ETH zürich

DRL meets GNN:

exploring a routing optimization use case

Hongze Wang
D.ITET

14. 03 2023

Highlights of paper

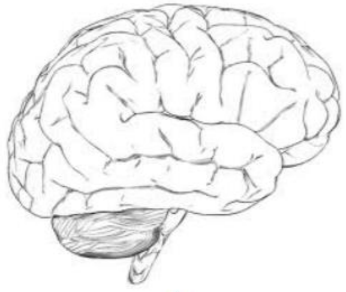
Reinforcement Learning ✚ Graph Neural Networks

Generalization Capability for RL

Graph as input data structure

What is RL?

agent



environment

Goal: Maximize the expected cumulative rewards!

RL basic concept

Episode: $s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{n-1}, a_{n-1}, r_n, s_n$

Policy: $\pi(a|s) = P(A_t = a|S_t = s)$

Q-value: $Q_\pi(s,a) = E_\pi[\sum_{n=0}^{N} \gamma^n r_{t+n} \,|S_t = s, A_t = a]$

Bellman equation

$$Q(s,a) = E[R_{t+1} + \gamma Q(s_{t+1}, a')|S_t = s, A_t = a]$$

Bellman optimality equation

$$Q^*(s,a) = E[R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') \,|S_t = s, A_t = a]$$

Q-learning

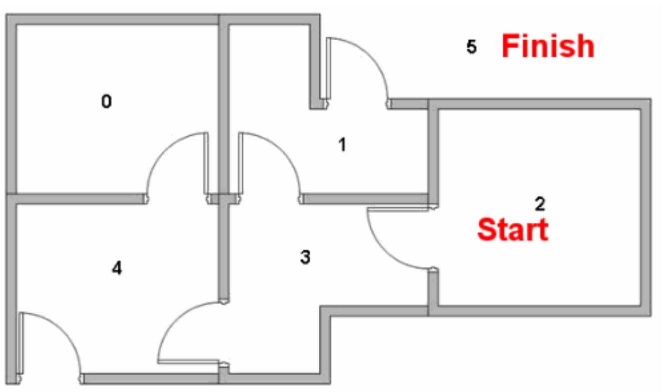
For each step t:

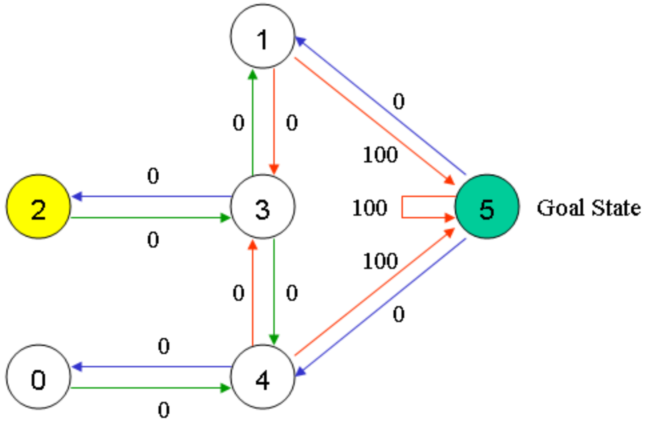
1. Choose an action a_t from s_t using policy derived from $Q - \mathrm{table}$.

2. Take the action a_t and observe r_{t+1}, s_{t+1}

3. $Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$
   $Q(s_t, a_t) = (1-\alpha) Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') \right]$

Q-learning





Initial:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$Q(s_1, a_5) = (1-\alpha)Q(s_1, a_5) + \alpha\left[100 + \gamma \max_{a'} Q(s_5, a')\right]$$
$$= 100\alpha$$

Q-learning

Initial:

$$
Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \end{array}
$$

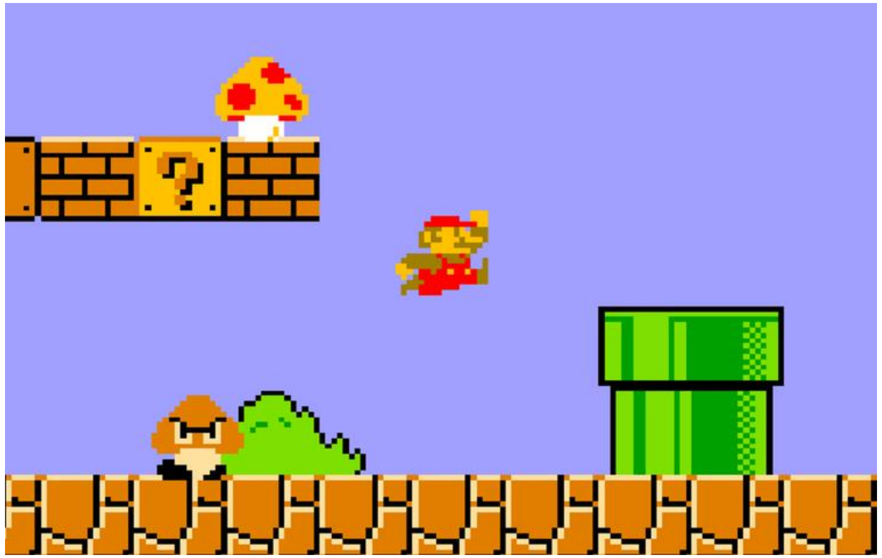
converge →

Q-table

$$
Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{bmatrix}
0 & 0 & 0 & 0 & \boxed{400} & 0 \\
0 & 0 & 0 & 320 & 0 & \boxed{500} \\
0 & 0 & 0 & \boxed{320} & 0 & 0 \\
0 & \boxed{400} & 256 & 0 & \boxed{400} & 0 \\
320 & 0 & 0 & 320 & 0 & \boxed{500} \\
0 & 400 & 0 & 0 & 400 & \boxed{500}
\end{bmatrix} \end{array}
$$

Intro to DRL: From Q-learning to DQN

For a real-world problem:

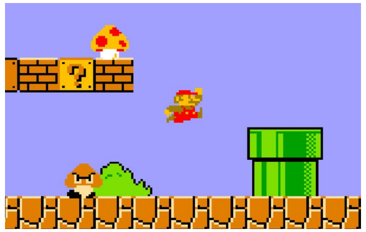


S_t *is the location of Mario*
$\leftarrow\uparrow\rightarrow$ *are the actions*

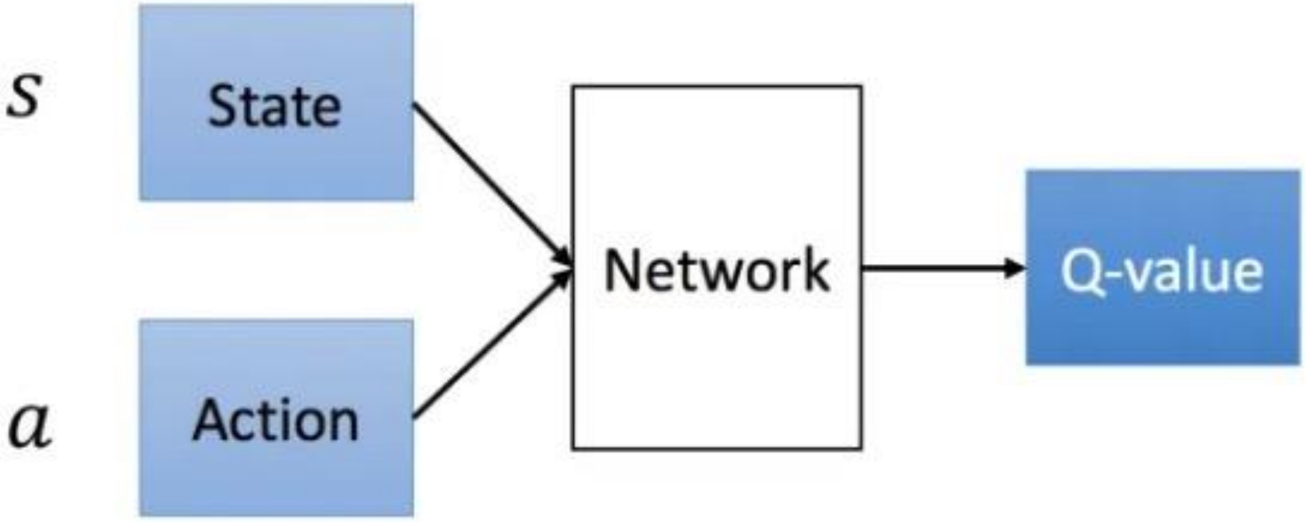
Q-table for this problem ✕

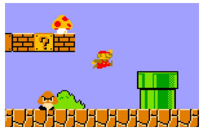
$$Q(s,a) \approx Q(s,a;\theta)$$

Intro to DRL: From Q-learning to DQN



←↑→

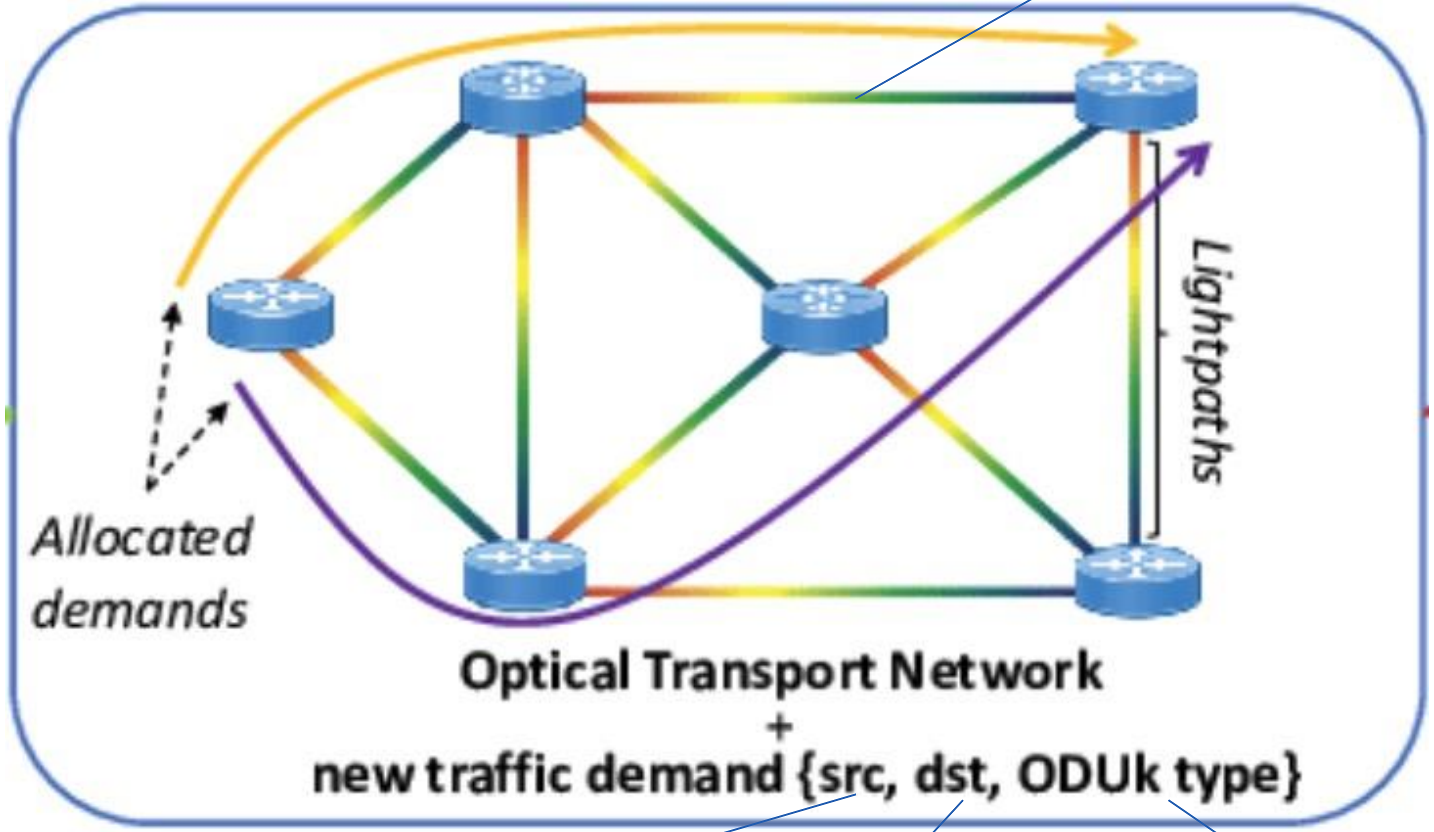


$Q($  $, \uparrow)$

You successfully learnt Deep Reinforcement Learning!

Optical Transport Networks routing problem

Available capacity of the link



Global view

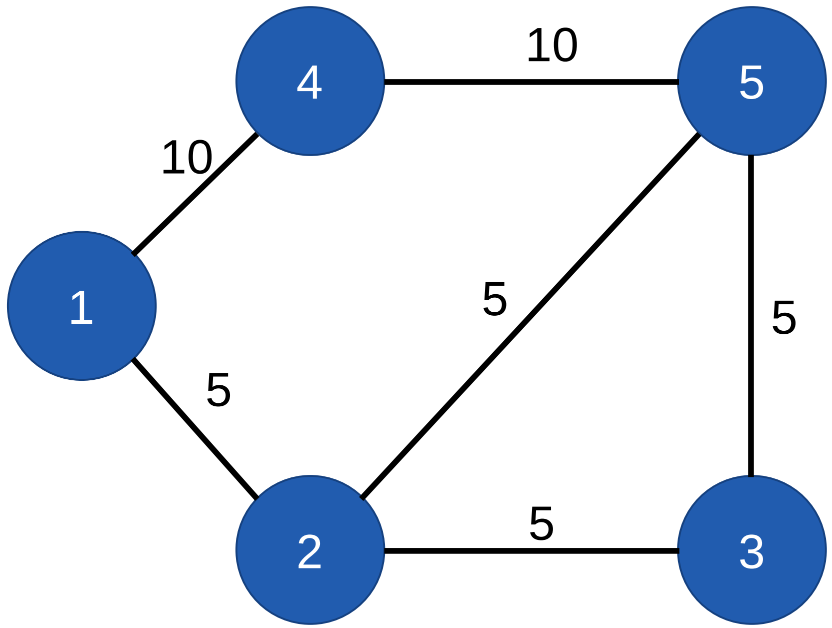
Source node

Destination node

Bandwidth of signal

Optical Transport Networks routing problem

For example:



Demand list:

1.{src=1,dst=5,bandwidth=8}

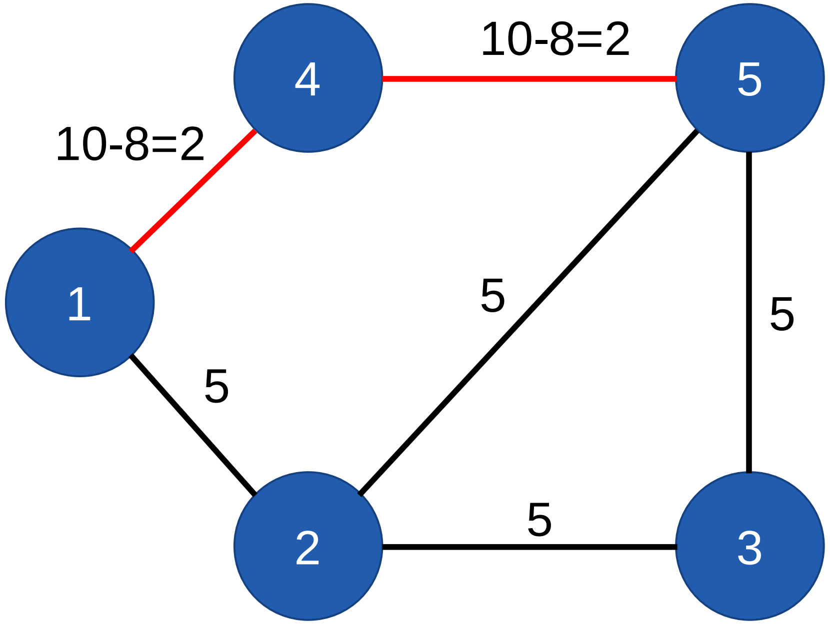
2.{src=1,dst=5,bandwidth=8}

…

n.{src,dst,bandwidth}

Optical Transport Networks routing problem

For example:



Demand list:

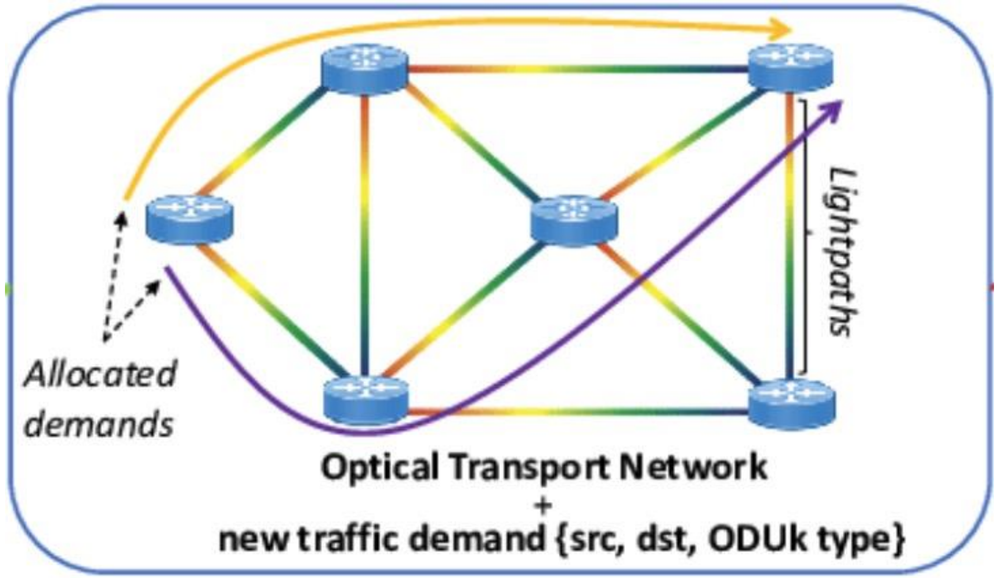
1.{src=1,dst=5,bandwidth=8} Properly allocated!

2.{src=1,dst=5,bandwidth=8} ✕ allocated!

…

n.{src,dst,bandwidth} Terminate!

Optical Transport Networks routing problem



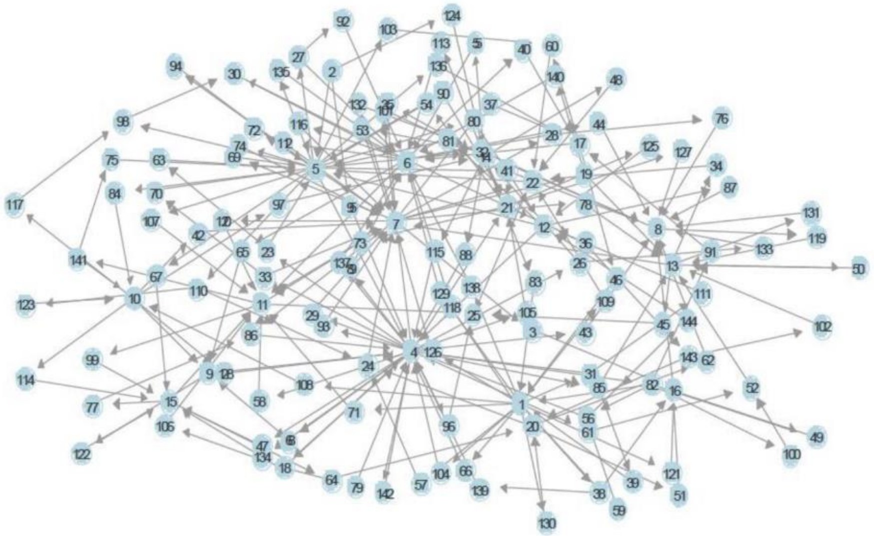
1. The agent must make decisions for every demand.
2. Traffic demands can not split over multiple paths.
3. Traffic demands will not expire until the end of episode.

Optical Transport Networks routing problem

Possible method?

Integer Linear Programming?

Constraint Programming?



Too complex

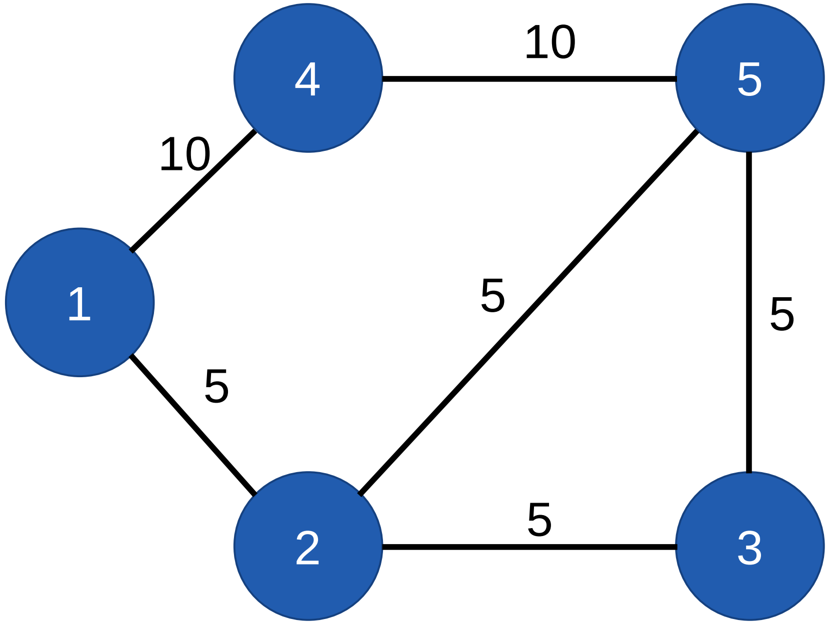
Optical Transport Networks routing problem

Possible method?

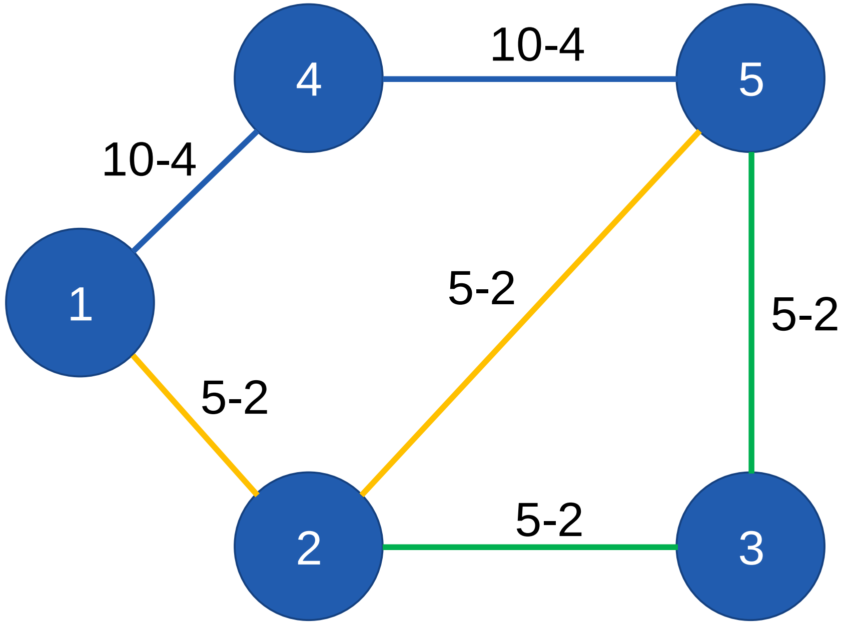
Theoretical Fluid?

Optical Transport Networks routing problem

Theoretical Fluid?



1.{src=1,dst=5,bandwidth=8}



Split into different sub-demand

Optical Transport Networks routing problem

Possible method?

2. Traffic demands can not split over multiple paths.

Theoretical Fluid?

Can not use in real world

Compute fast!
Great Performance!

Optical Transport Networks routing problem

Possible method?

MDP problem?

Dynamic programming?

…

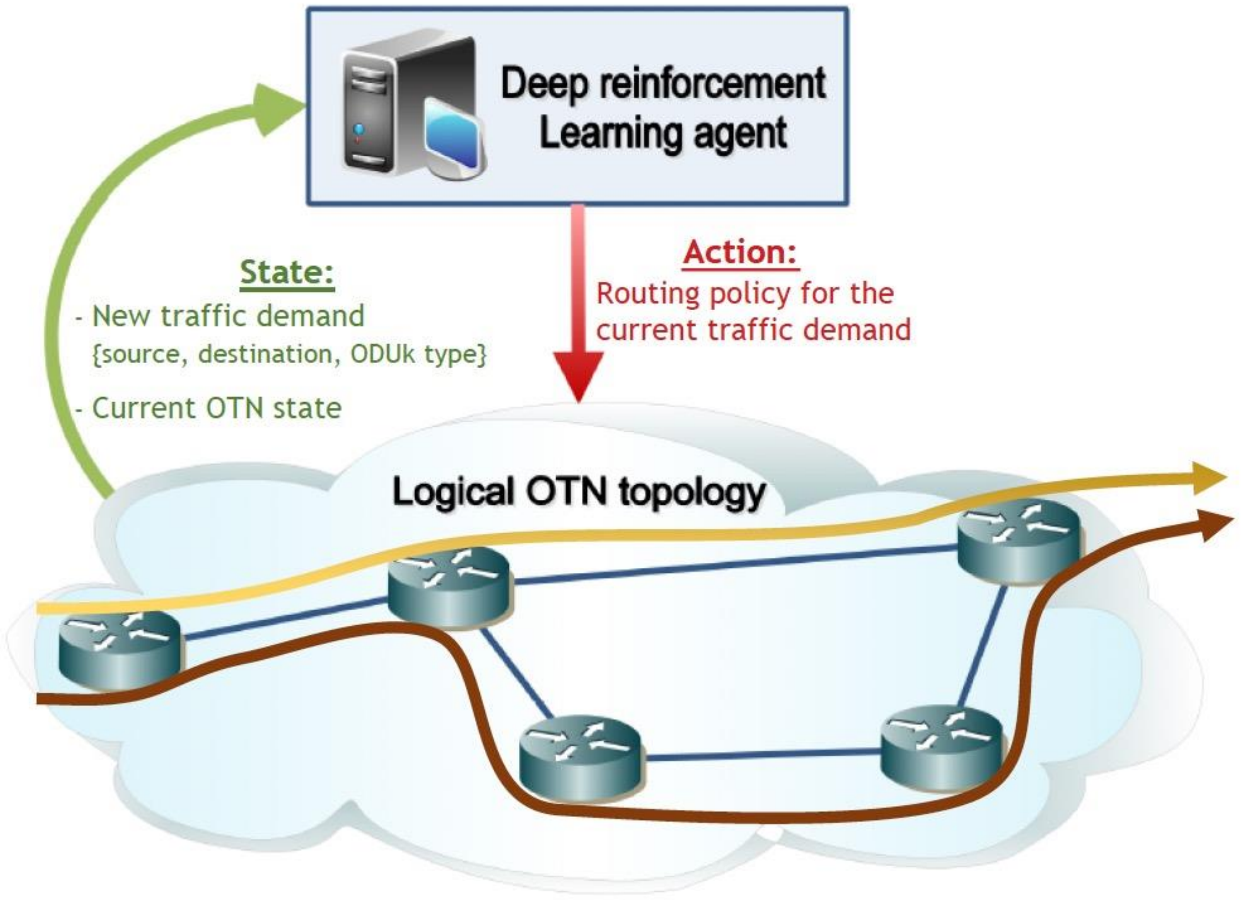
Too many states!

$S \approx O(N^E)$

Cost too much time

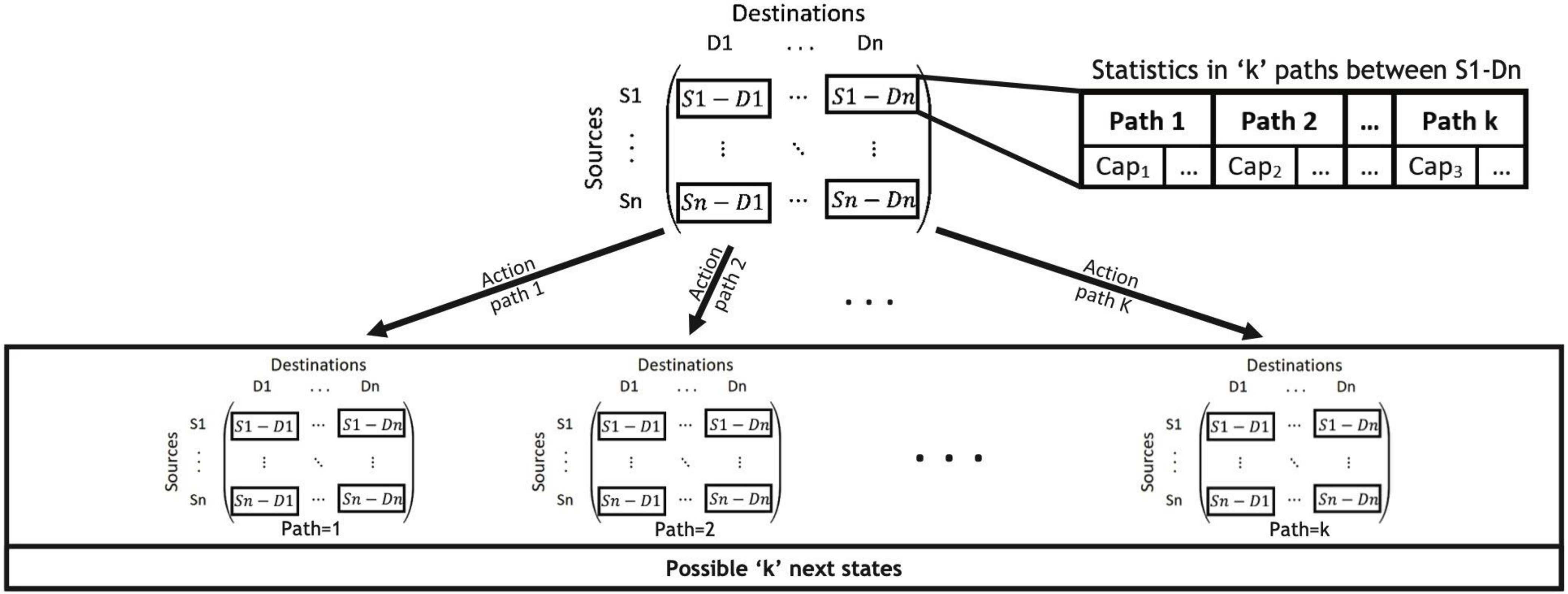
Wait! We have DRL!

DRL for Optimization (state-of-the-art algorithm)



Paper: *Routing Based On Deep Reinforcement Learning In Optical Transport Networks*

DRL for Optimization (state-of-the-art algorithm)

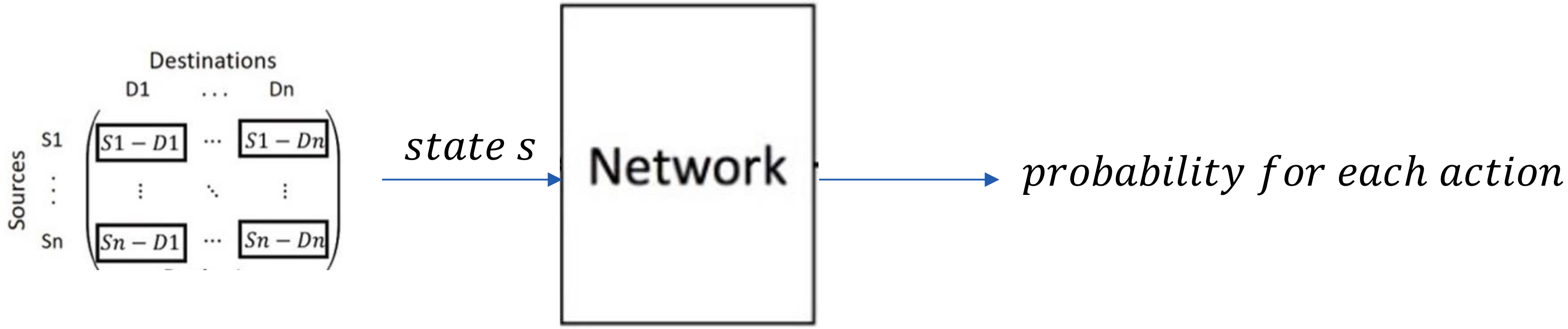


Paper: *Routing Based On Deep Reinforcement Learning In Optical Transport Networks*

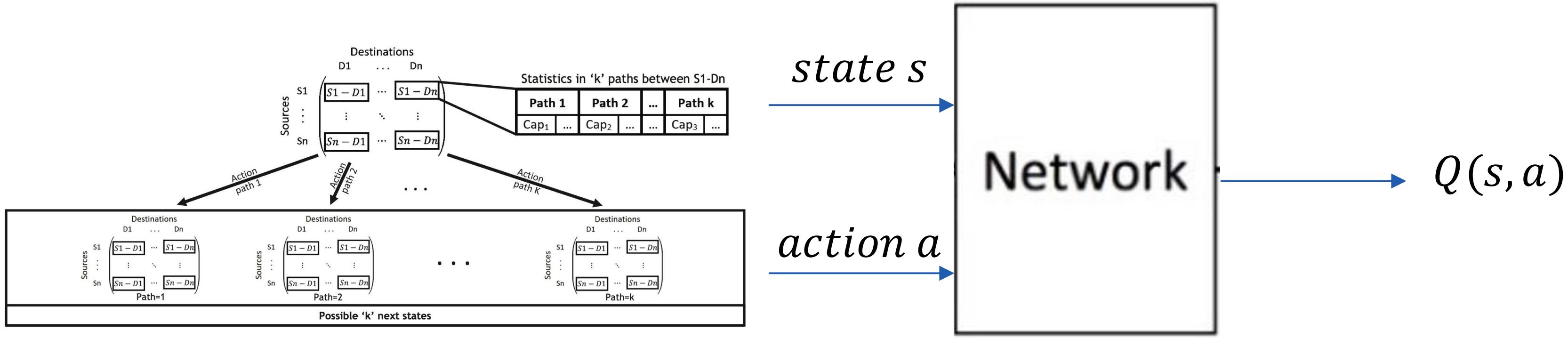
DRL for Optimization (state-of-the-art algorithm)

Actor-critic algorithm:

Actor:



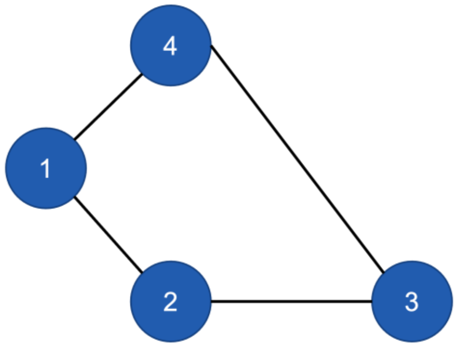
Critic

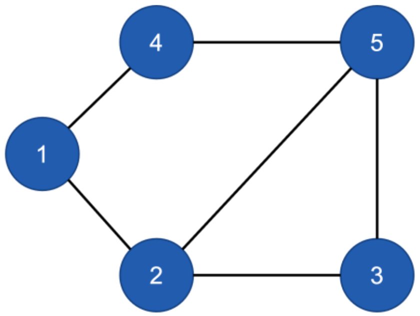


Paper: *Routing Based On Deep Reinforcement Learning In Optical Transport Networks*

DRL for Optimization (state-of-the-art algorithm)

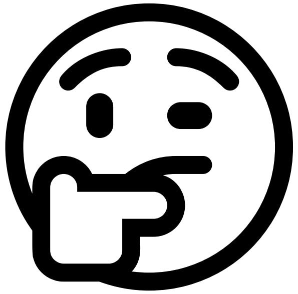
Drawback?

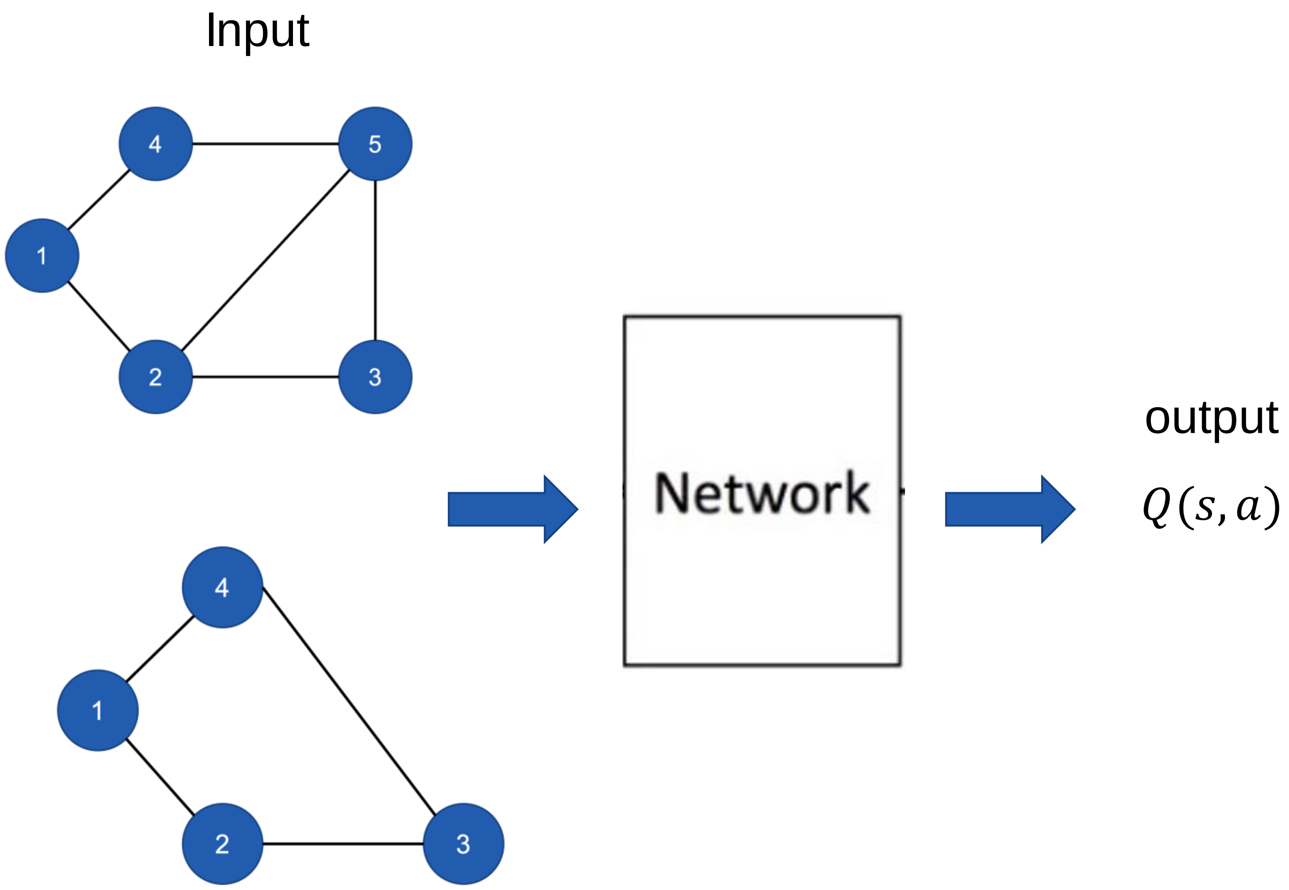




Lack of generalization capability!

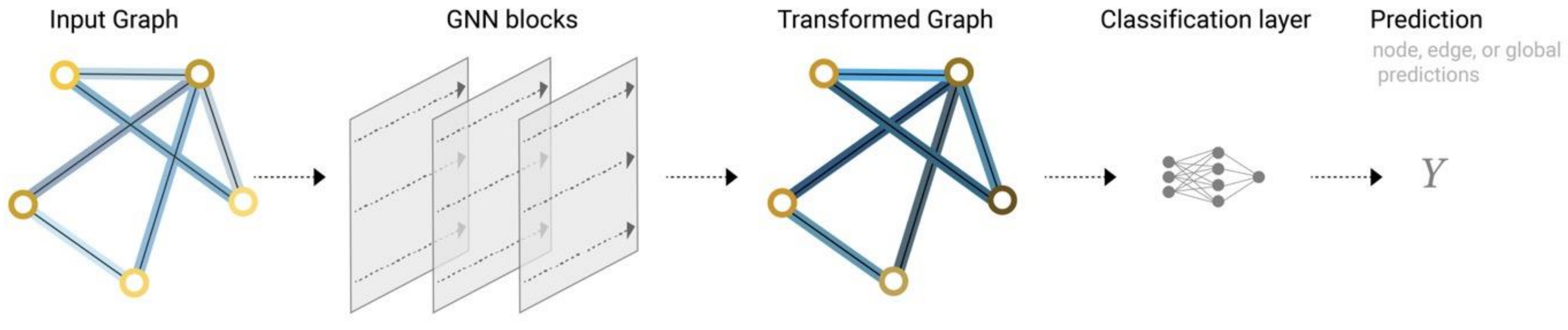
DRL for Optimization





GNN!

Graph Neural Networks (GNN)

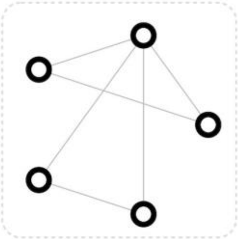


An end-to-end prediction task with a GNN model.

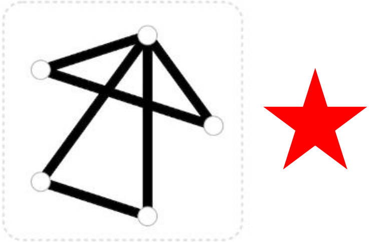
State s_t

Action a_t

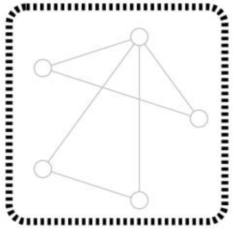
Use vertexes?



Use edges?

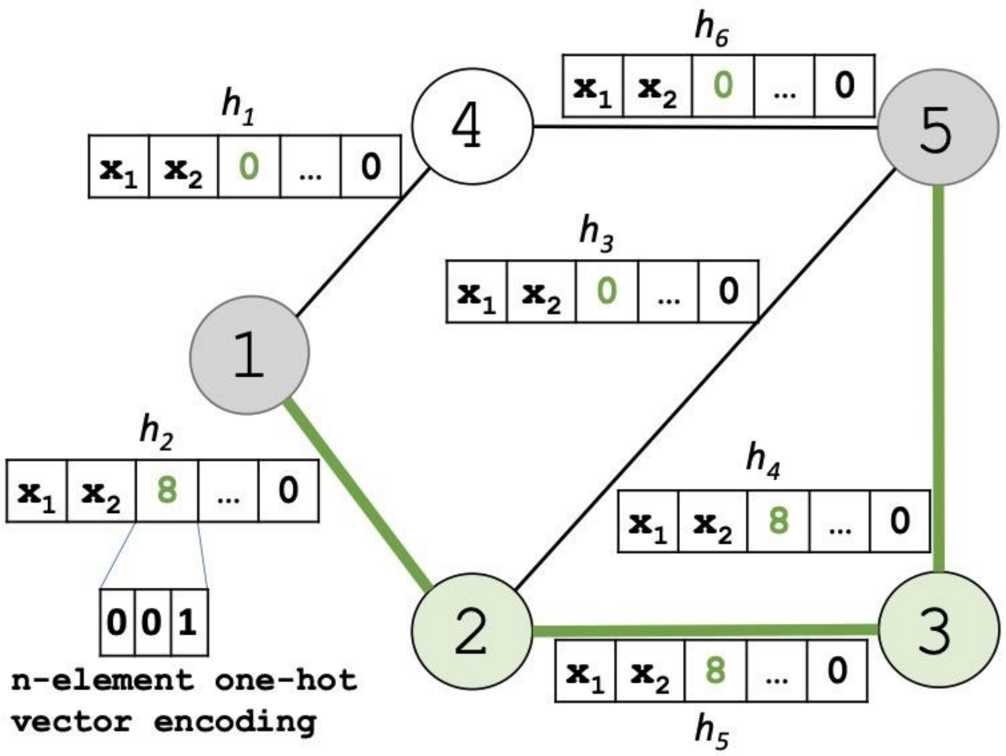


Use the whole graph?



Resource: https://distill.pub/2021/gnn-intro/

Graph Neural Networks (GNN)



Link feature

x_1 *is Link available capacity*

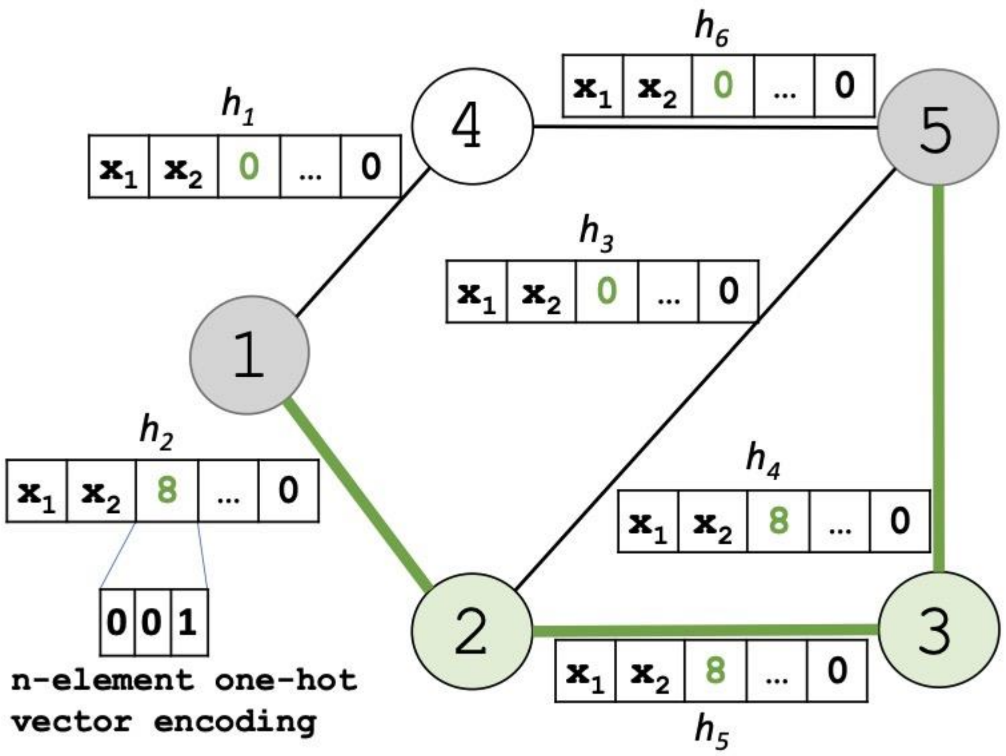
x_2 *is Link Betweenness*

x_3 *is Action vector*

$x_4 \dots x_N$ *are zeros*

Why these features?

Graph Neural Networks (GNN)



Link feature

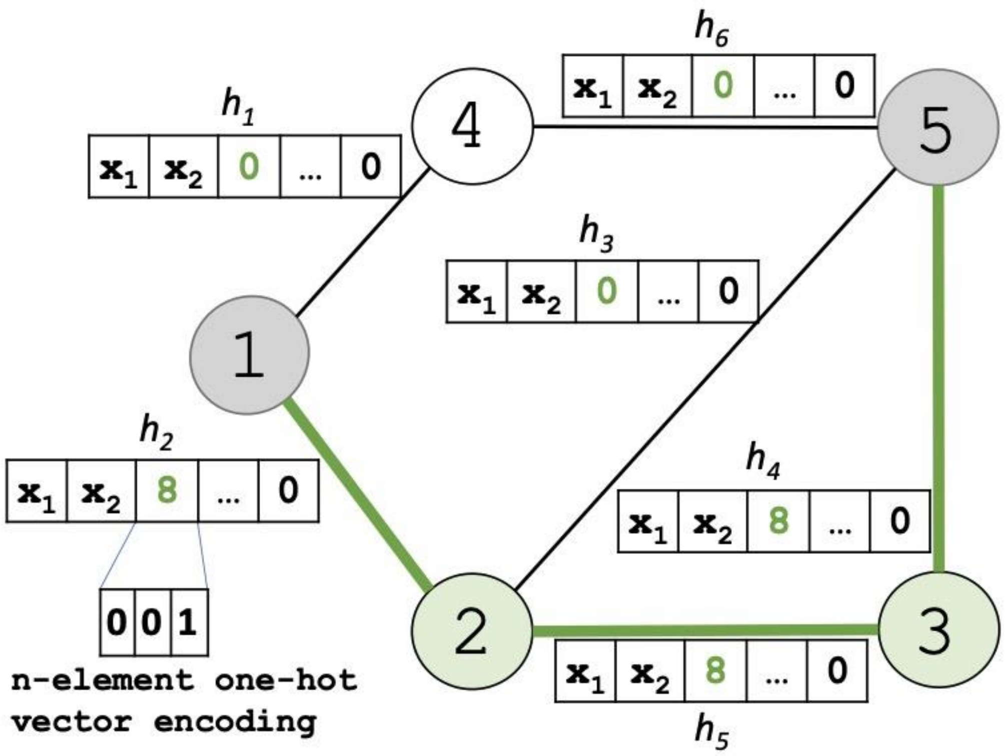
x_1 *is Link available capacity*

x_2 *is Link Betweenness*

x_3 *is Action vector*

$x_4 \ldots x_N$ *are zeros*

Graph Neural Networks (GNN)



Link feature

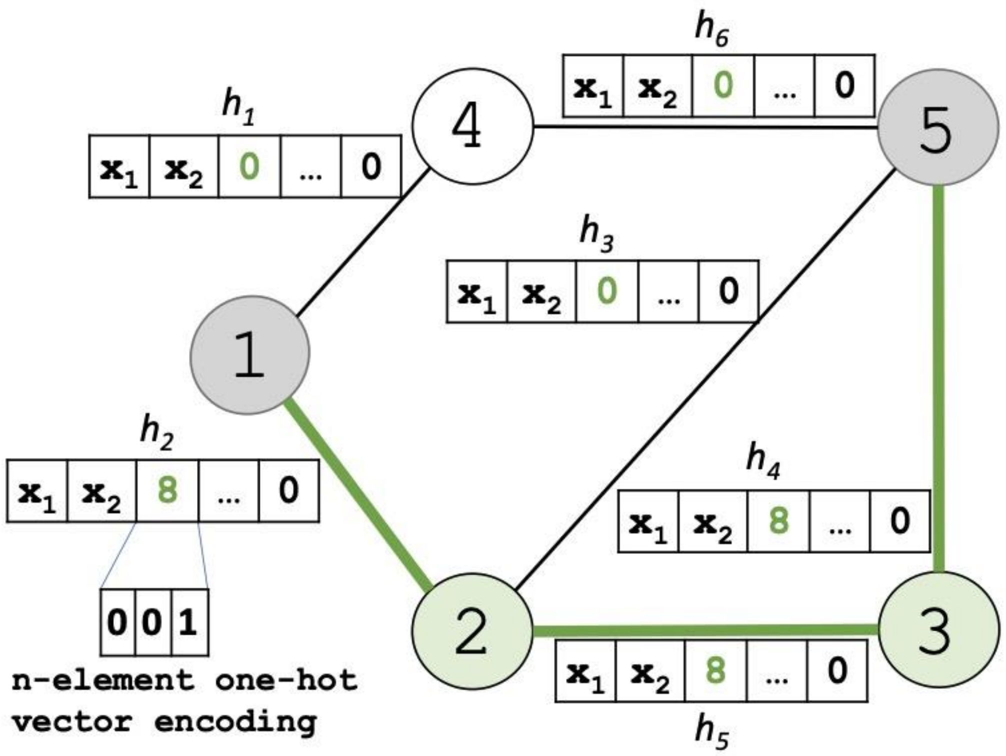
x_1 *is Link available capacity*

x_2 *is Link Betweenness*

$$Link\ Betweenness = \frac{The\ number\ of\ end\ to\ end\ paths\ crossing\ the\ link}{The\ number\ of\ tatal\ paths}$$

Guess?

Graph Neural Networks (GNN)



Link feature

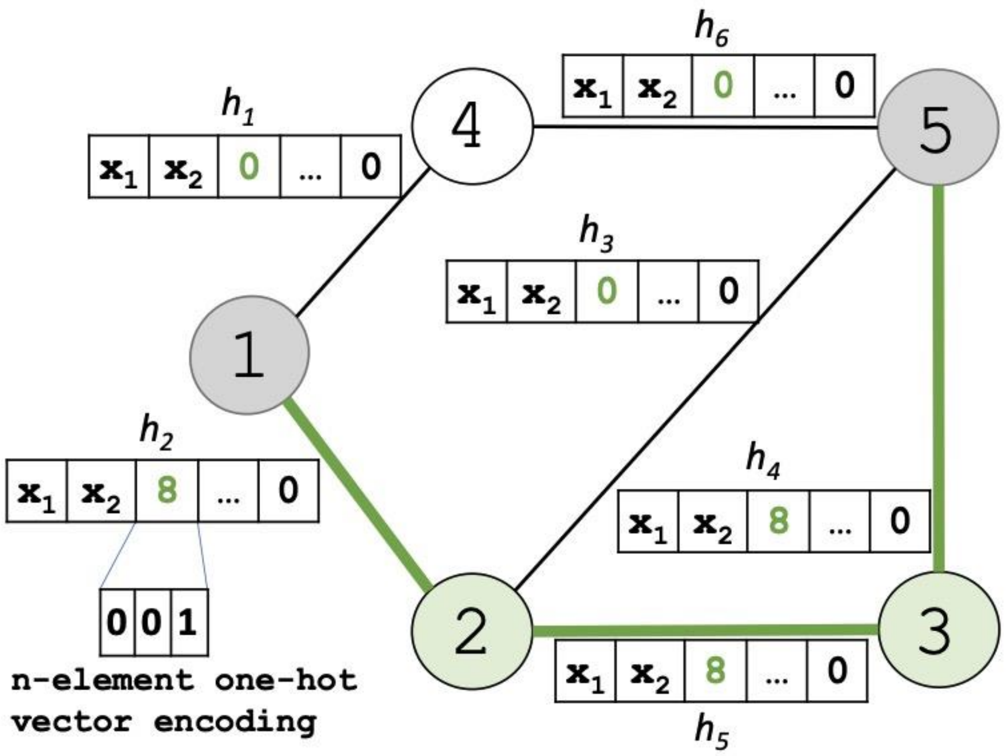
x_1 *is Link available capacity*

x_2 *is Link Betweenness*

x_3 *is Action vector*

$x_4 \dots x_N$ *are zeros*

Graph Neural Networks (GNN)



Link feature(hidden states)

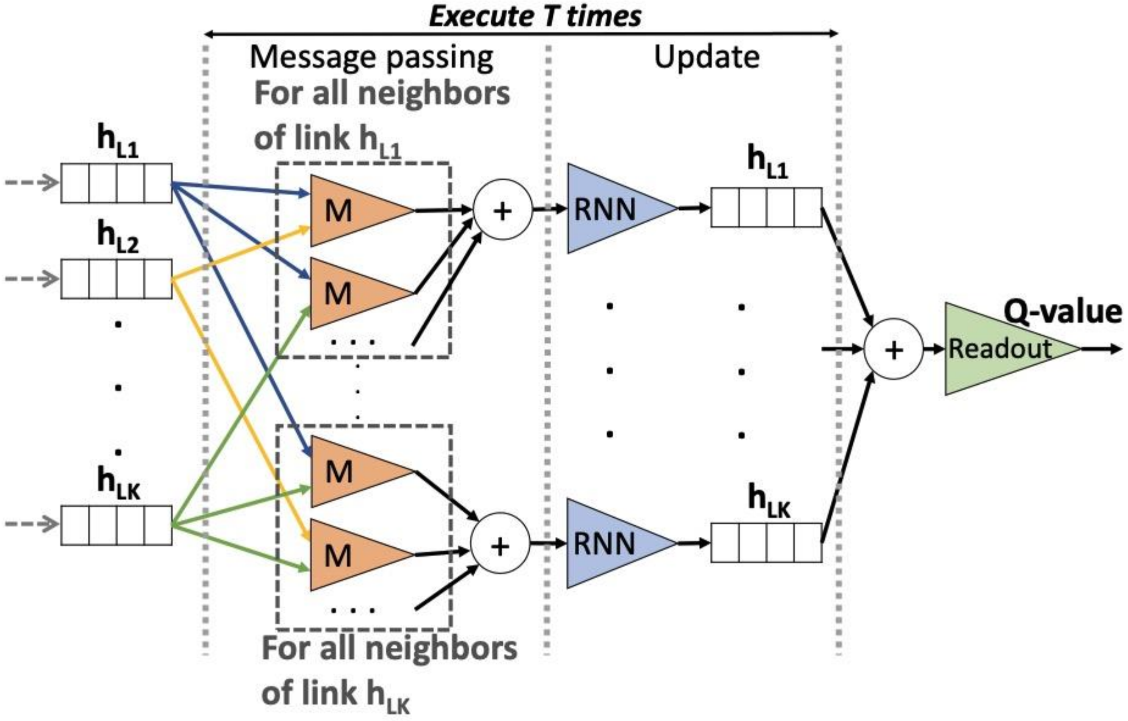
x_1 *is Link available capacity*

x_2 *is Link Betweenness*

x_3 *is Action vector*

$x_4 \ldots x_N$ *are zeros*

Message Passing Neural Network (MPNN)

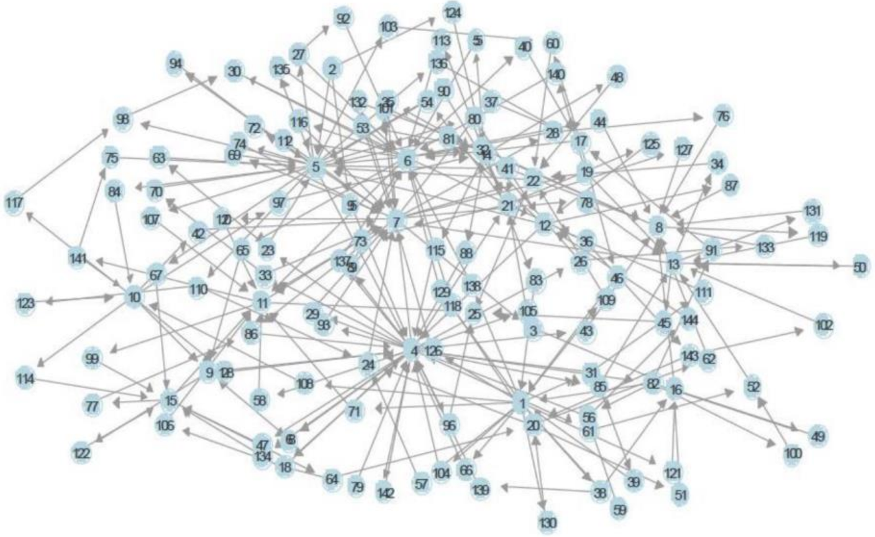


Algorithm 1 Message Passing

Input : $\mathbf{x}_l$
Output : $\mathbf{h}_l^T, q$

for each $l \in \mathcal{L}$ **do**
  $h_l^0 \leftarrow [\mathbf{x}_l, 0 \ldots, 0]$
for $t = 1$ to T **do**
  for each $l \in \mathcal{L}$ **do**
    $M_l^{t+1} = \sum_{i \in N(l)} m\left(h_l^t, h_i^t\right)$
    $h_l^{t+1} = u\left(h_l^t, M_l^{t+1}\right)$
$rdt \leftarrow \sum_{l \in \mathcal{L}} h_l$
$q \leftarrow R(rdt)$

DRL+GNN



S_t

For a demand:

Many possible actions $|A|$

Limit the action set to $k = 4$ shortest paths

Generalization

Trade off between complexity and flexibility

DRL+GNN

demand: {src,dst,bandwidth}

reward = bandwidth

reward = 0 (No capacity)

Cumulative Reward?

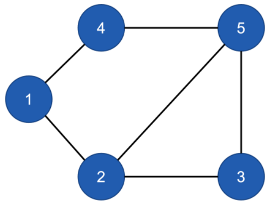
DRL + GNN algorithm

Algorithm 2 DRL Agent operation

```
s, src, dst, bw ← env.init_env()
reward ← 0, k ← 4, agt.mem ← { }, Done ← False
while not Done do
    k_q_values ← { }
    k_shortest_paths ← compute_k_paths(k, src, dst)
    for i in 0, ..., k do
        p' ← get_path(i, k_shortest_paths)
        s' ← env.alloc_demand(s, p', src, dst, dem)
        k_q_values[i] ← compute_q_value(s', p')
    q_value ← epsilon_greedy(k_q_values, ϵ)
    a ← get_action(q_value, k_shortest_paths, s)
    r, Done, s', src', dst', bw' ← env.step(s, a)
    agt.rmb(s, src, dst, bw, a, r, s', src', dst', bw')
    reward ← reward + r
    If training_steps % M == 0: agt.replay()
    src ← src'; dst ← dst'; bw ← bw', s ← s'
```

DRL+GNN

Stage 1:



+

demand: {src,dst,bandwidth}

S_t

$k = 4 \; shortest \; paths$

$a_t^1, a_t^2, a_t^3, a_t^4$

a_t

Stage 2:

$S_t + a_t^1$

$S_t + a_t^2$

$S_t + a_t^3$

$S_t + a_t^4$

MPNN

$S_{t+1}^1 + Q^1$

$S_{t+1}^2 + Q^2$

$S_{t+1}^3 + Q^3$

$S_{t+1}^4 + Q^4$

Every M steps

Stage 3:

$S_{t+1}^1 + Q^1$

$S_{t+1}^2 + Q^2$

$S_{t+1}^3 + Q^3$

$S_{t+1}^4 + Q^4$

$\epsilon - greedy$

$a_t \;\; S_{t+1} \; \mathrm{r_t}$

Stage 4:

S_t

$a_t \;\; S_{t+1} \; \mathrm{r_t}$

Buffer

Evaluation

DRL+GNN

DRL+CNN
(state-of-the-art)

LB

Theoretical Fluid

Evaluation

Demand generation:

demand: {src,dst,bandwidth}

generate uniformly

src, dst = {uniformly select a node pair}

bandwidth = {8,16,64}

Demand list:

1.{src=1,dst=5,bandwidth=8}

2.{src=3,dst=2,bandwidth=2}

…

n.{src,dst,bandwidth}

Evaluation

NSFNet:

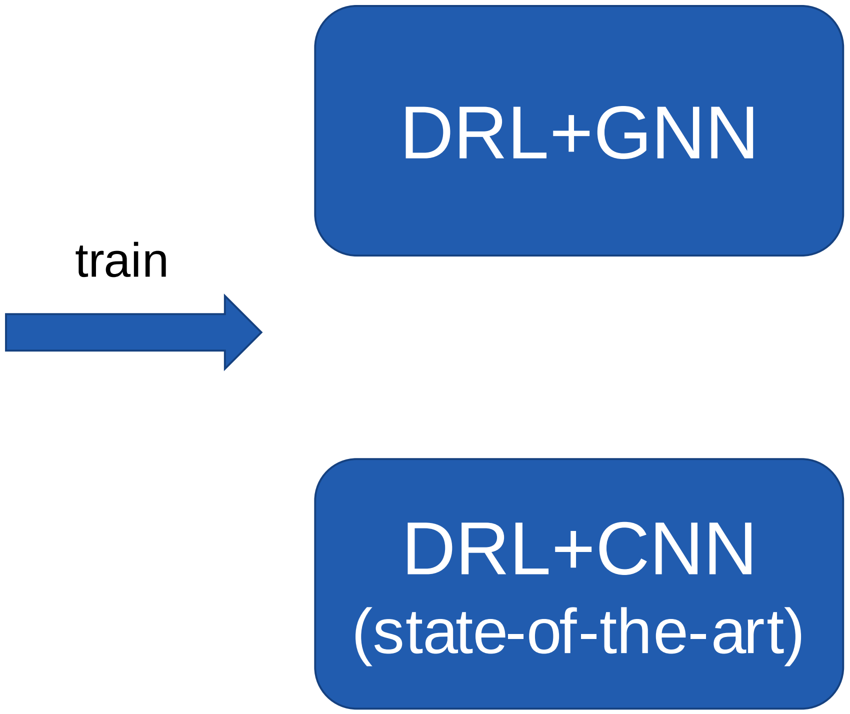
14 Nodes

Capacity = 200

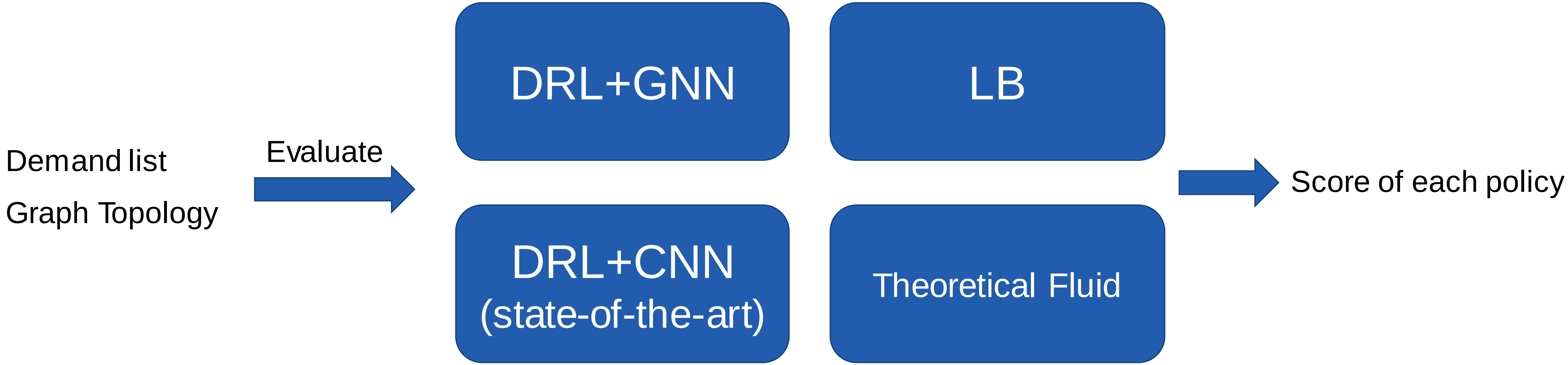
Geant2:

24 Nodes

Capacity = 200



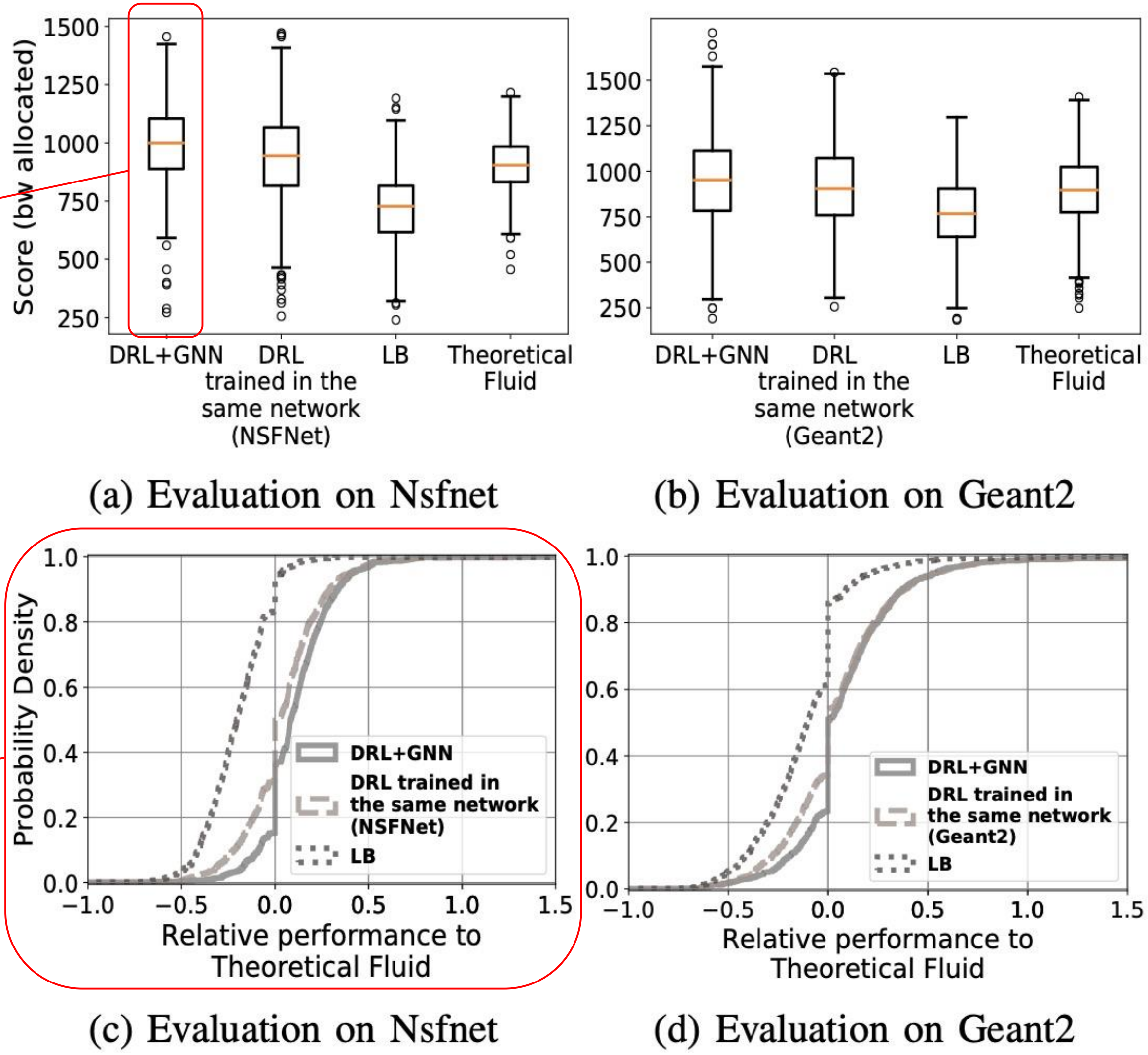
Evaluation


Demand list
Graph Topology
Evaluate
DRL+GNN
LB
DRL+CNN
(state-of-the-art)
Theoretical Fluid
Score of each policy

Evaluation

Distribution of 1000 experiments

How to read?



(a) Evaluation on Nsfnet

(b) Evaluation on Geant2

(c) Evaluation on Nsfnet

(d) Evaluation on Geant2

Evaluation

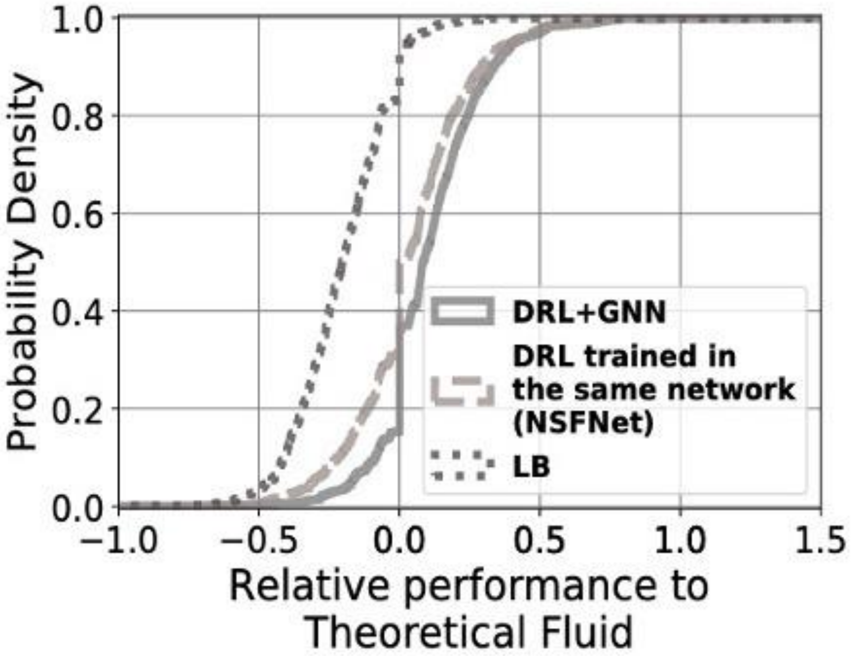
Score of 1000 experiments:

	DRL+GNN	DRL+CNN	LB	TF
1	1000	900	700	850
2	1300	1000	750	900
…				

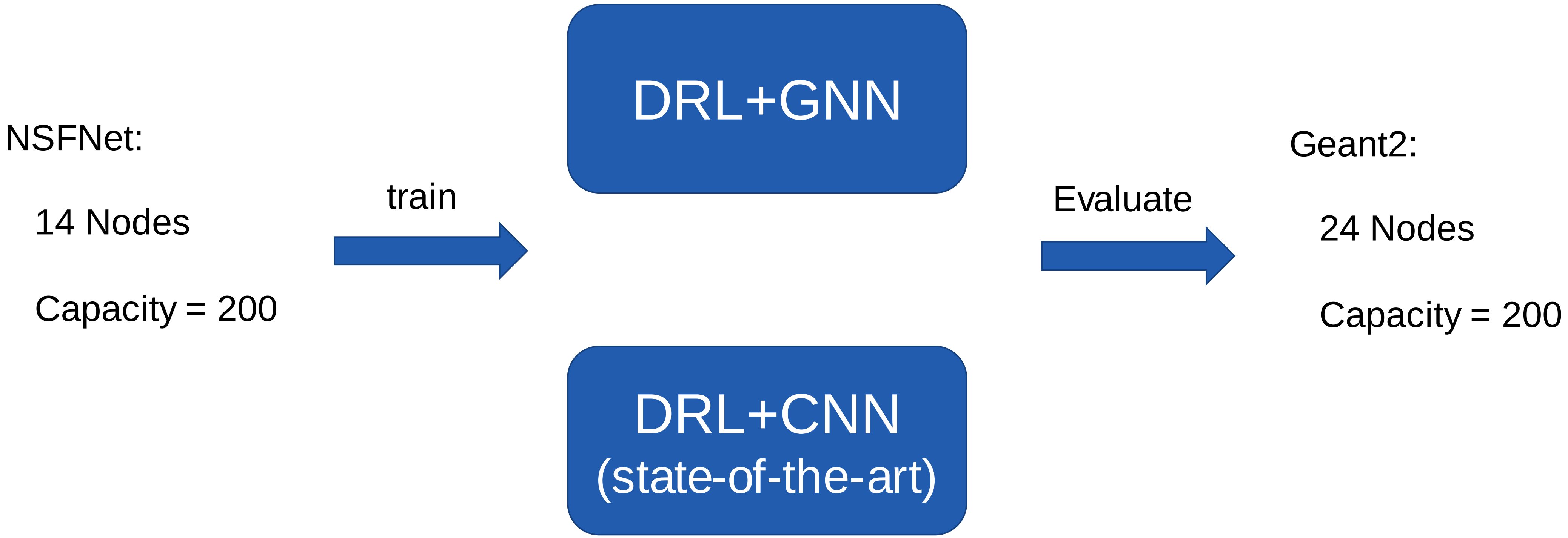
Relative performance wrt TF

	DRL+GNN	DRL+CNN	LB	TF
1	17.65%	5.88%	-17.65%	0
2	44.44%	11.11%	-16.67%	0
…				

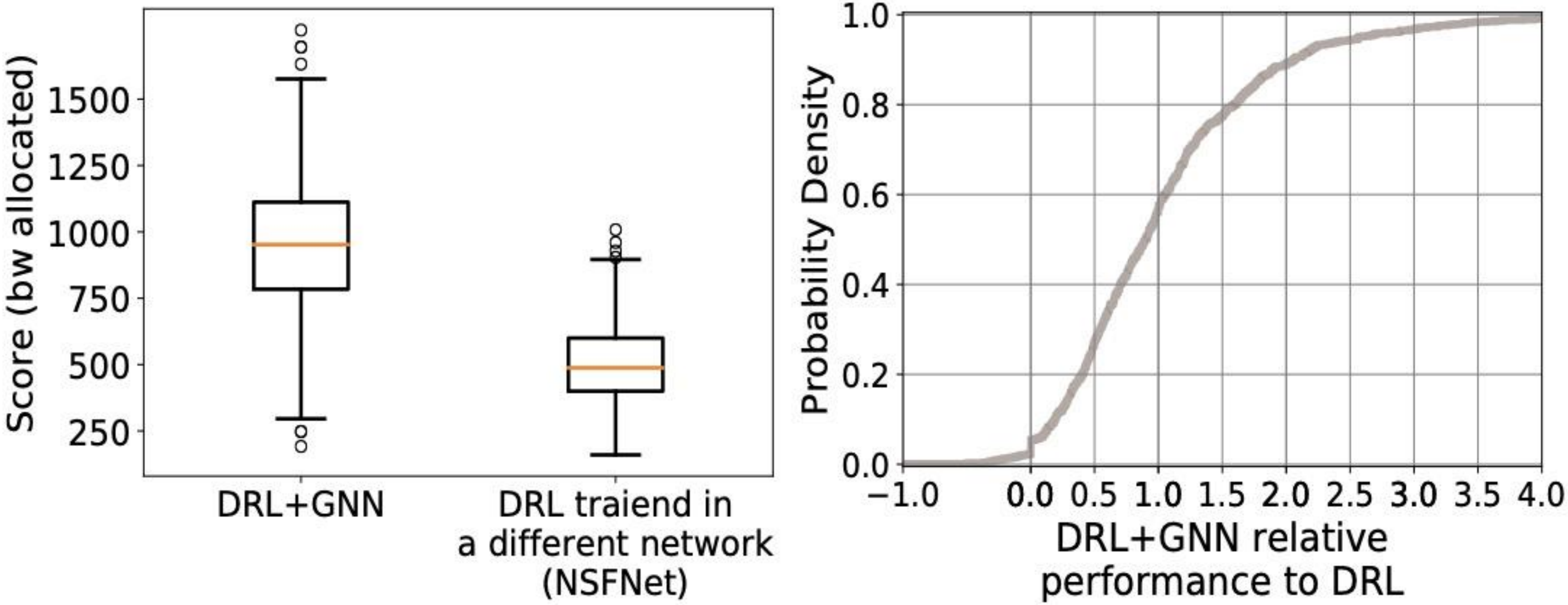
$$F_X(x) = P(X \leq x)$$



Evaluation


NSFNet:
14 Nodes
Capacity = 200
train
DRL+GNN
DRL+CNN
(state-of-the-art)
Evaluate
Geant2:
24 Nodes
Capacity = 200

Evaluation

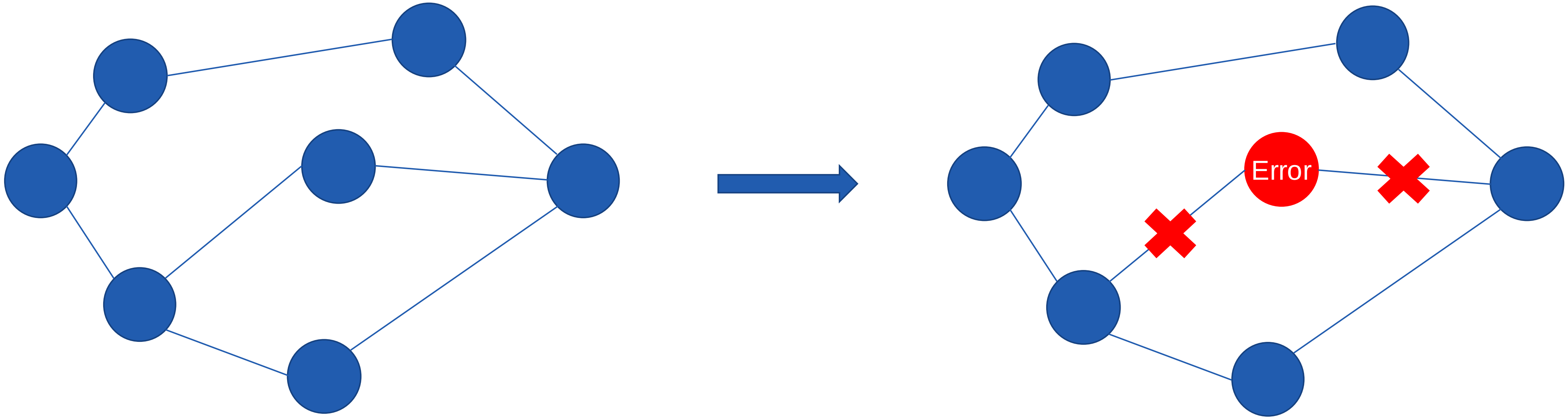


(a) Bandwidth allocated

(b) CDF

Evaluation

Real world



Evaluation

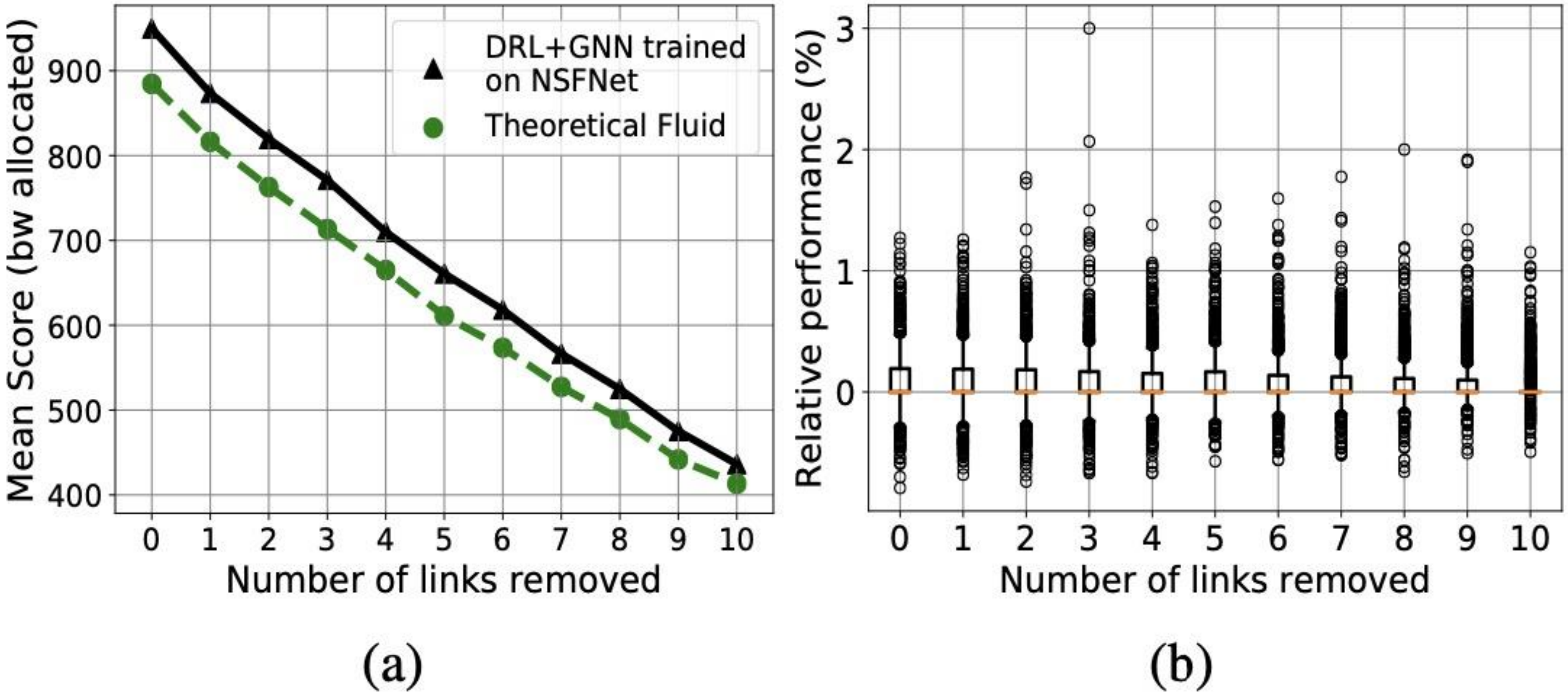


Fig. 6: DRL+GNN evaluation on a use case with link failures.

Conclusion

1. The paper combines Deep reinforcement learning with Graph neural networks.

2. For the same topology, DRL+GNN works better than other methods.

3.DRL+GNN have a better generalization capability.

Any question?

Thank you!