Deep Reinforcement Learning for Integer Programming

Akshay Ajagekar and Alyf Janmohamed

Potential Outline

1. Brief overview of MDP (e.g. our reward)
2. Methods
    a. Graph Neural Networks
    b. DQN
    c. DDQN
    d. Prioritized Replay
    e. Training Details
3. Results
    a. Graph of performance
    b. Table of performance

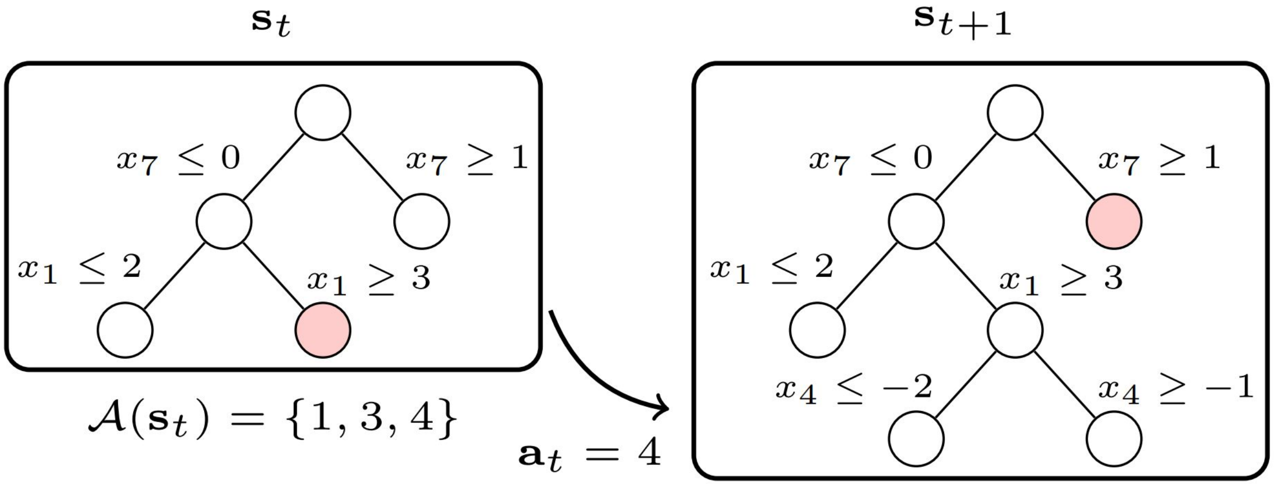
MDP Overview

State

Action

Reward

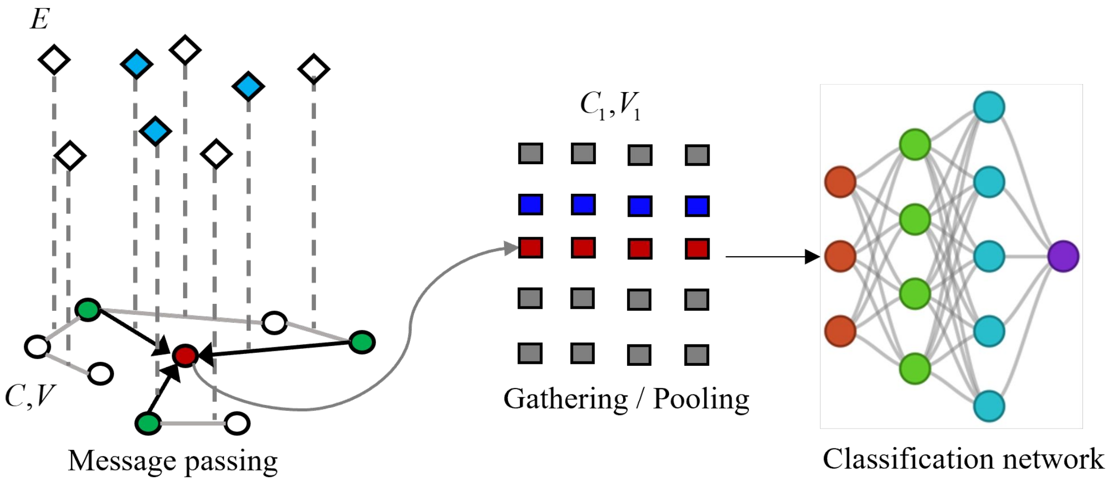
Transition



M. Gasse, D. Ch´etelat, N. Ferroni, L. Charlin, and A. Lodi, quot;Exact combinatorial optimization with graph convolutional neural networks,quot; arXiv preprint arXiv:1906.01629, 2019.

Methods : Graph Neural Networks

- GNN used to parameterize state-action value function *Q(s,a)*



- **Two half convolutions performed**
- **Layer wise normalization applied**

- Edge-conditioned filters used in the MPNN model

$$x_i' = \left(x_i W_{root} + b \right) + \sum_{j \in N(i)} x_j MLP\left(e_{ji} \right)$$

Methods : Algorithms

Deep Q-Network (DQN)

- $Q(s,a;\theta)\ \ and\ Q(s,a;\theta')$

- $Y_t^{DQN} = r_{t+1} + \gamma\ maxQ\left(s_{t+1}, a;\theta'\right)$

$$L_{DQN} = E_{(s,a,r,s')}\left[\left(Q\left(s_t, a_t;\theta\right) - Y_t^{DQN}\right)^2\right]$$

- Soft update

$$\theta' \leftarrow \tau\theta + (1-\tau)\,\theta'$$

Double Deep Q-Network (DDQN)

- $Q(s,a;\theta)\ \ and\ Q(s,a;\theta')$

- $Y_t^{DDQN} = r_{t+1} + \gamma\ Q\left(s_{t+1}, \underset{a}{argmax}\ Q\left(s_{t+1}, a;\theta\right);\theta'\right)$

$$L_{DQN} = E_{(s,a,r,s')}\left[\left(Q\left(s_t, a_t;\theta\right) - Y_t^{DDQN}\right)^2\right]$$

Algorithm 1 Q-Learning

```
1. Initialisation:
  Load a simulation environment: price series, fill probability;
  Initialise the value function V_0 and set the parameters: α, ε;
2. Optimisation:
for episode = 1,2,3... do
  for t = 1,2,3...T do
    Observe current state s_t;
    Take an action a_t(Q_t, s_t, ε);
    Observe new state s_{t+1};
    Receive reward r_t(s_t, a_t, s_{t+1});
    Update value function using r_t and current estimate Q_t:
    Compute targets and update Q(s,a)
  end for
end for
```

29

Methods - Training Details

- Implementation : TensorFlow + Spektral
- Exploration performed by epsilon greedy strategy where the exploration rate is decayed at rate of 0.995
- Agent’s memory buffer of size 5000 and sampled experiences with a batch size of 120.
- $$\gamma = 0.99,\ \tau = 0.001$$
- Data recording : TensorBoard

Methods: Prioritized Replay (PER)

- Experience replay smoothes the training distribution over the previous history of the RL agent
- Prioritized replay is a method to weight the training distribution by the TD-error
- Can be implemented using a binary heap

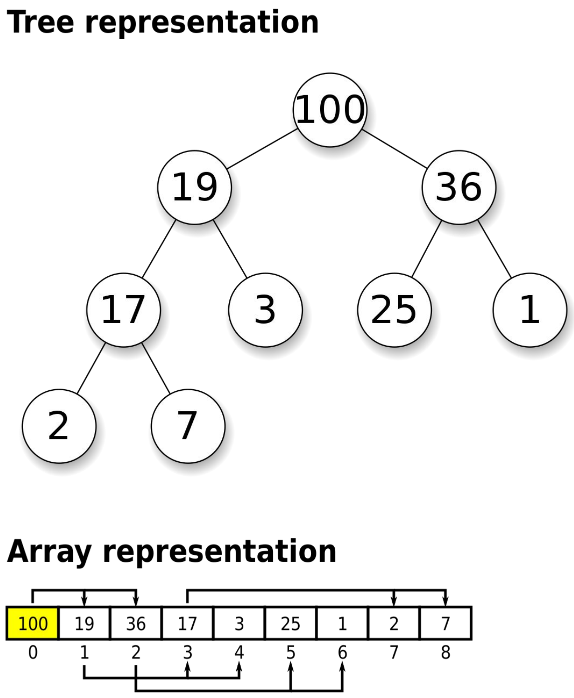
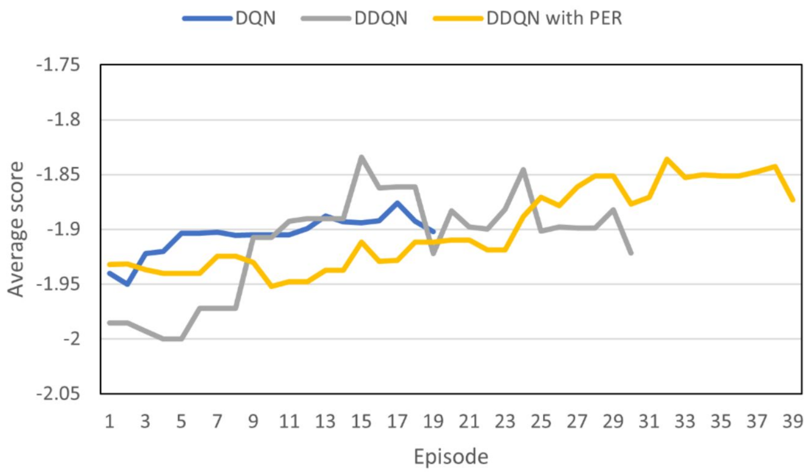


Image: https://en.wikipedia.org/wiki/Binary_heap#/media/File:Max-Heap.svg

Results



Model	Time (s)	Nodes
SCIP	1.21 ± 0.74	16 ± 25%
DQN	1.12 ± 0.3	5 ± 40%
DQN with PER	1.41 ± 0.88	10 ± 71%
DDQN	0.92 ± 0.23	5 ± 45%
DDQN with PER	1.04 ± 0.35	6 ± 60%

Further Work

1. Alternate reward functions
    - Can consider alternate metrics such as time at the node and information gained (e.g. improvement in bounds)?
2. Rules that generalize between problems
    - Can we identify rules that generalize beyond their class of problems? (e.g. set cover, capacitated facility location, max independent set)
3. Can we expand the decision space?
    - Node selection, cutting planes, etc.