DeepMind

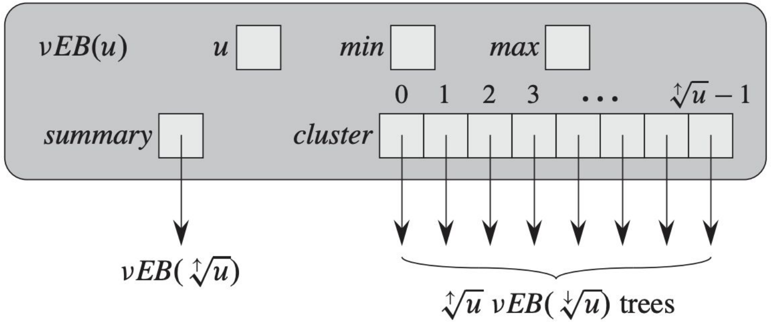
Enabling Empirically & Theoretically Sound Algorithmic Alignment

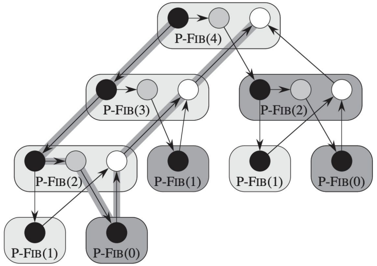
Petar Veličković

Learning to Discover
20 April 2022

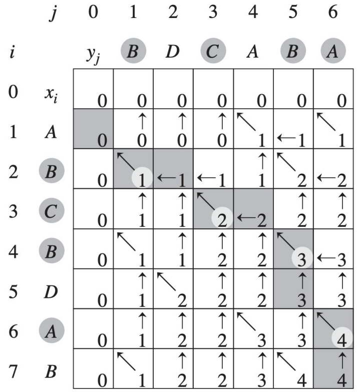
In this talk:
(Classical) **Algorithms**

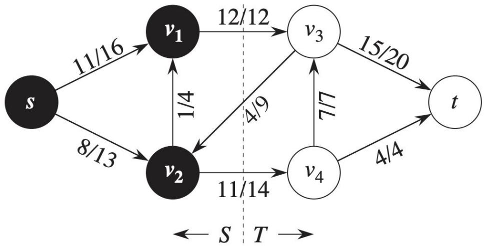
```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

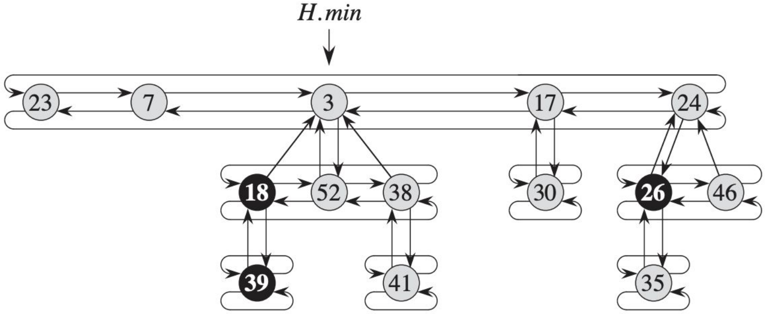


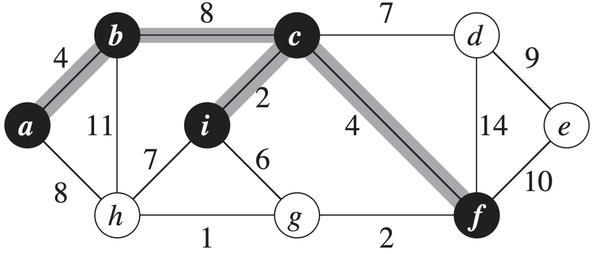


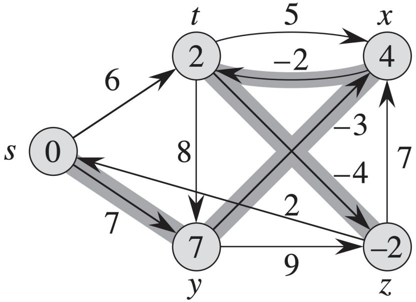
In this talk:
(Classical) **Algorithms**



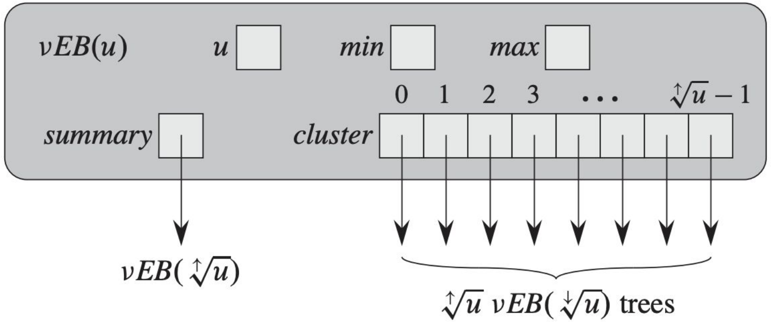




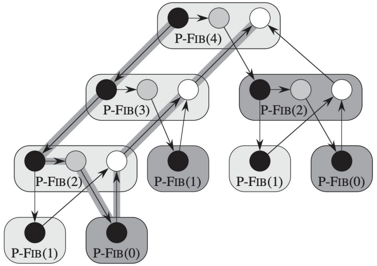




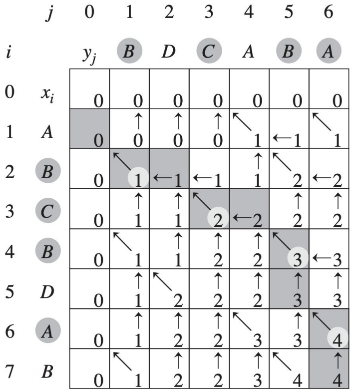
Algorithm figures: Cormen, Leiserson, Rivest and Stein. **Introduction to Algorithms.**

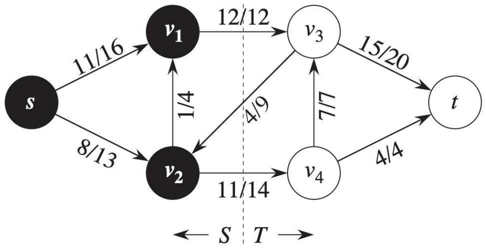


DeepMind




```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

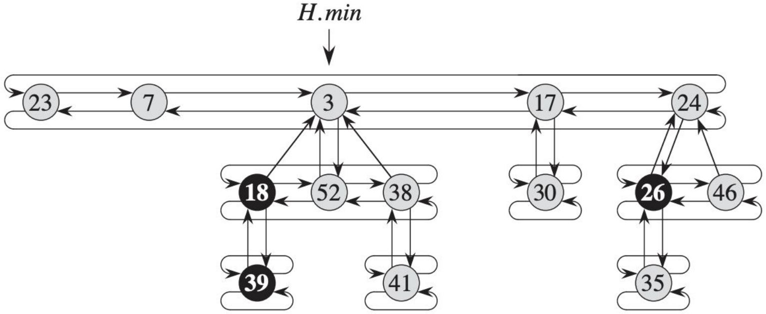


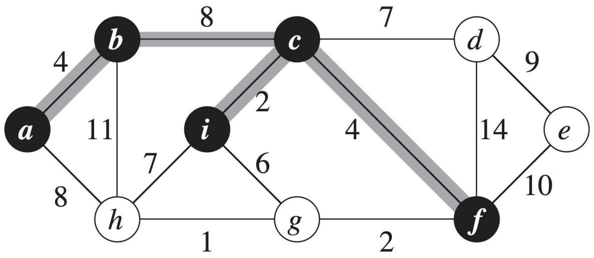


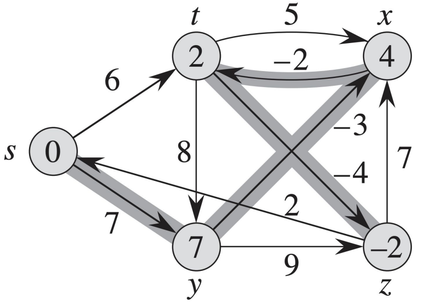
In this talk:

(Classical) **Algorithms**

(with a bit of neural spice)







Algorithm figures: Cormen, Leiserson, Rivest and Stein. **Introduction to Algorithms.**

Overview

Our aim is to address **three** key questions: (roughly ~15min 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 are there any libraries or resources to facilitate this?
- Are there ways to strengthen the **alignment** of GNNs and to algorithms?
  - Explore the fascinating connection between GNNs and dynamic programming

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

DeepMind

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

DeepMind

2

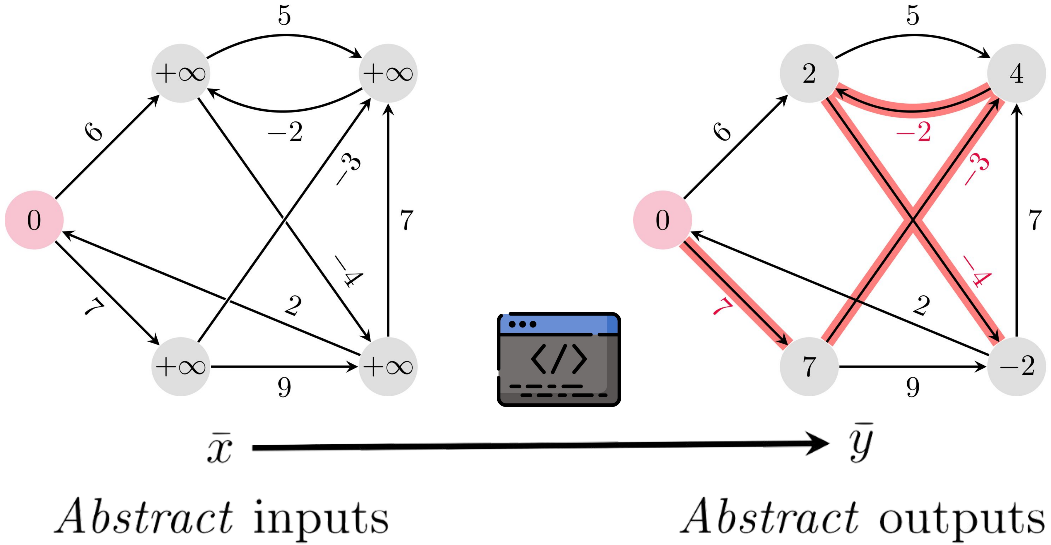
Applying algorithms in the wild

A simple example

- “Find the **optimal path** from A to B”

A simple example

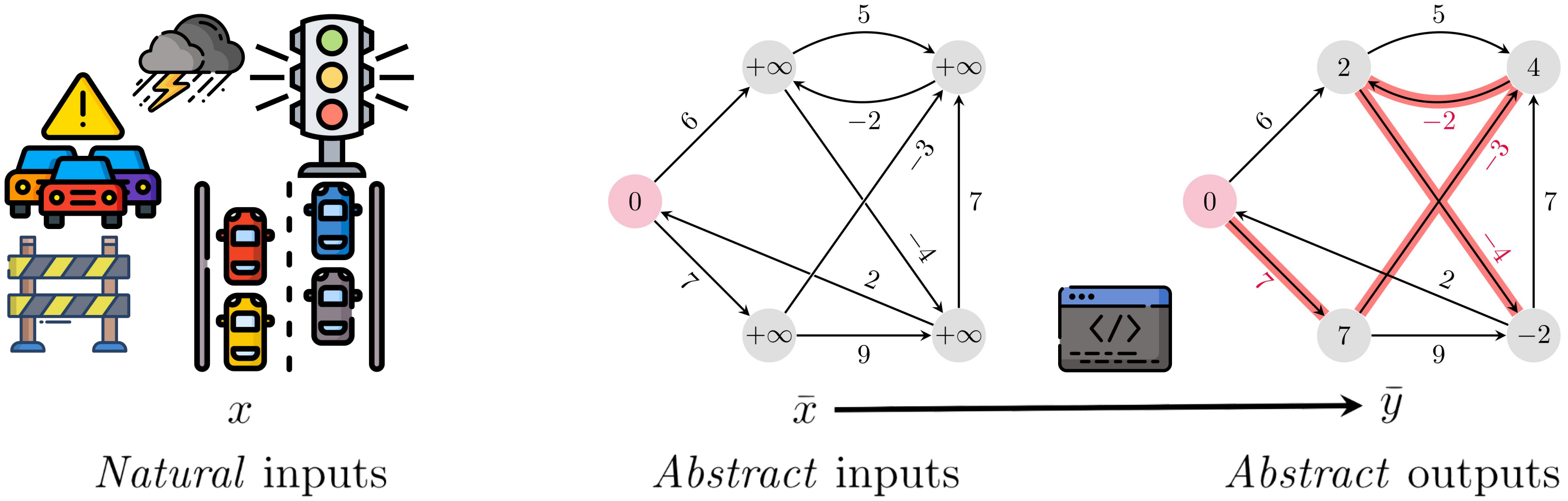
- "Find the **optimal path** from A to B"
    - *The theoretical computer scientist diligently uses the Dijkstra hammer!*



What did the theoretical computer scientist subtly assume?

A simple example

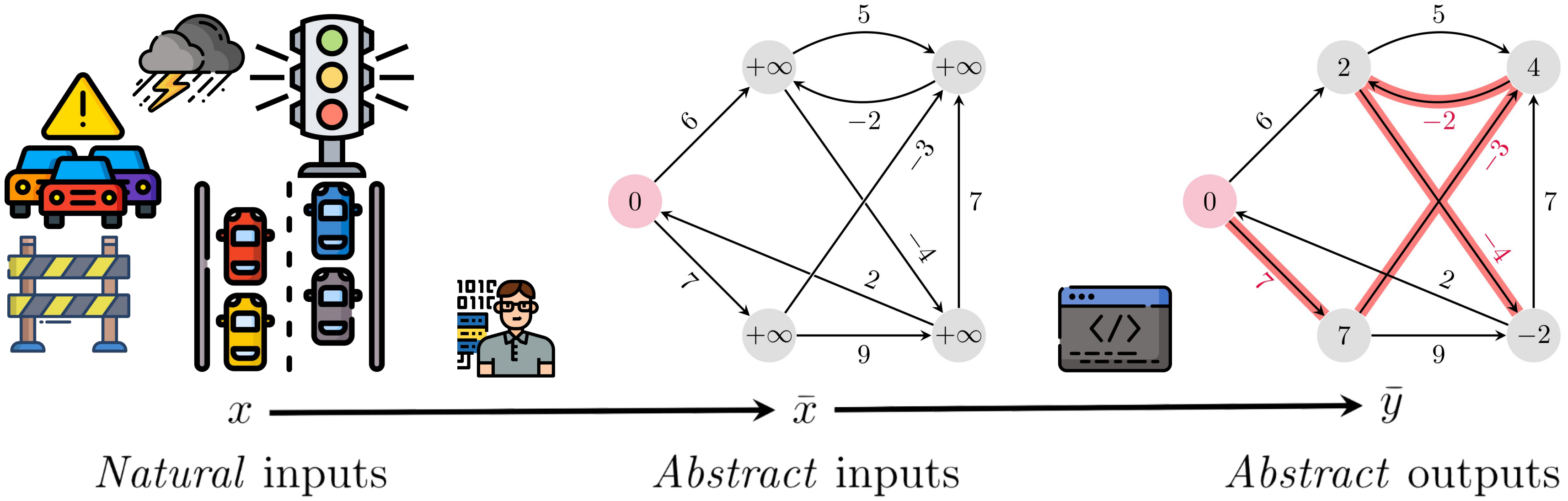
- "Find the **optimal path** from A to B"
  - *This kind of question usually hides the real-world problem underneath…*



*Also, what do we even mean by **optimal**? Are we sure we mean shortest? Is Dijkstra **enough**?*

A simple example

- Let's ignore the multiple-algorithms problem for now, and assume optimal == shortest.
  - Can we ever hope to manually / heuristically do the mapping necessary?



Not really… (known since at least 1955)

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. 37

This material contains information affecting the national defense of the United States within the meaning of the espionage laws, Title 18 U.S.C., Secs. 793 and 794, the transmission or the revelation of which in any manner to an unauthorized person is prohibited by law

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

- This “core problem” plagues applications of classical combinatorial algorithms to this day!

- If we manually satisfy algorithm preconditions, 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.

- Typically ignored in the theoretical CS community
  - But of high interest for *both* combinatorial and operations research communities.

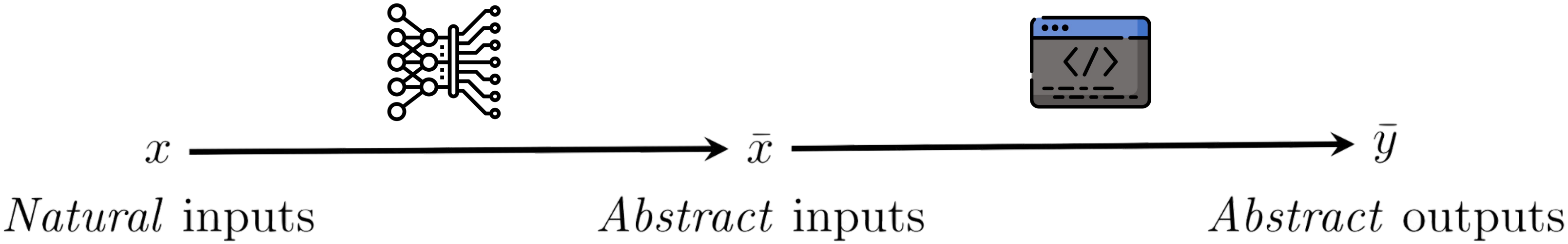
- Here we will attack the core problem by **neuralising** the algorithm

3 Neuralising an algorithm

Attacking the core problem

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

- 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.
  - But there exist great proposals for solving this (e.g. Vlastelica *et al.*)

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**!

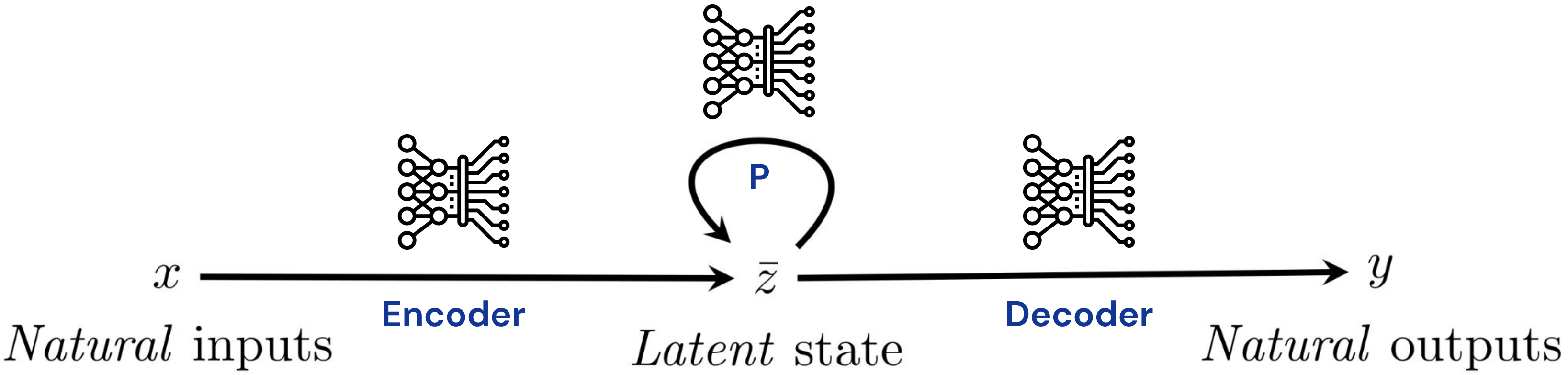
- If there are insufficient training data to properly estimate the scalars, we hit same issues!
    - Algorithm will give a **perfect** solution, but in a ***suboptimal*** environment

- *(**Third** issue: what if the algorithm doesn't give us the entire solution?)*

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**!

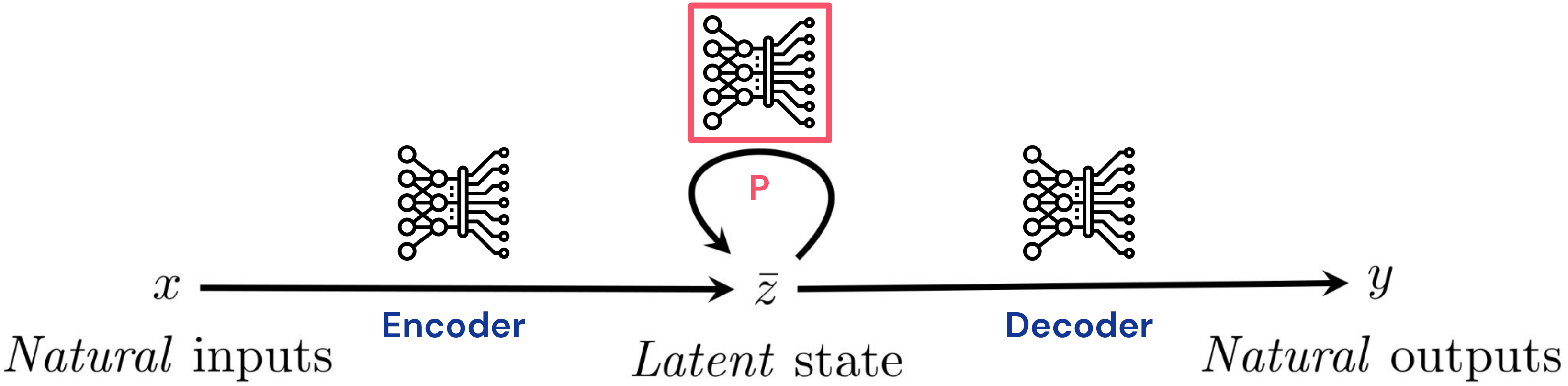


(The setting naturally aligns with *encode-**process**-decode* (Hamrick *et al.*, CSS'18))

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.
  - We can add **skip connections**, if the algorithm is not the whole answer.

- How do we obtain latent-state neural networks that **align** with algorithms?



DeepMind

4 Algorithmic reasoning and the CLRS Benchmark

Algorithmic reasoning

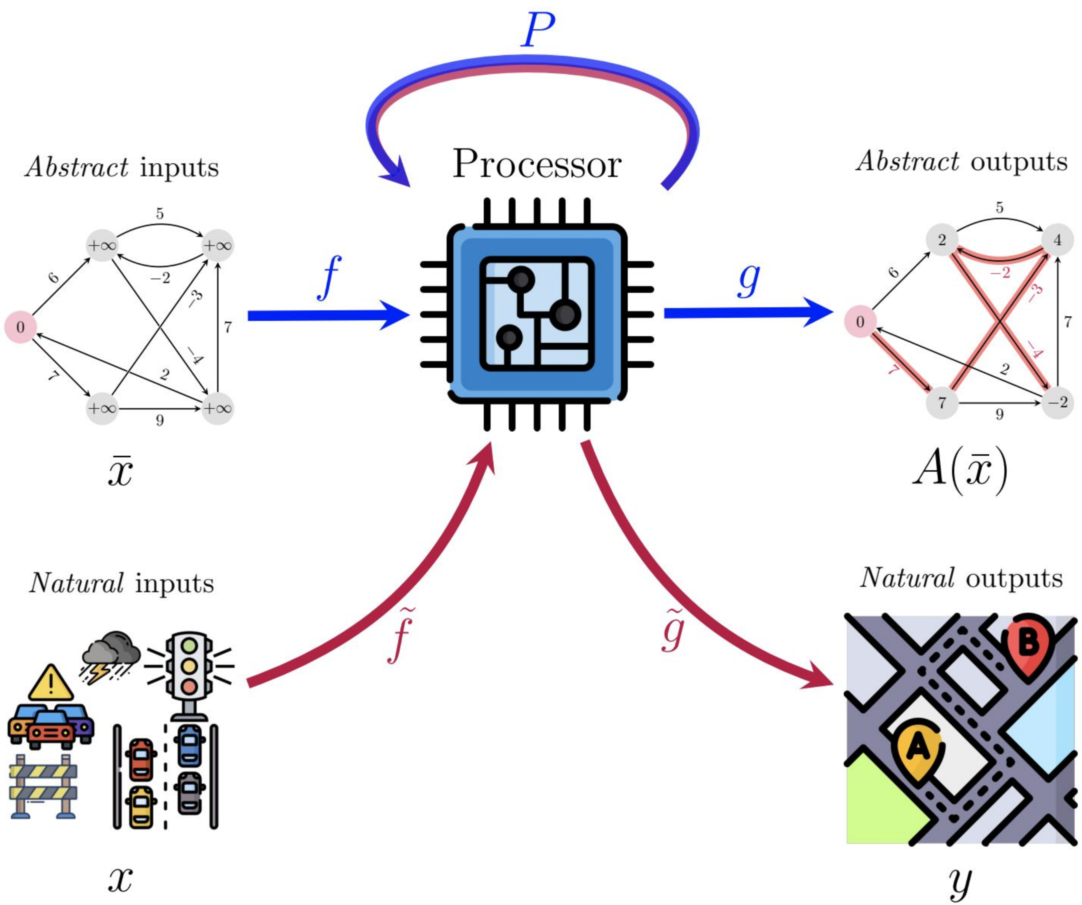
- The desiderata for our processor network **P** are slightly different than usual:
  - They are required to imitate the steps of the algorithm *faithfully*
  - This means they must **extrapolate**!
  - (*Related*: *how to best decide the **weights** of **P** to **robustly** match the algorithm?*)

- 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 <u>*architectures*</u>, <u>*inductive biases*</u> and <u>*featurisations*</u> that are useful for extrapolating combinatorially

- This is a **very** active research area, with many key papers published only last year!

Blueprint of algorithmic reasoning (Veličković & Blundell, Patterns'21)

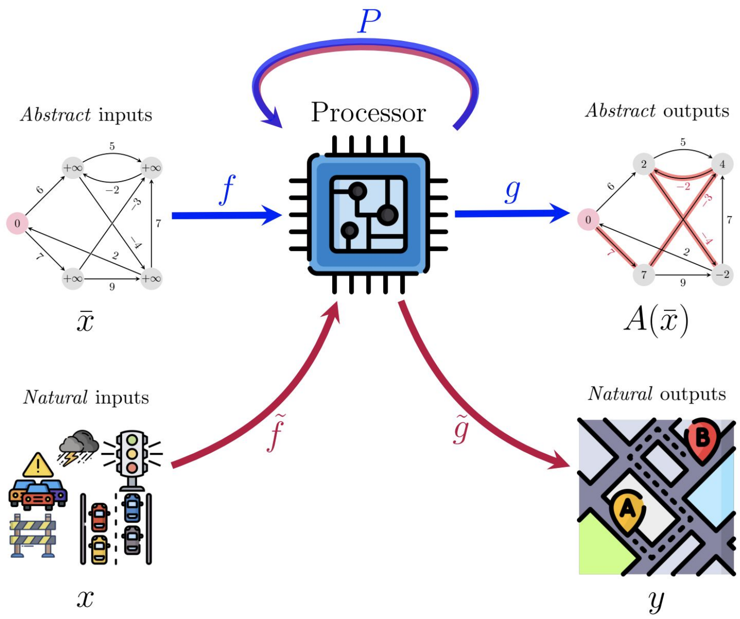


Blueprint of algorithmic reasoning (Veličković & Blundell, Patterns'21)

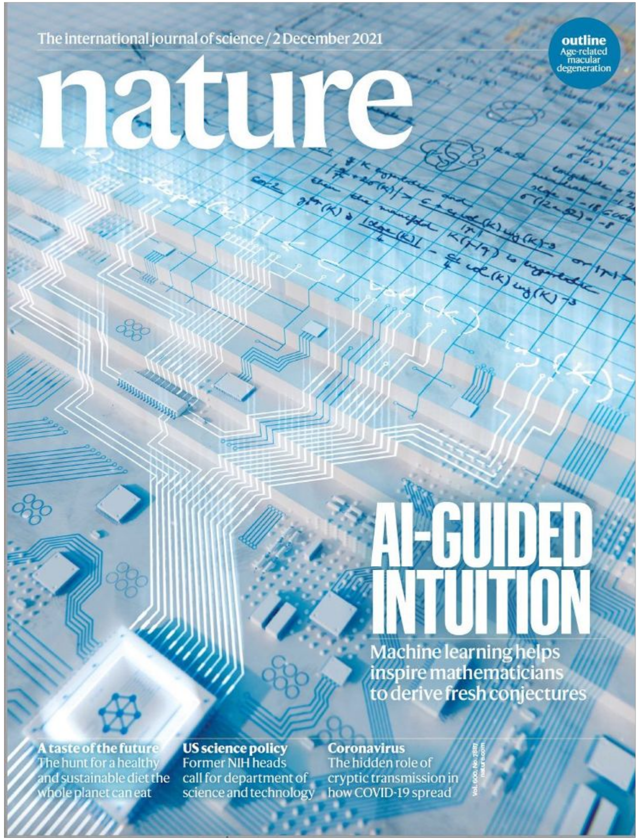
- GNNs **pre-trained** on algorithm, then **deployed** on natural task!

- *"...running classical algorithms on inputs previously considered inaccessible to them"*

- **Proofs-of-concept** exist!
  - XLVIN (Deac *et al.*, NeurIPS'21)
  - RMR (Veličković, Bošnjak *et al.*, 2021)
  - CNAP (He *et al.*, 2022)



Recently also on the cover of Nature!



Model and training procedure

For modelling the Bruhat intervals, we used a particular GraphNet architecture called a message-passing neural network (MPNN)[48]. The design of the model architecture (in terms of activation functions and directionality) was motivated by the algorithms for computing KL polynomials from labelled Bruhat intervals. While labelled Bruhat intervals contain privileged information, these algorithms hinted at the kind of computation that may be useful for computing KL polynomial coefficients. Accordingly, we designed our MPNN to algorithmically align to this computation[49]. The model is bi-directional, with a hidden layer width of 128, four propagation steps and skip connections. We treat the prediction of each coefficient of the KL polynomial as a separate classification problem.

NEWSLETTERS
Sign up to read our regular email newsletters

NewScientist

News Podcasts Video Technology Space Physics Health More Shop Courses Events

DeepMind AI collaborates with humans on two mathematical breakthroughs

Humans and AI working together can reveal new areas of mathematics where data sets are too large to be comprehended by mathematicians

TECHNOLOGY 1 December 2021

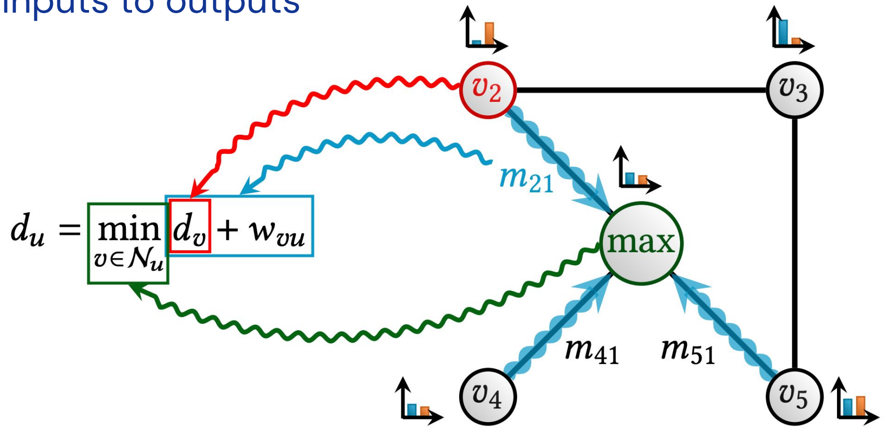
By Matthew Sparkes

49. Veličković, P., Ying, R., Padovano, M., Hadsell, R. & Blundell, C. Neural execution of graph algorithms. Preprint at https://arxiv.org/abs/1910.10593 (2019).

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)
  - PGN (Veličković *et al.*, NeurIPS'20)
  - PMP (Strathmann *et al.*, ICLR'21 SimDL)

- ***Linear*** *algorithmic alignment* is highly beneficial (Xu *et al.*, ICLR'21)
- Properly handling extrapolation may necessitate **causality** (Bevilacqua *et al.*, ICML'21)

DeepMind

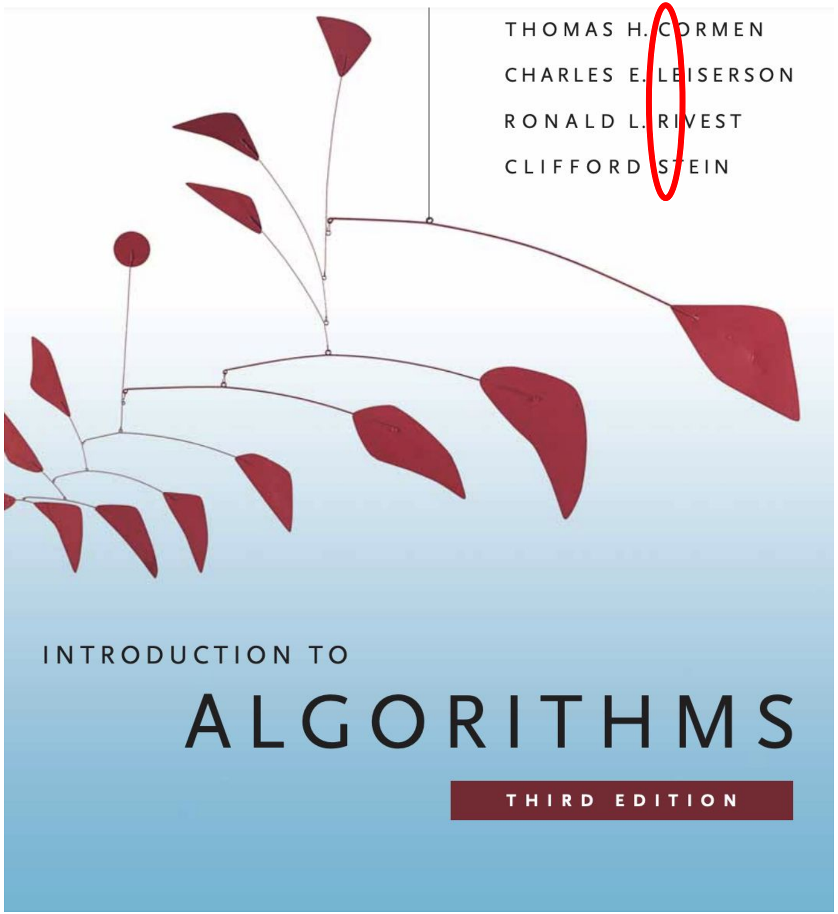
Critically:
Every paper generates its **own dataset**,
making **progress tracking** a *nightmare*

Our inspiration...

Textbook for algorithms at many universities

Summarises the wealth of knowledge of many professional computer scientists

Only about ~90 distinct algorithms
(which we initially reduced to 30)



https://github.com/deepmind/clrs

The CLRS Algorithmic Reasoning Benchmark

Petar Veličković
DeepMind
petarv@deepmind.com

Adrià Puigdomènech Badia
DeepMind
adriap@deepmind.com

David Budden
DeepMind
budden@deepmind.com

Razvan Pascanu
DeepMind
razp@deepmind.com

Andrea Banino
DeepMind
abanino@deepmind.com

Misha Dashevskiy
DeepMind
dashevskiy@deepmind.com

Raia Hadsell
DeepMind
raia@deepmind.com

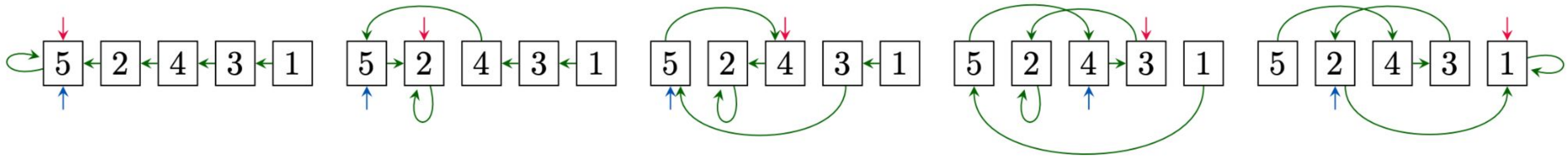
Charles Blundell
DeepMind
cblundell@deepmind.com

Paper to be released on arXiv soon!

Representation

- Generators provide inputs which fully specify the input, output (and trajectory)
    - Trajectory can tell apart many different algos that implement the **same** function

- For convenience, provided pre-processors convert it into **graphs**
    - Node / edge / graph-level **inputs, hints**, and **targets.**



Restrictions

- No ambiguity in evaluation (no numerical algorithms output)
- No ambiguity in representation (no auxiliary memory tasks)
- No approximation or NP-hard problems

→ This converted the 90 candidates into a final set of 30 algorithms

Tasks!

Sorting: Insertion sort, bubble sort, heapsort (Williams, 1964), quicksort (Hoare, 1962).

Searching: Minimum, binary search, quickselect (Hoare, 1961).

Divide and Conquer (D&C): Maximum subarray (Kadane's variant (Bentley, 1984)).

Greedy: Activity selection (Gavril, 1972), task scheduling (Lawler, 1985).

Dynamic Programming: Matrix chain multiplication, longest common subsequence, optimal binary search tree (Aho et al., 1974).

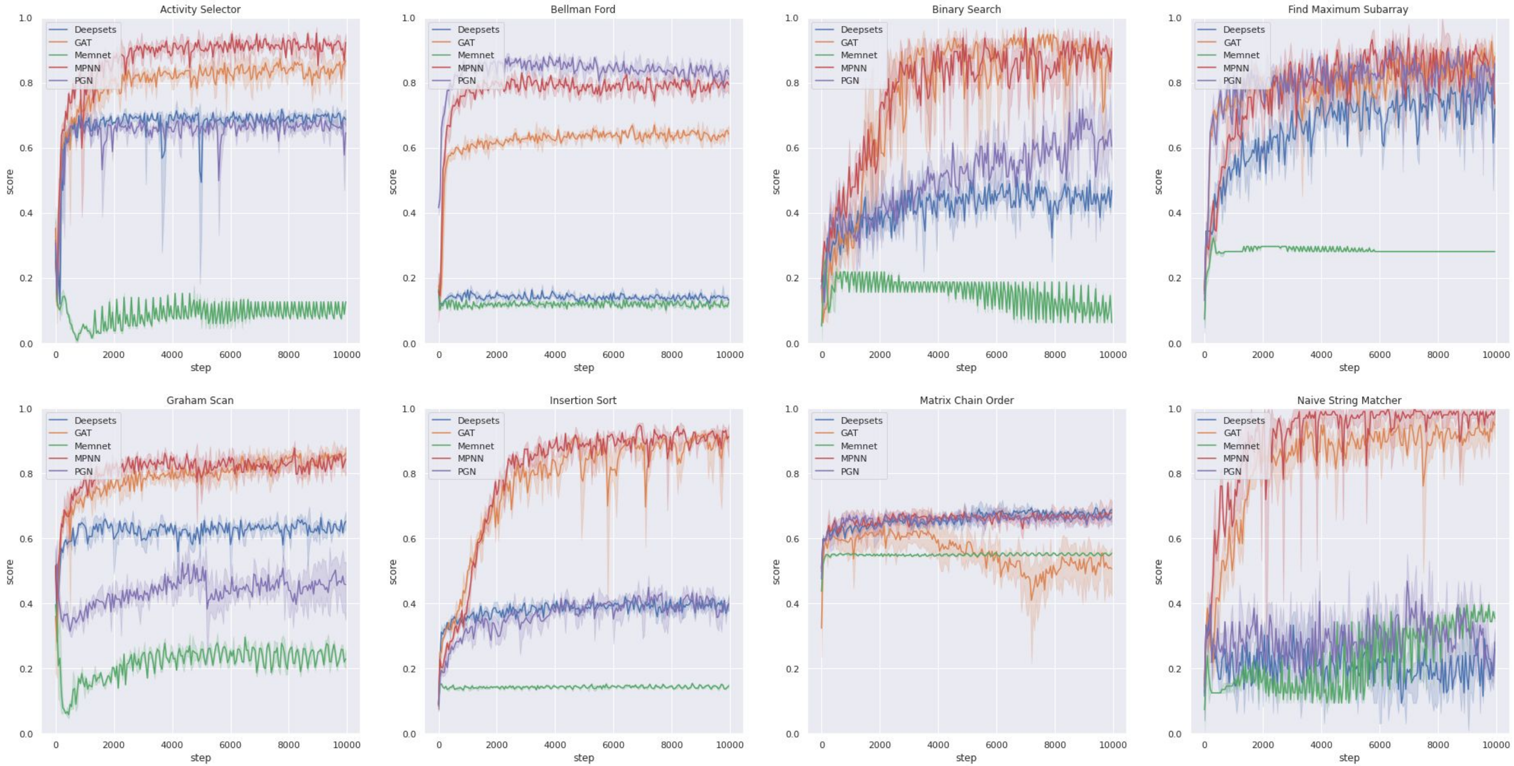
Graphs: Depth-first and breadth-first search (Moore, 1959), topological sorting (Knuth, 1973), articulation points, bridges, Kosaraju's strongly-connected components algorithm (Aho et al., 1974), Kruskal's and Prim's algorithms for minimum spanning trees (Kruskal, 1956; Prim, 1957), Bellman-Ford and Dijkstra's algorithms for single-source shortest paths (Bellman, 1958; Dijkstra et al., 1959) (+ directed acyclic graphs version), Floyd-Warshall algorithm for all-pairs shortest paths (Floyd, 1962).

Strings: Naïve string matching, Knuth-Morris-Pratt (KMP) string matcher (Knuth et al., 1977).

Geometry: Segment intersection, Convex hull algorithms: Graham scan (Graham, 1972), Jarvis' march (Jarvis, 1973).

Performance in-distribution… (8 tasks out of 30)

It might seem as if our models (MPNN in red) can do quite well…



Performance out-of-distribution (4x larger)

Still a long way to go! Hence CLRS is hopefully a useful source of measuring progress :)

Table 1. Average test micro-F_1 score of all models on all algorithm classes.

Algorithm	**DeepSets**	**GAT**	**Memnet**	**MPNN**	**PGN**
Divide & Conquer	$8.85\% \pm 1.13$	$20.31\% \pm 5.56$	$13.02\% \pm 0.43$	$6.25\% \pm 2.21$	$\mathbf{57.81\% \pm 3.21}$
Dynamic programming	$\mathbf{57.68\% \pm 0.62}$	$55.03\% \pm 1.58$	$54.28\% \pm 1.57$	$56.14\% \pm 0.62$	$53.19\% \pm 2.12$
Geometric algorithms	$33.73\% \pm 1.61$	$41.27\% \pm 5.06$	$40.94\% \pm 4.77$	$\mathbf{44.01\% \pm 4.48}$	$34.58\% \pm 1.88$
Graph algorithms	$13.42\% \pm 2.51$	$18.54\% \pm 3.51$	$11.65\% \pm 1.95$	$28.33\% \pm 3.23$	$\mathbf{30.00\% \pm 4.12}$
Greedy algorithms	$62.26\% \pm 7.88$	$66.72\% \pm 7.31$	$60.77\% \pm 9.02$	$\mathbf{86.20\% \pm 1.42}$	$72.39\% \pm 4.70$
Search algorithms	$34.03\% \pm 9.74$	$26.39\% \pm 9.40$	$25.35\% \pm 10.08$	$\mathbf{36.11\% \pm 9.02}$	$29.51\% \pm 7.97$
Sorting algorithms	$9.51\% \pm 1.19$	$7.72\% \pm 0.48$	$\mathbf{13.32\% \pm 0.77}$	$6.07\% \pm 1.29$	$4.60\% \pm 0.75$
String algorithms	$1.04\% \pm 0.60$	$\mathbf{4.17\% \pm 2.17}$	$3.12\% \pm 1.04$	$2.60\% \pm 1.15$	$\mathbf{4.17\% \pm 2.29}$
Overall average	27.57%	30.02%	27.81%	33.22%	**35.78%**

DeepMind

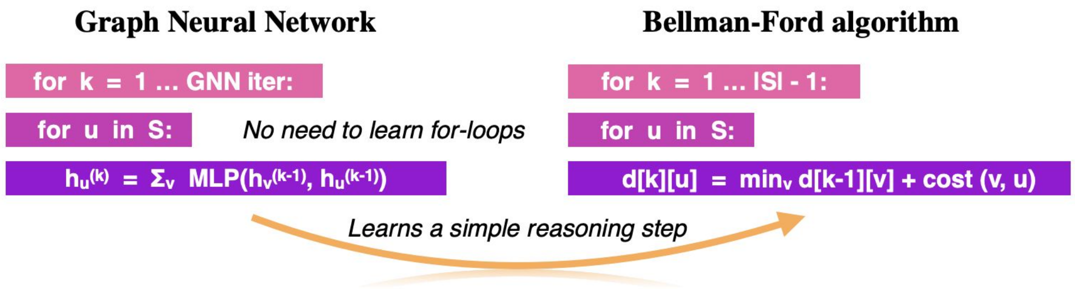
5 Deepening the theoretical link

Existing results on GNN-DP alignment

- *Graph neural networks* (GNNs) *align* well with **dynamic programming** (Xu *et al.*, ICLR'20)

- However, has this alignment truly been **demonstrated** and theoretically **quantified**?
  - It quickly became apparent that *not just any GNN* will suffice to learn an algorithm
  - Hence, the flurry of follow-up work!

- We believe that this is due to the fact the GNN-DP connection is *not* sufficiently explored!
  - The original paper merely mentions in-passing the alignment with Bellman-Ford
  - We know of no follow-up work that goes beyond this!



A comment on recent work on this connection

The Exact Class of Graph Functions Generated by Graph Neural Networks

Mohammad Fereydounian[*] Hamed Hassani[*] Javid Dadashkarimi[†] Amin Karbasi[†]

- Rigorously states which DP tasks can be solved using GNNs, when *arbitrarily initialised*
  - Min-cut works, path-finding doesn't!
  - It could be interesting to see if this is at all related to WL-style arguments

- We do not focus on this setting; we seek to classify *computational power* under the **correct** initialisation (e.g. for path-finding, one that identifies the *source* vertex)

GNNs

- Graph: a tuple of nodes and edges, G = (V, E)
- Define one-hop neighbourhoods N_u in the usual way: $\mathcal{N}_u = \{v \in V \mid (v, u) \in E\}$
- Node feature matrix **X**; omit graph- and edge-level features for clarity

$$\mathbf{h}_u = \phi\left(\bigoplus_{v \in \mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v)\right)$$

- Here, ψ is the message function, and ϕ is an update function.
- ⊕ is a permutation-invariant aggregator (e.g. sum, avg, max)

Dynamic programming

- Solve problems in a *divide et impera* fashion

- We want to solve a problem instance, *x*
  - First, identify a set of subproblems, η(*x*)
  - Recombine the answers: $f(x) = \rho(\{f(y) \mid y \in \eta(x)\})$
  - For some subproblems *y*, f(*y*) will be trivially known (*base case*)
  - NB: this induces a **graph structure** over **subproblems**!

- Often conveniently expressed **programmatically**:

$$\mathtt{dp}[\mathtt{x}] \leftarrow \mathtt{recombine}(\mathtt{score}(\mathtt{dp}[\mathtt{y}], \mathtt{dp}[\mathtt{x}])\ \mathtt{for\ y\ in\ expand}(\mathtt{x}))$$

- Also, **categorically** (de Moor, 1994):

$$\mathrm{dp} = \underbrace{\rho}_{\mathtt{recombine}} \circ \underbrace{\sigma}_{\mathtt{score}} \circ \underbrace{\eta}_{\mathtt{expand}}$$

Bellman-Ford as a specific instance

- Finding *single-source shortest paths* in a graph:

$$d_u \quad \leftarrow \quad \min\left(d_u, \min_{u \in \mathcal{N}_v} d_v + w_{v \to u}\right)$$

with base cases given by $d_s = 0$, and $+\infty$ otherwise.

- Here, the set of subproblems is *exactly* the set of nodes in the graph
    - And the expansion function yields exactly the one-hop neighbourhoods!

- **NB:** More *general* forms of Bellman-Ford exist
    - Rely on specific definitions of + and min (specific *semiring*)
    - Connection used to motivate several works, e.g. NBFNet (Zhu *et al.*, NeurIPS'21)

The difficulty of connecting GNNs and DP

- The basic technical obstacle to establishing a rigorous correspondence between neural networks and DP is the vastly different **character** of the computations they perform
  - Neural networks work through linear algebra over real numbers
  - Algorithms often operate over "tropical" objects like $(\mathbb{N} \cup \{\infty\}, \min, +)$
    - Often studied as "degenerations" of Euclidean space. Hard to reconcile!

- Our attempt: define a **latent space** *R* and minimise the assumptions over it!
  - We can then plug the appropriate *R* to describe both GNNs and DP!

- Behaviours of interest arise by instead studying *functions* $S \to R$, where *S* is a *finite* **set**
  - Principal objects are the *category of finite sets*, and "*R*-valued quantities" over them

- Our aim: find an abstract object capable of representing both GNN and DP equations
  - Our proposal: *integral transforms*

Spans, Pullback and Pushforward

- In general, we define a gadget called a *span*
  - Object E equipped with two morphisms s, t
  - When $V = W$, this can be used to describe the *edges* of a graph
  - $s(e)$, $t(e)$ give sender and receiver nodes, respectively
  - We are given data $f : V \to R$, and we need to use this span to obtain data on W

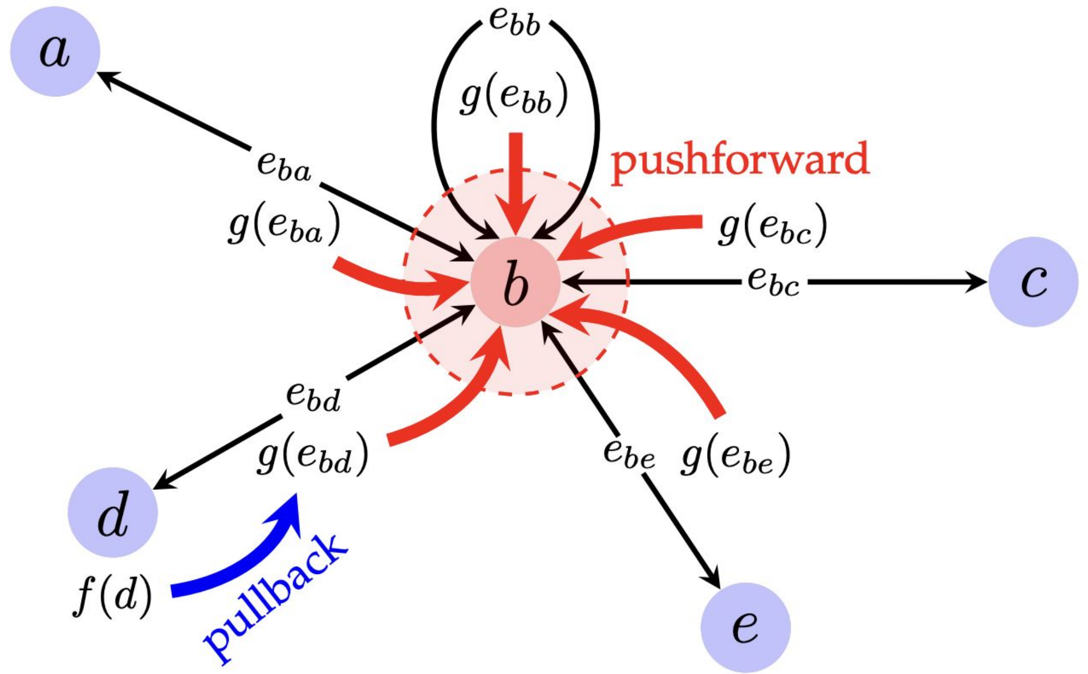
$$V \xleftarrow{\;s\;} E \xrightarrow{\;t\;} W$$

- First principal operation: **pullback** along s, which is trivial:
  - Gives us $g : E \to R$, data sent to the respective edge $s^*f := f \circ s$

- Secondly, **pushforward** along t to send messages to the receiver.
  - Here the morphism t is not facing the right way!
  - We need the ***preimage:*** $t^{-1}(w) = \{e \mid t(e) = w\}$
  - Now can define pushforward as $t_*g := g \circ t^{-1}$, but it gives us **multisets** $W \to \mathbb{N}[R]$

- Need to **aggregate** these multisets to obtain $W \to R$

Illustration of pullback and pushforward



How to aggregate?

- In general, we need an *aggregator*: $\mathbb{N}[R] \to R$ to do this final step.
  - First, observe that $\mathbb{N}[R]$ are polynomials with integer coefficients over R: $\sum_{s \in S} n_s s$
- Given a function $f : S \to T$, one can define $\mathbb{N}[f] : \mathbb{N}[S] \to \mathbb{N}[T]$ as follows:

$$\mathbb{N}[f]\left(\sum_{s \in S} n_s s\right) := \sum_{s \in S} n_s f(s)$$

- Note that, for each set S, we can also define two special functions:
  - **unit**: $S \to \mathbb{N}[S]$ (x -> {{x}})
  - **join**: $\mathbb{N}[\mathbb{N}[S]]$ -> $\mathbb{N}[S]$ (collapsing a nested sum over S into a single sum)
- This tells us that the multiset is a **monad**, and it is well-known that the algebras over such monads are *commutative monoids*
  - This imposes our **only** restriction over R; it must support a commutative monoid.

On the multiset monad

- A commutative monoid structure on R is equivalent to defining our desired **aggregator**, $\bigoplus$!

$$
\begin{tikzcd}
R \arrow[r, "\texttt{unit}"] \arrow[dr, "\texttt{id}"'] & \mathbb{N}[R] \arrow[d, "\bigoplus"] \\
& R
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbb{N}[\mathbb{N}[R]] \arrow[r, "\mathbb{N}[\bigoplus]"] \arrow[d, "\texttt{join}"'] & \mathbb{N}[R] \arrow[d, "\bigoplus"] \\
\mathbb{N}[R] \arrow[r, "\bigoplus"'] & R
\end{tikzcd}
$$

- We can now relate our discovered aggregator to the pullback and pushforward:

$$
\begin{tikzcd}
V \arrow[d, "f"'] & E \arrow[l, "s"'] \arrow[r, "t"] \arrow[d, dashed, "s^* f"] & W \arrow[d, dashed, "t_* s^* f"] \\
R & R \arrow[l, "\texttt{id}"] & \mathbb{N}[R] \arrow[l, "\bigoplus"]
\end{tikzcd}
$$

The four arrows of the integral transform

- Finally, we obtain the following four-stage diagram:

$$[V, R] \xrightarrow{\quad s^* \quad} [E, R] \xrightarrow{\quad k \quad} [E, R] \xrightarrow{\quad t_* \quad} [V, \mathbb{N}[R]] \xrightarrow{\quad \bigoplus \quad} [V, R]$$

- So long as *R* is a commutative monoid, this diagram works for both GNNs and DP!

- The diagram looks straightforward but hides a lot of **constraints** on the arrows (cf. previously shown diagrams)

- Some *embarrassingly obvious* alignments emerge when one tries to match the choice of $\bigoplus$; e.g. using max to represent path-finding DP tasks.

Some thoughts for the future

- The kernel arrow (corresponding to the GNN message function / DP scoring function)
  - We may have to dig deeper into the constraints induced by the kernel…

- We made pullback and pushforward *static* because we assume a pre-determined graph!
  - In the future, want to formalise GNNs / DP that dynamically **alter** their computations
  - For GNNs, this corresponds to methods akin to *graph rewiring*!
  - For DP, it implies not having the expansions precomputed!
  - Could be highly useful to model situations where subproblems need to be *inferred*

- Lastly, the connections detected here could stretch way deeper!
  - Integral transforms and span used to define *Fourier series* and Yang-Mills equations
  - Could we properly understand the **common ground** behind all of these?

See our paper to find out more!

Graph Neural Networks are Dynamic Programmers

ICLR'22 GroundedML / GTRL

https://arxiv.org/abs/2203.15544

Andrew Dudzik*
DeepMind
adudzik@deepmind.com

Petar Veličković*
DeepMind
petarv@deepmind.com

Abstract

Recent advances in neural algorithmic reasoning with graph neural networks (GNNs) are propped up by the notion of algorithmic alignment. Broadly, a neural network will be better at learning to execute a reasoning task (in terms of sample complexity) if its individual components align well with the target algorithm. Specifically, GNNs are claimed to align with dynamic programming (DP), a general problem-solving strategy which expresses many polynomial-time algorithms. However, has this alignment truly been demonstrated and theoretically quantified? Here we show, using methods from category theory and abstract algebra, that there exists an intricate connection between GNNs and DP, going well beyond the initial observations over individual algorithms such as Bellman-Ford. Exposing this connection, we easily verify several prior findings in the literature, and hope it will serve as a foundation for building stronger algorithmically aligned GNNs.

DeepMind

Thank you!

petarv@deepmind.com | https://petar-v.com

In collaboration with Andrew Dudzik, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell and Charles Blundell