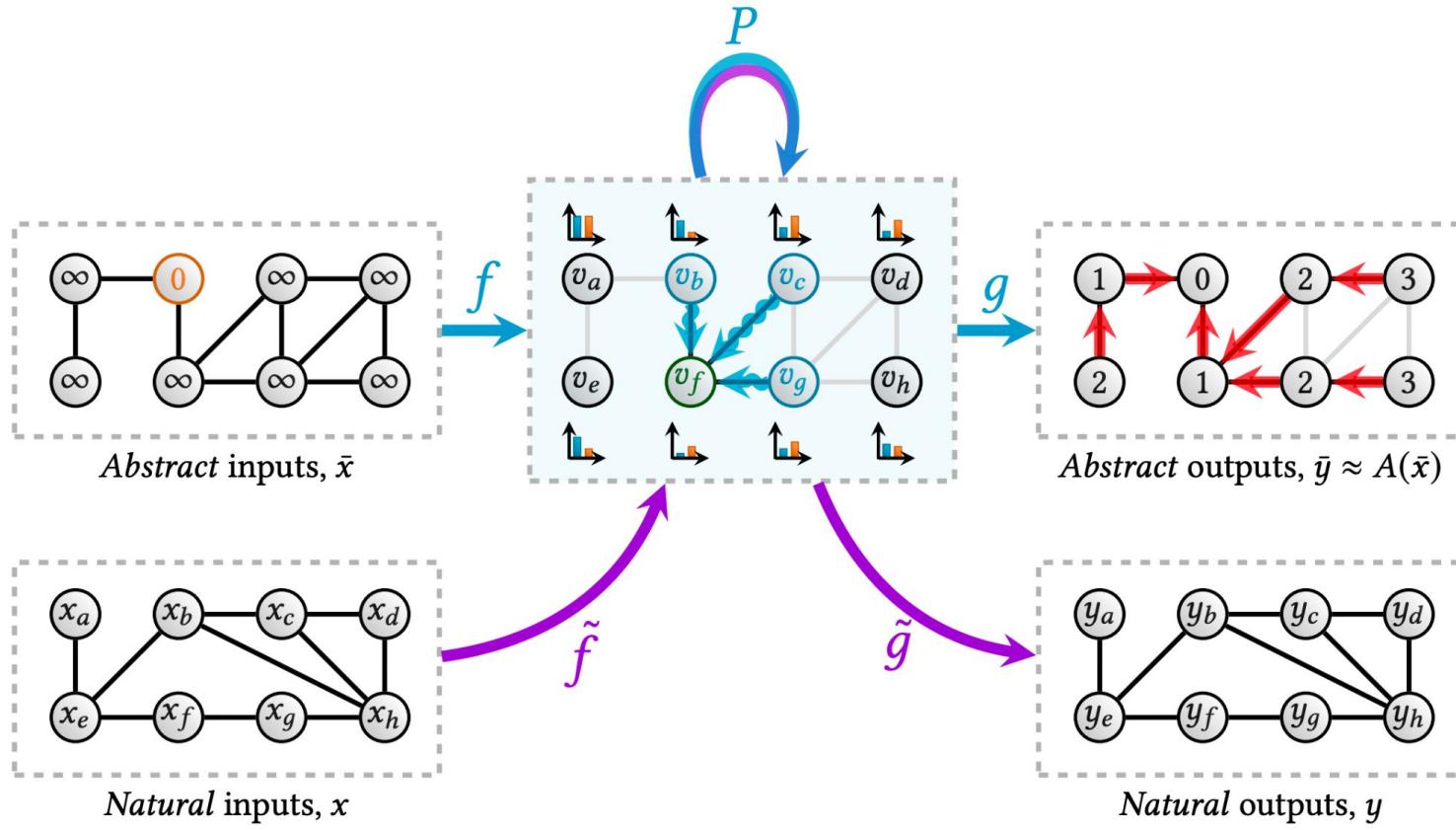
The diagram illustrates the flow of information in a Graph Neural Network (GNN) during training. It shows a graph structure with nodes \$v\_2\$ and \$v\_4\$. A green box labeled \$d\_u = \min\_{v \in N\_u} d\_v + w\_{vu}\$ represents the update rule for node \$v\_2\$, where \$d\_v\$ is highlighted in red and \$w\_{vu}\$ is highlighted in blue. Red arrows show the flow of \$d\_v\$ values from other nodes to \$v\_2\$. Blue arrows show the flow of weights \$w\_{vu}\$ from \$v\_4\$ to \$v\_2\$. A green circle labeled 'max' indicates the aggregation function being used. The labels \$m\_{21}\$, \$m\_{41}\$, and \$m\_{51}\$ are also present, likely referring to message passing or specific edge weights.



# Blueprint of algorithmic reasoning



# Blueprint of algorithmic reasoning, in-depth

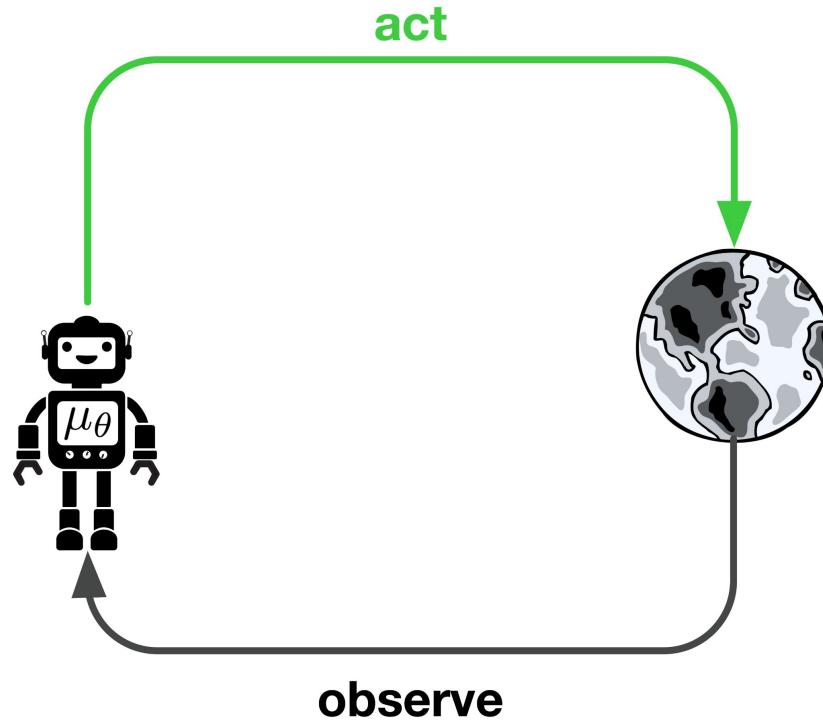


6

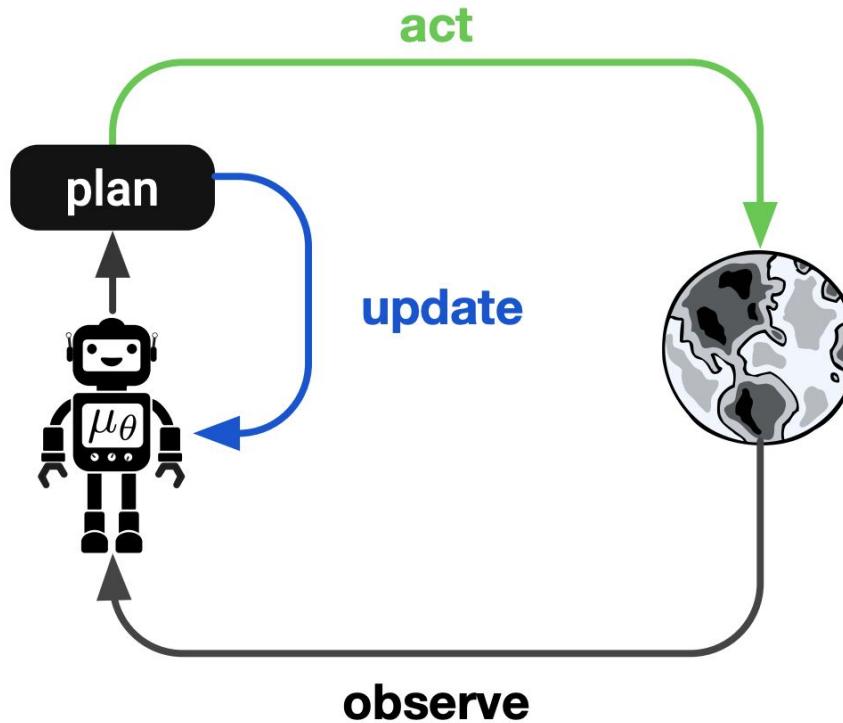
# An algorithmic *implicit planner*



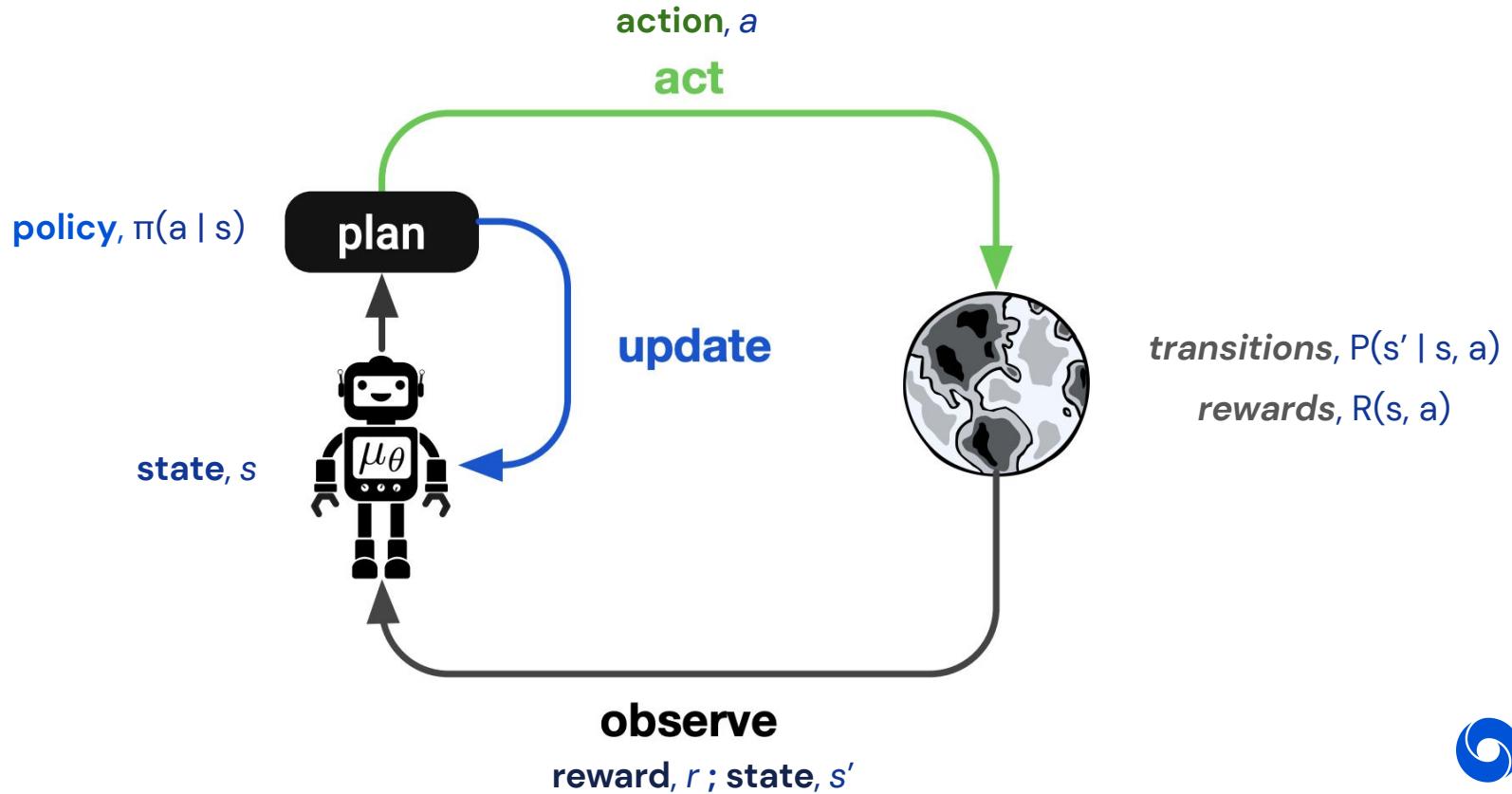
# Reinforcement learning (RL) setting



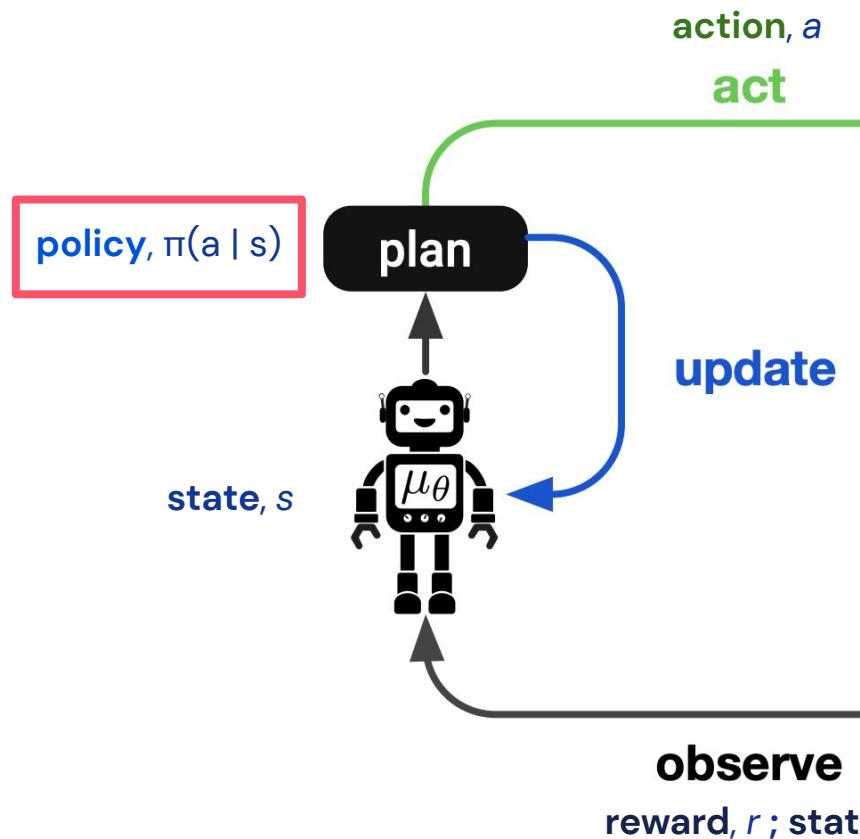
# Reinforcement learning (RL) setting (with *planning*)



# Reinforcement learning (RL) setting (variables)



# Reinforcement learning (RL) setting



Want to optimise:

**Discounted cumulative reward**

$$G = \sum_{t \geq 0} \gamma^t r_t$$

*transitions,  $P(s' | s, a)$   
rewards,  $R(s, a)$*



# Intro to value iteration

- Value Iteration: *dynamic programming* algorithm for **perfectly** solving an RL environment

$$v^{(t+1)}(s) = \max_{a \in \mathcal{A}_s} r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a)v^{(t)}(s')$$

where  $v(s)$  corresponds to the **value** of state  $s$ .

- **Guaranteed** to converge to *optimal* solution (fixed-point of Bellman optimality equation)!

$$V^*(s) = \max_{a \in \mathcal{A}} \left( R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a)V^*(s') \right)$$

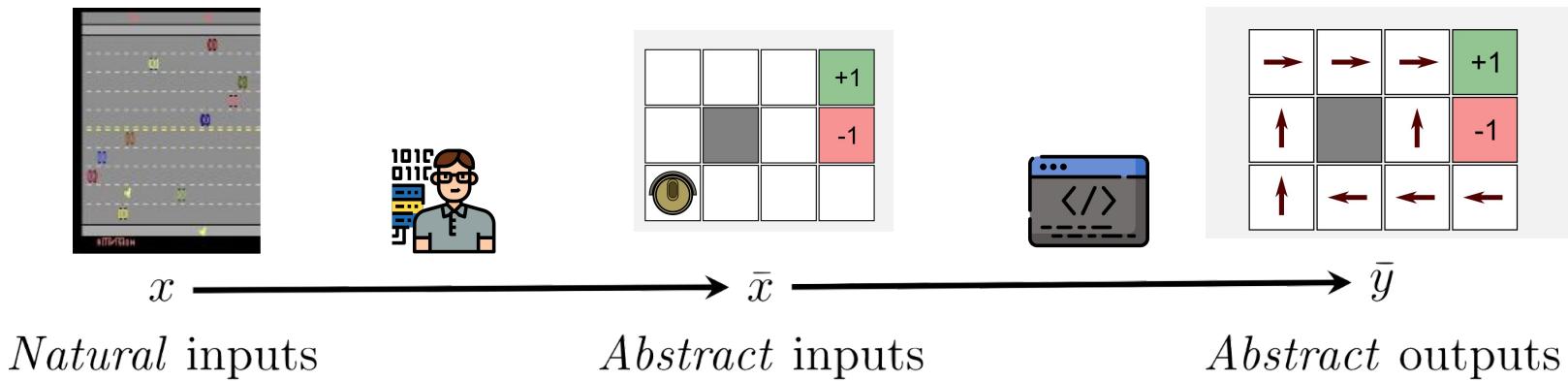
Optimal policy takes actions that **maximise** expected value:  $\operatorname{argmax}_a \sum_{s'} V^*(s') P(s' | s, a)$

- BUT requires **full knowledge** of underlying MDP ( $P / R$ )
  - Prime target for our previously studied blueprint :)



# Algorithmic reasoning over Value Iteration

- How would a human feature engineer make VI applicable?
  - Looking back to our blueprint example...
- As before, we will try to automate away the **manual** feature extraction



# Latent-space transition models

- Assume we have encoded our state (e.g. with a NN) into **embeddings**,  $z(s) \in \mathbb{R}^k$
- To expand a “*local MDP*” we can apply VI over, we can then use a *transition model*,  $T$ 
  - It is then of the form  $T : \mathbb{R}^k \times A \rightarrow \mathbb{R}^k$
  - Optimised such that  $T(z(s), a) \approx z(s')$
- Many popular methods exist for learning  $T$  in the context of *self-supervised learning*
- **Contrastive** learning methods try to discriminate  $(s, a, s')$  from *negative pairs*  $(s, a, s^\sim)$

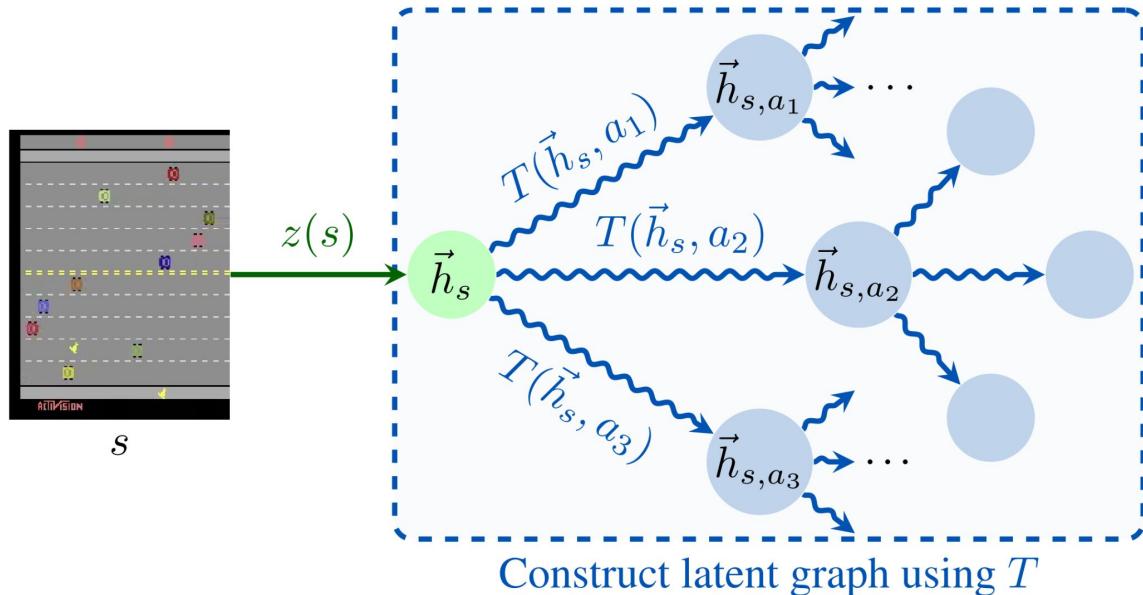


# Using a transition model to expand

We can use a learned transition model on **every** action, to be exhaustive (~breadth-first search)

Doesn't **scale** with large action spaces / thinking times;  $O(|A|^K)$

Can find more interesting *rollout policies*, e.g. by **distilling** well-performing **model-free** ones.



# TreeQN / ATreeC

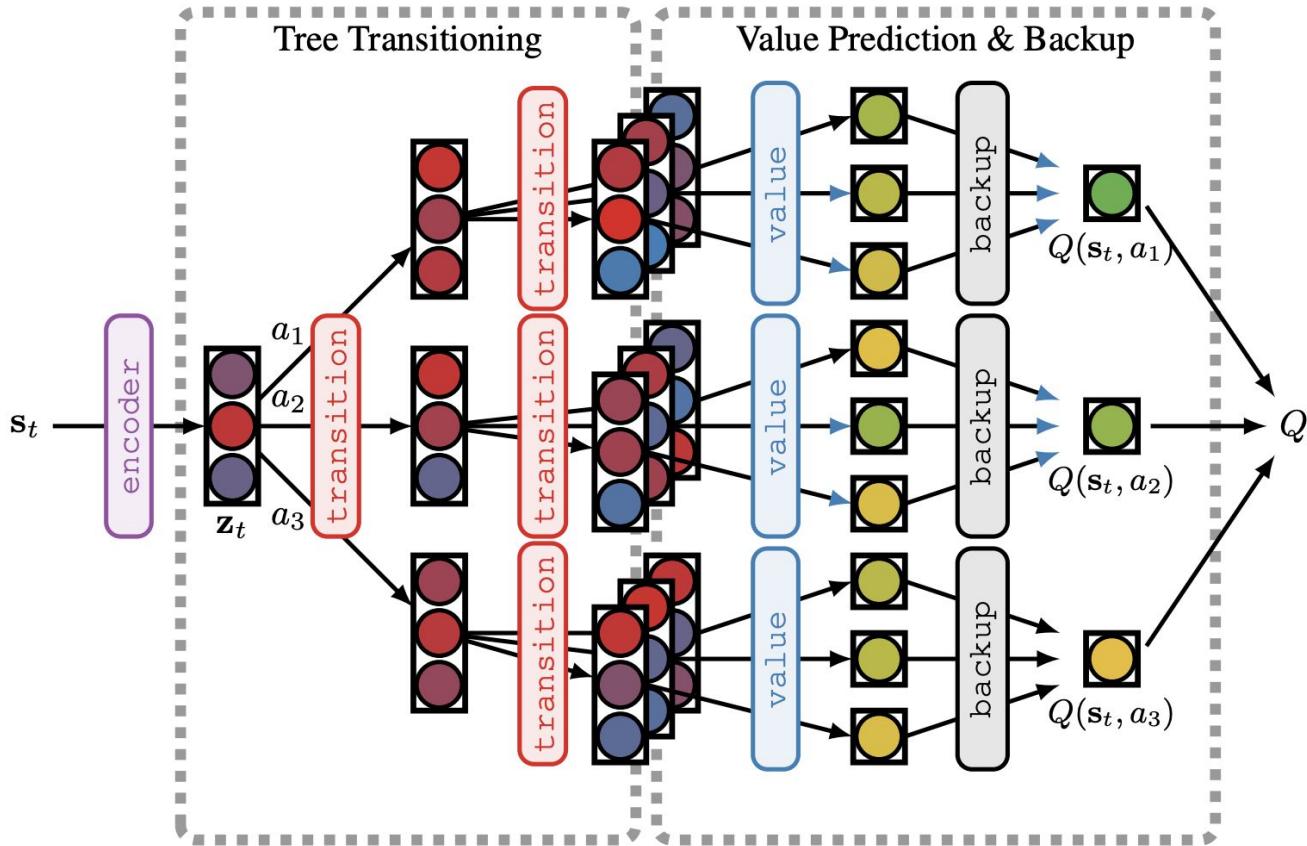
- Assume that we have reward/value models, giving us scalar **values** in every expanded node
- We can now **directly** apply a VI-style update rule!

$$Q(\mathbf{z}_{l|t}, a_i) = r(\mathbf{z}_{l|t}, a_i) + \begin{cases} \gamma V(\mathbf{z}_{d|t}^{a_i}) & l = d - 1 \\ \gamma \max_{a_j} Q(\mathbf{z}_{l+1|t}^{a_i}, a_j) & l < d - 1 \end{cases}$$

- Can then use the computed Q-values **directly** to decide the policy
- Exactly as leveraged by models like TreeQN / ATreeC (Farquhar et al., ICLR'18)
  - Also related: Value Prediction Networks (Oh et al., NeurIPS'17)

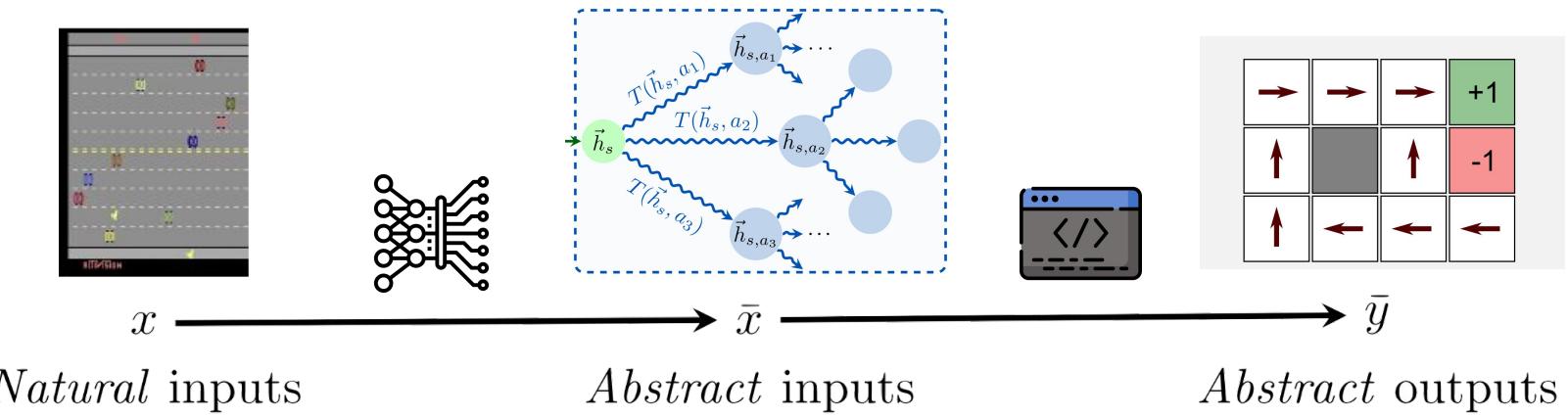


# TreeQN / ATreeC in action



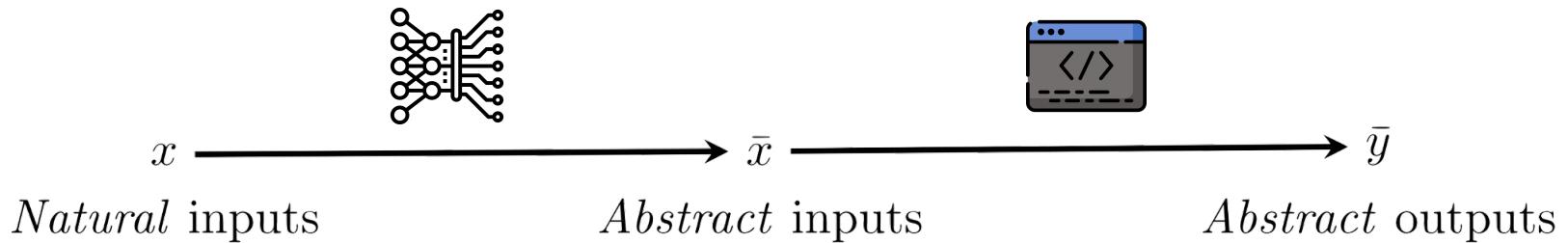
# High-level view

- It's good to take a recap and realise what we have done so far



# High-level view

- It's good to take a recap and realise what we have done so far
  - We mapped our **natural** inputs (e.g. pixels) to the space of abstract inputs
  - (local MDP + reward values in every node)
  - This allowed us to execute VI-style algorithms **directly** on the abstract inputs

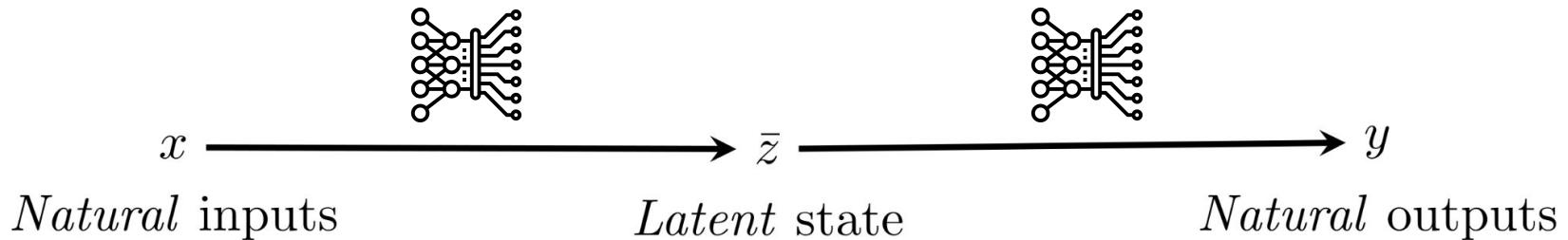


- The VI update is differentiable, and hence so is our entire implicit planner.



# Breaking the bottleneck

- We hit bottleneck-based ***data efficiency*** issues again!
  - If there are insufficient training data to properly estimate the scalars...
  - Algorithm will give a **perfect** solution, but in a ***suboptimal*** environment



- To break the bottleneck, we replace the VI update with a **neural network**!
- As before, we can use **graph neural networks** to perform VI-aligning computations.



# Algorithmic reasoning

- GNN over state representations aligns with VI, but may put **pressure** on the planner
  - Same gradients used to construct correct graphs **and** make VI computations
- To alleviate this issue, we choose to **pre-train** the GNN to perform value iteration-style computations (over many **synthetic** MDPs), then deploying it within our planner
- This exploits, once again, the concept of *algorithmic alignment* (Xu et al., ICLR'20)

$$v^{(t+1)}(s) = \max_{a \in \mathcal{A}_s} r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s, a) v^{(t)}(s') .$$

Diagram illustrating the computation of  $v^{(t+1)}(s)$  and its relation to the state representation  $h_v^{t+1}$  and message  $m_v^{t+1}$ .

The equation shows the update rule for the value function  $v^{(t+1)}(s)$  based on the maximum reward  $r(s, a)$ , discount factor  $\gamma$ , and transition probabilities  $p(s'|s, a)$  leading to the previous value function  $v^{(t)}(s')$ .

The diagram shows the flow of information from the value function update to the state representation and message computation:

- The reward  $r(s, a)$  and transition probability  $p(s'|s, a)$  are highlighted in orange.
- The discount factor  $\gamma$  and summation term are highlighted in green.
- The resulting value  $v^{(t+1)}(s)$  is used to compute the next state representation  $h_v^{t+1}$  and message  $m_v^{t+1}$ .
- The state representation  $h_v^t$  and message  $m_v^{t+1}$  are shown as inputs to the function  $U_t$ .
- The message  $m_v^{t+1}$  is shown as input to the function  $M_t$ .
- The function  $M_t$  also receives inputs  $h_w^t$  and  $e_{vw}$ .

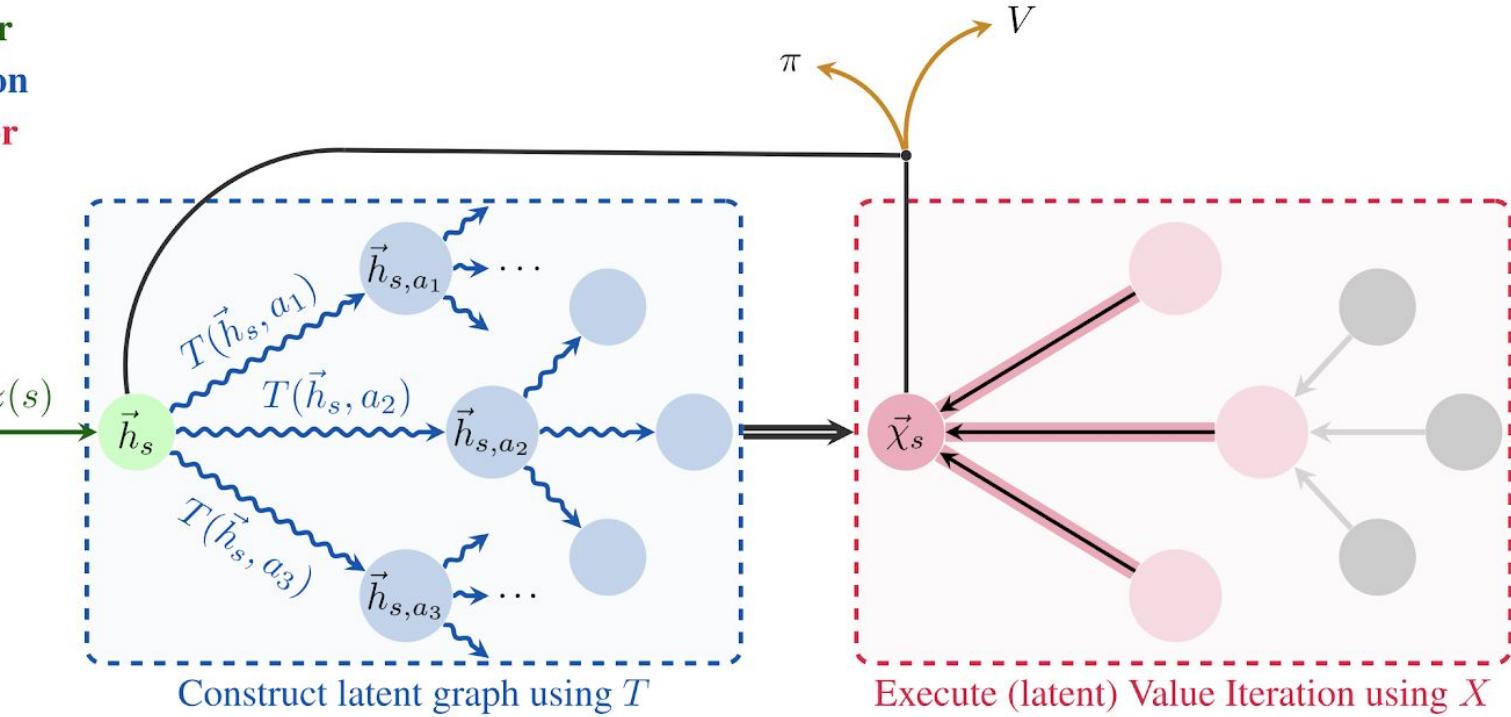


# Putting it all together!

- Encoder
- ~~~~ Transition
- Executor
- Tail



$s$



# XLVIN Components

- **Encoder** ( $z: S \rightarrow \mathbb{R}^k$ ) provides state representations
- **Transition** ( $T: \mathbb{R}^k \times A \rightarrow \mathbb{R}^k$ ) simulates effects of actions in *latent* space
  - Pre-trained & Fine-tuned on the TransE loss (observed trajectories)
- **Executor** ( $X: \mathbb{R}^k \times \mathbb{R}^{|A| \times k} \rightarrow \mathbb{R}^k$ ) simulates a planning algorithm (Value Iteration) in *latent* space
  - Pre-trained to execute VI on synthetic MDPs of interest, then frozen
- **Policy / Value Head**, computing action probabilities and state-values given embeddings
  - Use PPO as the policy gradient method

The entire procedure is end-to-end differentiable, does not impose any assumptions on the structure of the underlying MDP, and has the capacity to perform computations directly aligned with value iteration. Hence our model can be considered as a generalisation of VIN-like methods to settings where the MDP is not provided or otherwise difficult to obtain.



# Results on low-data envs.

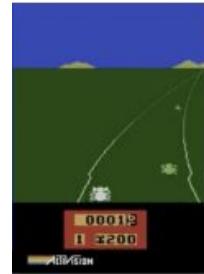
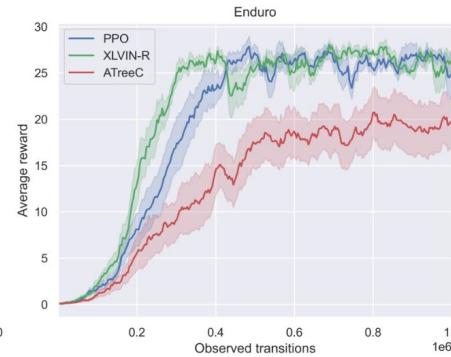
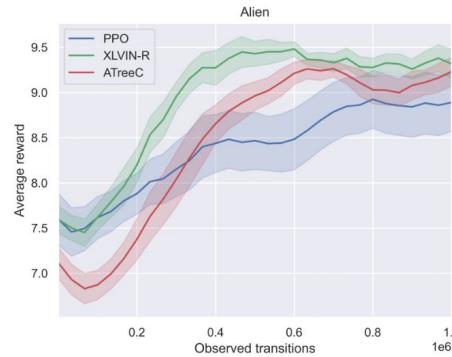
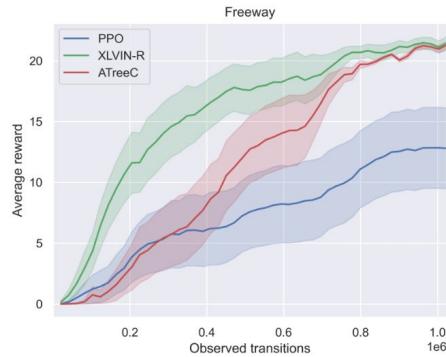


Table 1: Mean scores for low-data CartPole-v0, Acrobot-v1, MountainCar-v0 and LunarLander-v2, averaged over 100 episodes and five seeds.

Agent	CartPole-v0 10 trajectories	Acrobot-v1 100 trajectories	MountainCar-v0 100 trajectories	LunarLander-v2 250 trajectories
PPO	104.6 $\pm$ 48.5	-500.0 $\pm$ 0.0	-200.0 $\pm$ 0.0	90.52 $\pm$ 9.54
ATreeC	117.1 $\pm$ 56.2	-500.0 $\pm$ 0.0	-200.0 $\pm$ 0.0	84.04 $\pm$ 5.35
XLVIN-R	<b>199.2</b> $\pm$ 1.6	-353.1 $\pm$ 120.3	-185.6 $\pm$ 8.1	<b>99.34</b> $\pm$ 6.77
XLVIN-CP	<b>195.2</b> $\pm$ 5.0	<b>-245.4</b> $\pm$ 48.4	<b>-168.9</b> $\pm$ 24.7	N/A



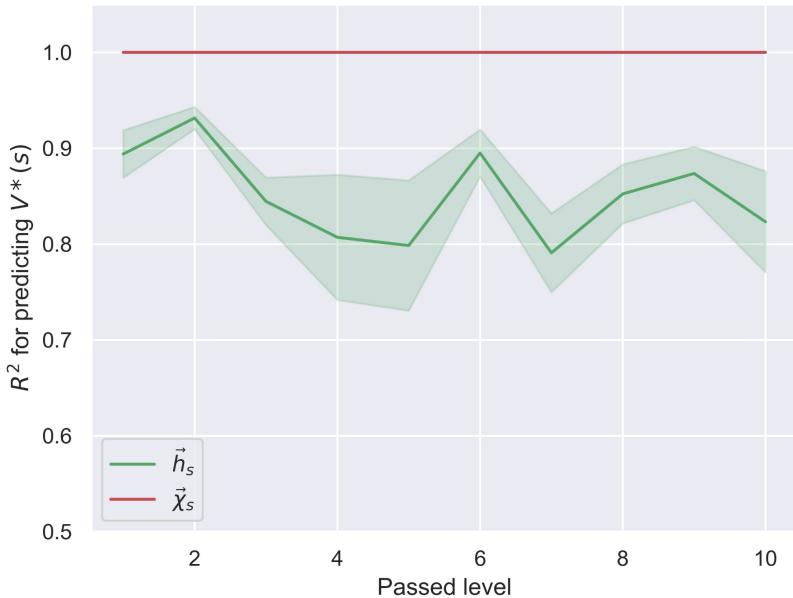
# ...why did it work?

- Recall, our executor network was pre-trained and **frozen**
- The pixel-level encoder needed to learn to map **rich** inputs into the executor's latent space
  - Analogous to a human who tries to map real-world problems to algorithmic inputs!
- We set out to investigate to what extent it succeeded.



# Grid-world qualitative study

- We evaluate the quality of the embeddings **before** and **after** applying the executor, in a *grid-world* environment
  - Here we can compute optimal  $V^*(s)$
  - Evaluate linear decodability by linear regression!
- Results verify our hypothesis!
  - Input values are already predictive
  - But the executor consistently  
**refines** them!
- Our encoder learnt to correctly *map* the input to the latent algorithm! :)



7

# Summary and conclusions



# Overview, revisited

Our aim ~~is~~ was to address **three** key questions: (roughly ~10min for each)

- Why should we, as deep learning practitioners, study **algorithms**?
  - Further, why might it be beneficial to make '*algorithm-inspired*' neural networks?
- How to **build** neural networks that behave algorithmically?
  - And why am I even telling you this in a "*Graph Machine Learning*" context?
- Do algorithmic neural networks actually **work** when deployed?
  - If so, how are they *actually* being used?

Hopefully, also some ideas on **where** you might be able to **apply** the ideas above :)



# Further insight: Algorithmic reasoning

If you would like to know more details about constructing good processor networks:



<https://www.youtube.com/watch?v=IPO6CPoluok>

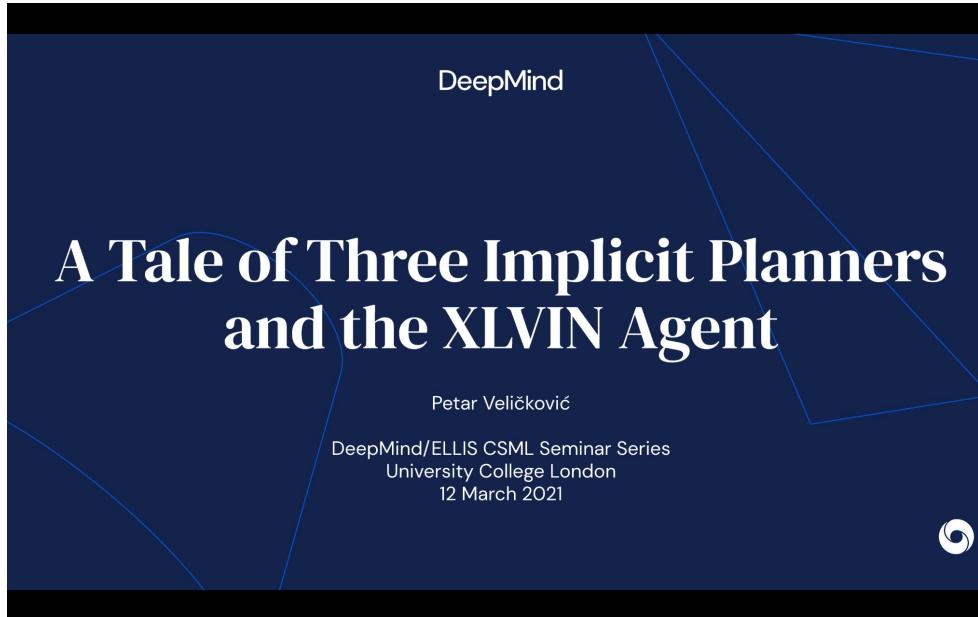


[https://drive.google.com/file/d/1\\_EQ9Yu7VEkvrHaVH1\\_WbT5ABvxrSNY-s/view?usp=sharing](https://drive.google.com/file/d/1_EQ9Yu7VEkvrHaVH1_WbT5ABvxrSNY-s/view?usp=sharing)



# Further insight: Algorithmic implicit planning

If you would like to know more details about implicit planning and XLVIN:



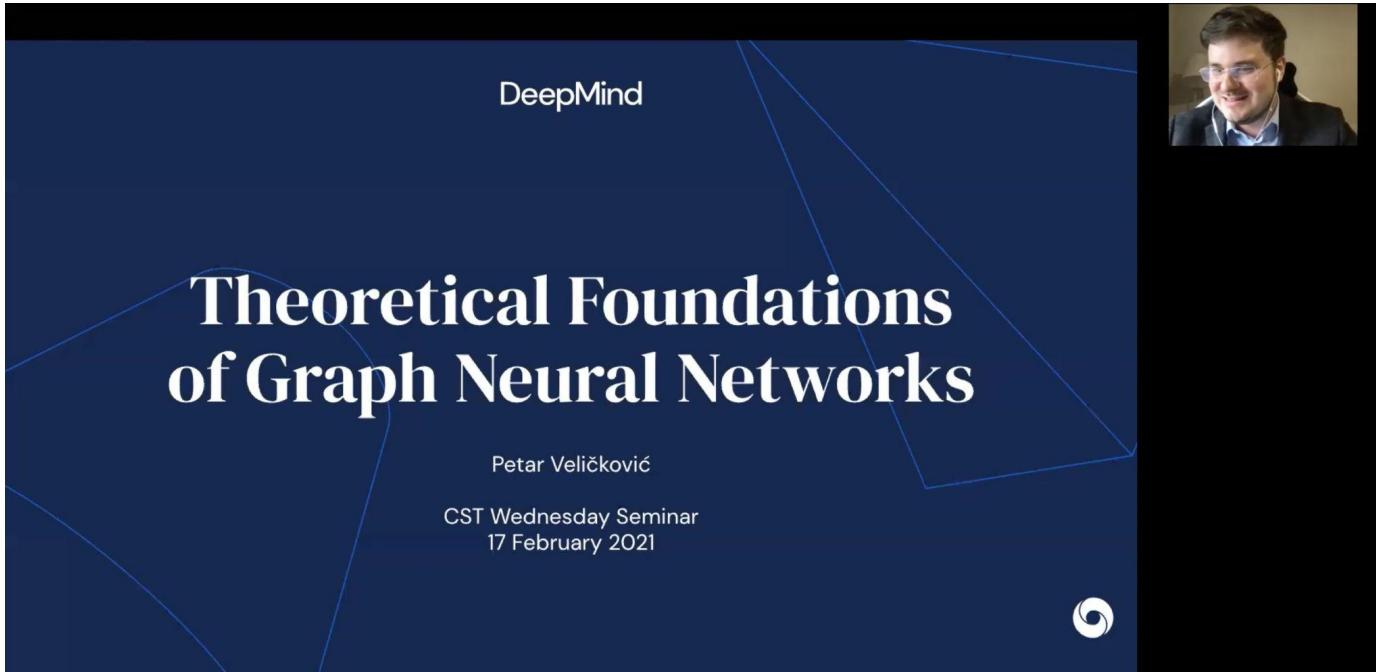
<https://www.youtube.com/watch?v=mGw9ewL8wCU>



# Further insight: graph representation learning

If GNNs are new(ish) to you, I recently gave a useful talk on **theoretical GNN foundations**:

<https://www.youtube.com/watch?v=uF53xsT7mjc>



# Want to know more?

## Combinatorial optimization and reasoning with graph neural networks

Quentin Cappart<sup>1</sup>, Didier Chételat<sup>2</sup>, Elias Khalil<sup>3</sup>, Andrea Lodi<sup>2</sup>,  
Christopher Morris<sup>2</sup>, and Petar Veličković<sup>\*4</sup>

<sup>1</sup>Department of Computer Engineering and Software Engineering, Polytechnique Montréal

<sup>2</sup>CERC in Data Science for Real-Time Decision-Making, Polytechnique Montréal

<sup>3</sup>Department of Mechanical & Industrial Engineering, University of Toronto

<sup>4</sup>DeepMind

Our 43-page survey on GNNs for CO!

<https://arxiv.org/abs/2102.09544>

**Section 3.3.** details algorithmic reasoning,  
with comprehensive references.

Combinatorial optimization is a well-established area in operations research and computer science. Until recently, its methods have focused on solving problem instances in isolation, ignoring the fact that they often stem from related data distributions in practice. However, recent years have seen a surge of interest in using machine learning, especially graph neural networks (GNNs), as a key building block for combinatorial tasks, either as solvers or as helper functions. GNNs are an inductive bias that effectively encodes combinatorial and relational input due to their permutation-invariance and sparsity awareness. This paper presents a conceptual review of recent key advancements in this emerging field, aiming at both the optimization and machine learning researcher.



DeepMind

# Thank you!

[petarv@google.com](mailto:petarv@google.com) | <https://petar-v.com>

In collaboration with Charles Blundell, Raia Hadsell, Rex Ying, Matilde Padovano,  
Andreea Deac, Ognjen Milinković, Pierre-Luc Bacon, Jian Tang, Mladen Nikolić,  
Christopher Morris, Quentin Cappart, Elias Khalil, Didier Chétalat, Andrea Lodi,  
Lovro Vrček, Mile Šikić, Lars Buesing, Matt Overlan, Razvan Pascanu and Oriol Vinyals

