Stanford MS&E 236 / CS 225: Lecture 9
Neural algorithmic reasoning

Ellen Vitercik*

May 1, 2024

In the last few classes, we have seen how approximation algorithms for NP-hard problems can be decomposed into learnable "modules," namely, greedy update steps. Today, we will see that the same is true for problems that are polynomial-time solvable. In particular, we will discuss a line of research called "neural algorithmic reasoning" [1, 4, 6–9, 11, 12] that has studied how to train GNNs to imitate well-known, efficient algorithms like those you may have learned in an introductory algorithms class.

1 Neural algorithmic reasoning: motivation

Before diving in, we must address the elephant in the room: if we already have an efficient algorithm that exactly solves a problem . . . why train a GNN to solve it? This literature points out that classical algorithms—such as Dijkstra's algorithm, Bellman-Ford, Ford–Fulkerson, etc.—are designed with abstraction in mind, and their inputs must conform to stringent preconditions. For example, if we were to run Dijkstra's shortest-paths algorithm in a real-world routing setting, we would be implicitly assuming that the graph's edge weights exactly equal the road network's commute times. Of course, in actuality, we may only have high-dimensional *features* about the road network, such as current congestion levels, weather patterns, time of day, and so on, all of which factor into commute times in some messy way.

So, let's assume we have messy, real-world inputs (i.e., with high-dimensional features assigned to each edge, but not exact commute times), but our algorithm only admits "abstract" inputs (i.e., graphs with a single, exact commute time per edge). The first thing we might try would be to manually convert the real-world inputs to the appropriate abstract inputs, as illustrated in Figure 1. However, this task is clearly prone to human error. Next, we might try replacing the human with a neural network, as illustrated in Figure 2. Many papers have pointed out issues with this approach [e.g., 3, 5, 10]. Veličković and Blundell [6], for example, point to its data inefficiency: real-world data is often incredibly rich, and the approach illustrated in Figure 2 requires us to compress, for example, the data describing a road segment (for example, traffic and weather conditions) down to a single scalar edge weight. The classical algorithm (e.g., Dijkstra's) then commits to using this scalar, assuming that it is perfect.

Motivated by this discussion, the overall goal will be to create a seamless, differentiable pipeline from natural inputs to outputs. Since combinatorial tasks are typically too challeng-

*These notes are course material and have not undergone formal peer review. Please feel free to send me any typos or comments.

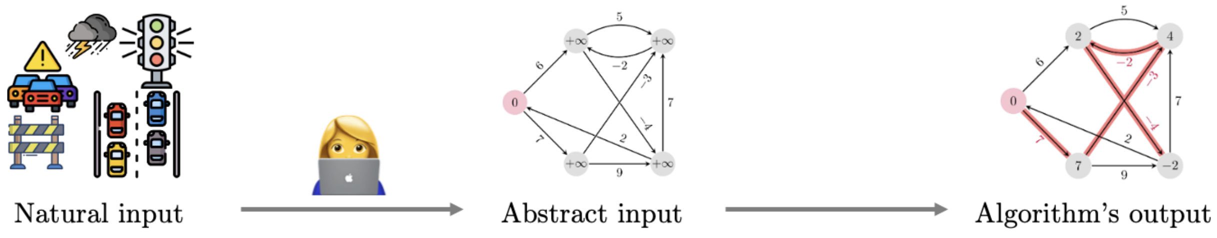


Figure 1: Natural inputs converted to abstract inputs by hand. This figure is from a talk by Petar Veličković.

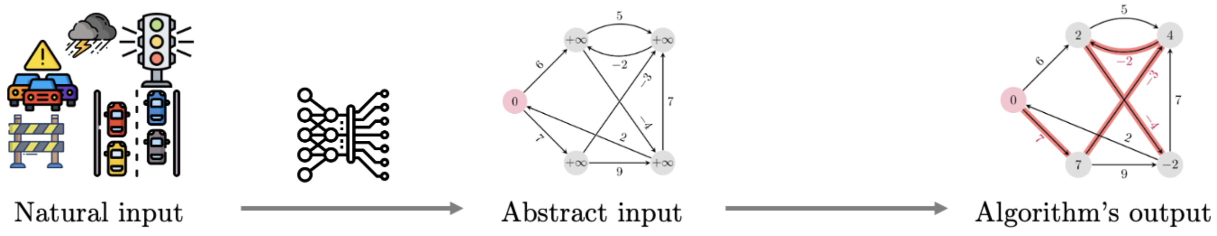


Figure 2: Natural inputs converted to abstract inputs by a machine learning model. This figure is from a talk by Petar Veličković.

ing for a model to learn end-to-end, the approach will use existing classical algorithms to (1) guide our selection of learnable modules and (2) provide intermediate supervision signals.

2 Bellman-Ford algorithm for computing shortest paths

We will use the Bellman-Ford algorithm (Algorithm 1) as a running example. Its intermediate computations are illustrated in Figure 3. Upon termination, the value $x_i^{(|V|-1)}$ is the length of the shortest path between node s and i (if there are no negative cycles in the graph). In iteration ℓ of Algorithm 1, $x_i^{(\ell)}$ is the length of the shortest path between s and i in at most ℓ hops. Inductively, the length of the shortest path between s and i in at most $\ell + 1$ hops, $x_i^{(\ell+1)}$, is the shorter of:

1. The length of the shortest path between s and i in at most ℓ hops, i.e., $x_i^{(\ell)}$, and
2. The length of the shortest path between s and one of node i's neighbors in at most ℓ hops, plus the length of the edge between that neighbor and i. In Algorithm 1, this value is denoted $h_i^{(\ell)}$.

Thus, we define

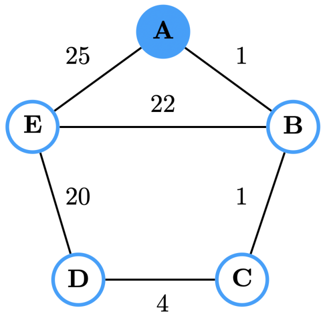
$$x_i^{(\ell+1)} \leftarrow \min\left\{h_i^{(\ell)}, x_i^{(\ell)}\right\}$$

in Step 2. Algorithm 1 can easily be updated to return the nodes and edges in the shortest path between the source node s and any other node.

In the homework, you will show that the Bellman-Ford algorithm can be written as a simple GNN. (In Lecture 6, we focused on GNNs with only node features. See Algorithm 2 for an example of how edge features could be integrated into the message-passing framework.)

Algorithm 1 Bellman-Ford algorithm

```
Input: Graph $G = (V, E)$ with edge weights $w_{i,j} \in \mathbb{R}$ for each edge $(i, j) \in E$. Source node $s \in V$.
1: for $\ell \in \{0, 1, 2, \ldots, |V| - 2\}$ do
2:     For each node $i \in V$, define $h_i^{(\ell)}$ and $x_i^{(\ell)}$ as follows:
           $h_i^{(\ell)} \leftarrow \min_{j \in N(i)} \left\{ x_j^{(\ell)} + w_{i,j} \right\}$
           $x_i^{(\ell+1)} \leftarrow \min \left\{ h_i^{(\ell)}, x_i^{(\ell)} \right\}$.
Output: $\left\{ x_i^{(|V|-1)} : i \in V \right\}$
```



(a) Input graph

	A	B	C	D	E
$\boldsymbol{x}^{(0)}$	0	∞	∞	∞	∞
$\boldsymbol{x}^{(1)}$	0	1	∞	∞	25
$\boldsymbol{x}^{(2)}$	0	1	2	45	23
$\boldsymbol{x}^{(3)}$	0	1	2	6	23
$\boldsymbol{x}^{(4)}$	0	1	2	6	23

(b) Computation trees for nodes A and B

Figure 3: Figure 3b shows the intermediate computations performed by the Bellman-Ford algorithm given the graph in Figure 3a as input. Node A is the source node.

3 Neural algorithmic reasoning: pipeline

In this lecture, we will cover a simplified version of the neural algorithmic pipeline laid out by Veličković et al. [8]. For simplicity, we will describe the pipeline when the goal is to train a GNN to imitate the Bellman-Ford algorithm and thus return the length of the shortest path between two nodes. (This can easily be extended to predicting the predecessor of each node of the shortest path, though predicting the length is easier to describe.)

The input is a graph $G = (V, E)$ together with edge features $\boldsymbol{e}_{i,j}$ for all $(i, j) \in E$ denoting, for example, traffic and weather conditions. (This can easily be extended to node and graph features as well.) The input also include a source node $s \in V$.

At a high level, across each iteration $\ell \in 0, 1, 2, \ldots, T\}$, we will update three variables:

- The scalar $\hat{x}_i^{(\ell)}$ is meant to predict the length of the shortest path between s and i in at most ℓ hops. In other words, it is a prediction of $x_i^{(\ell)}$ in Algorithm 1.
- The vector $\boldsymbol{z}_i^{(\ell)}$ is an encoding of $\hat{x}_i^{(\ell)}$ in a higher-dimensional space.
- The vector $\boldsymbol{h}_i^{(\ell)}$ is a latent embedding of node i in a higher-dimension space, which combines the node's encoding $\boldsymbol{z}_i^{(\ell)}$, its neighbors' encodings $\boldsymbol{z}_j^{(\ell)}$, and the edge features $\boldsymbol{e}_{i,j}$.

Algorithm 2 Simple message-passing graph neural network

```
Input: Graph $G = (V, E)$ with node features $\boldsymbol{x}_i$ for each $i \in V$ and edge features $\boldsymbol{e}_{i,j}$ for each $(i, j) \in E$.
1: Define initial node embeddings $\{\boldsymbol{h}_i^{(0)} = \boldsymbol{x}_i : i \in V\}$.
2: for $\ell \in \{0, 1, 2, \dots, L-1\}$ do
3:     For each node $i \in V$, define $h_i^{(\ell)}$ as follows:
           $$\boldsymbol{h}_i^{(\ell+1)} \leftarrow U_\ell\left(\boldsymbol{h}_i^{(\ell)}, \bigoplus_{j \in N(i)} M_\ell\left(\boldsymbol{h}_i^{(\ell)}, \boldsymbol{h}_j^{(\ell)}, \boldsymbol{e}_{i,j}\right)\right)$$
       ▷ The functions $U_\ell$ and $M_\ell$ are learnable, like (shallow) neural networks, and $\bigoplus$ is an aggregation function, like element-wise maximum.
Output: $\left\{\boldsymbol{h}_i^{(L)} : i \in V\right\}$
```

Algorithm 3 Simplified neural algorithmic reasoning pipeline

```
Input: Graph $G = (V, E)$ with edge features $\boldsymbol{e}_{i,j}$ for each $(i, j) \in E$ and a source node $s \in V$.
1: Define $\boldsymbol{h}_i^{(0)} = \boldsymbol{0}$ for all $i \in V$.
2: Define $\hat{x}_s^{(0)} = 0$ and $\hat{x}_i^{(0)} = \infty$ (or, for numerical stability, a large number) for all other $i \in V$
3: for $\ell \in \{0, 1, 2, \dots, T\}$ do
4:     Encode: For all $i \in V$, compute the encoding $\boldsymbol{z}_i^{(\ell)} \leftarrow f\left(\boldsymbol{h}_i^{(\ell)}, \hat{x}_i^{(\ell)}\right)$
5:     Process: Compute the latent embeddings
           $$\left\{\boldsymbol{h}_i^{(\ell+1)} : i \in V\right\} = P\left(\left\{\boldsymbol{z}_i^{(\ell)} : i \in V\right\}, \{\boldsymbol{e}_{i,j} : (i, j) \in E\}\right)$$
6:     Decode: For all $i \in V$, compute the decoding $\hat{x}_i^{(\ell+1)} = g\left(\boldsymbol{z}_i^{(\ell)}, \boldsymbol{h}_i^{(\ell+1)}\right)$
Output: $\left\{\hat{x}_i^{(T+1)} : i \in V\right\}$
```

In more detail, the pipeline depends on three functions: an encoding function f, a processing network P, and a decoding function g. Oftentimes, f and g are simply learned linear transformations, and P is a simple (e.g., single-layer) GNN. The approach is summarized by Algorithm 3. Whether to terminate the for-loop in Step 3 (i.e., the ultimate value of T) is also a learned function of the embeddings $\left\{\boldsymbol{h}_i^{(\ell)} : i \in V\right\}$.

To train the models involved in Algorithm 3, let $x_i^{(\ell)}$ be the true values computed by Bellman-Ford in Algorithm 1. The training loss is computed using this intermediate supervision (on graphs for which we know the true commute times) as

$$\sum_{\ell=1}^{T} \left\|\boldsymbol{x}^{(\ell)} - \hat{\boldsymbol{x}}^{(\ell)}\right\|.$$

Multi-task reasoning. Many algorithms have similar "modules." For example, as Veličković et al. [8] observe, the Bellman-Ford algorithm and breadth-first-search are essentially the same algorithm up to a transformation of the edge weights. This has inspired researchers

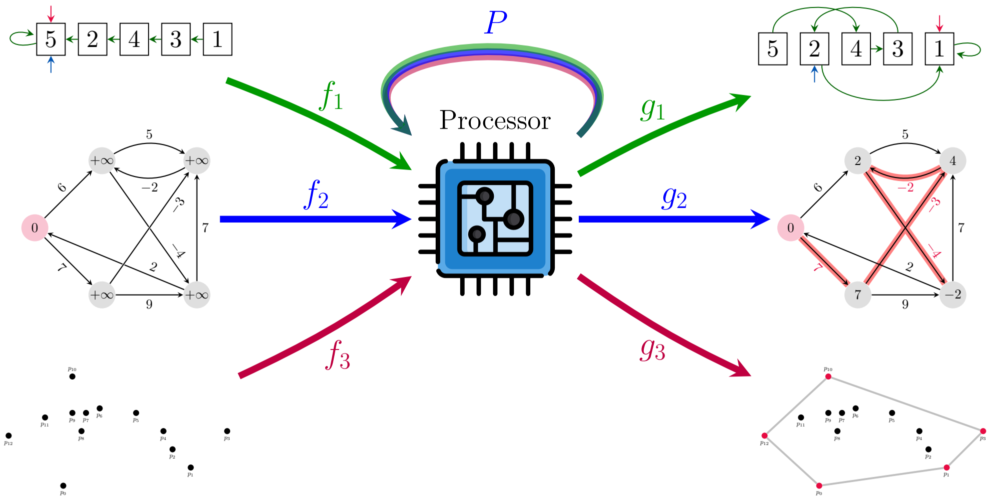


Figure 4: Illustration by Ibarz et al. [4] of their multi-task learning approach to neural algorithmic reasoning. We learn task-specific encoding and decoding functions for three algorithms: sorting, finding shortest paths, and finding convex hulls.

to multi-task approaches to neural algorithmic reasoning, where for each algorithm A, we learn task-specific encoding and decoding functions f_A and g_A, together with a single, shared processor network P. This multi-task framework is illustrated in Figure 4.

4 Results

Veličković et al. [9] developed a benchmark that generalizes Algorithm 3 to thirty different classic algorithms. It is called the "CLRS Algorithmic Reasoning Benchmark," named after the famous algorithms textbook by Cormen, Leiserson, Rivest, and Stein [2]. Graph problems, for example, are trained on Erdős-Rényi (ER) graphs with 16 nodes and tested on ER graphs with 64 nodes.

Figure 5 is from a paper by Ibarz et al. [4]. Their approach, called Triplet-GMPNN, builds significantly only the basic framework outlined in Algorithm 3. Figure 5 compares training Triplet-GMPNN using a multi-task versus single-task training pipeline. This y-axis measures the test-F_1 micro scores reported by Ibarz et al. [4]. Under shortest-paths, for example, these scores are measured based on the architecture's accuracy when predicting the predecessor of each node in the shortest path to it from the source node. As Veličković et al. [9] write, performance is not measured using scalar values: "evaluating their performance is ambiguous, and may be dependent on the way architectures choose to represent numbers." Subsequent papers [1] have improved beyond the performance exhibited by Ibarz et al. [4], as illustrated by Figure 6.

References

[1] Beatrice Bevilacqua, Kyriacos Nikiforou, Borja Ibarz, Ioana Bica, Michela Paganini, Charles Blundell, Jovana Mitrovic, and Petar Veličković. Neural algorithmic reasoning

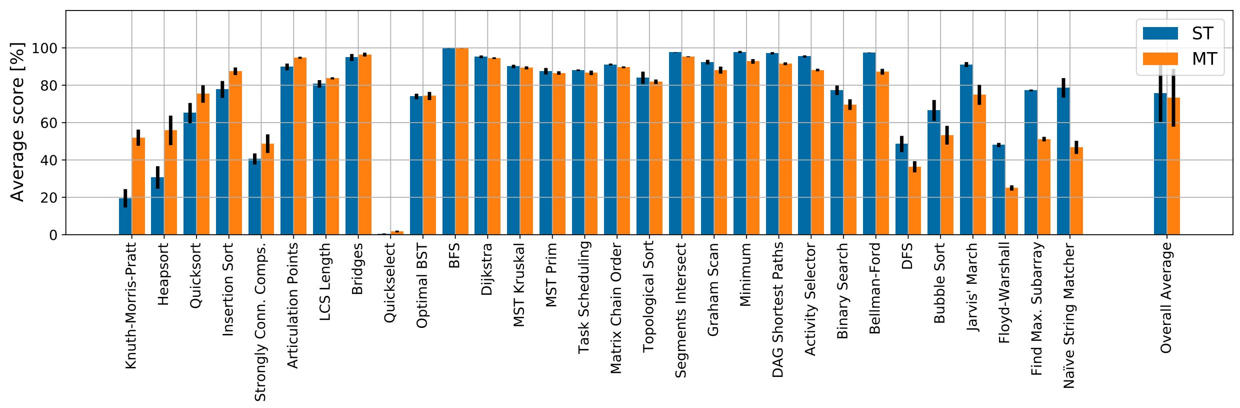


Figure 5: Test-F_1 micro scores from the paper by Ibarz et al. [4]. The single-task (ST) and multi-task (MT) bars represent the performance of the Triplet-GMPNN model by Ibarz et al. [4], differing only based on whether the model was trained using a single- or multi-task learning approach. Interestingly, some tasks, like insertion-sort, benefit from a multi-task approach, whereas others, like bubble sort, suffer.

with causal regularisation. In *International Conference on Machine Learning (ICML)*, 2023.

[2] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms.* MIT press, 2022.

[3] Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 68(1):9–26, 2022.

[4] Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Joseph Dudzik, Matko Bošnjak, Alex Vitvitskyi, Yulia Rubanova, et al. A generalist neural algorithmic learner. In *Learning on Graphs Conference*, pages 2–1. PMLR, 2022.

[5] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *Communications of the ACM*, 65(7):33–35, 2022.

[6] Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7): 100273, 2021.

[7] Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[8] Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[9] Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The CLRS algorithmic reasoning benchmark. In *International Conference on Machine Learning (ICML)*, 2022.

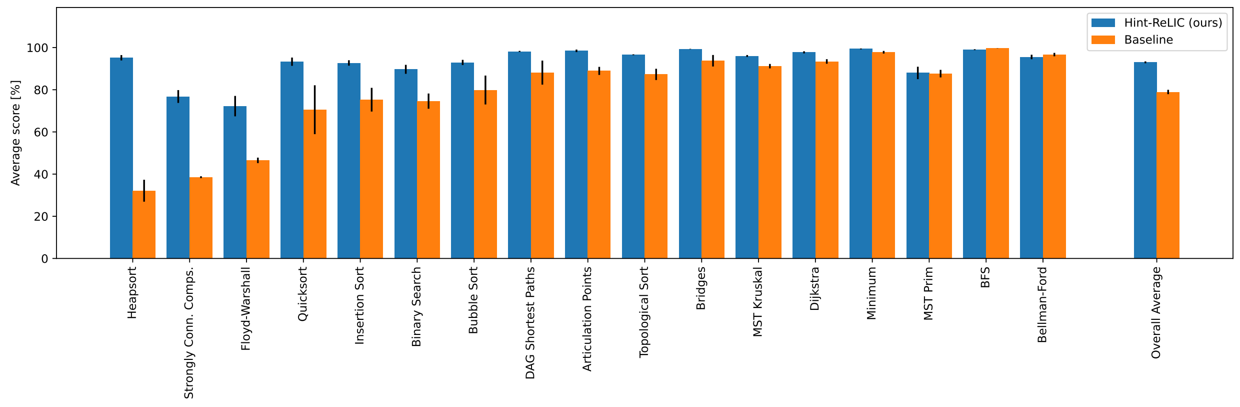


Figure 6: Test-F_1 micro scores from the paper by Bevilacqua et al. [1]. Both models are trained and tested on single tasks. The baseline is TRIPLET-GMPNN model by Ibarz et al. [4], so (unfortunately) the orange bars in this figure correspond to the blue bars in Figure 5.

[10] Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *AAAI Conference on Artificial Intelligence*, 2019.

[11] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[12] Keyulu Xu, Mozhi Zhang, Jingling Li, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.