GCP Approaches and Preliminary Results

Lungu Andrei, Isac Lucian, Fomin Bogdan

April 2025

1 Deep Q-Learning using GNNs

The Deep Q-Network Algorithm

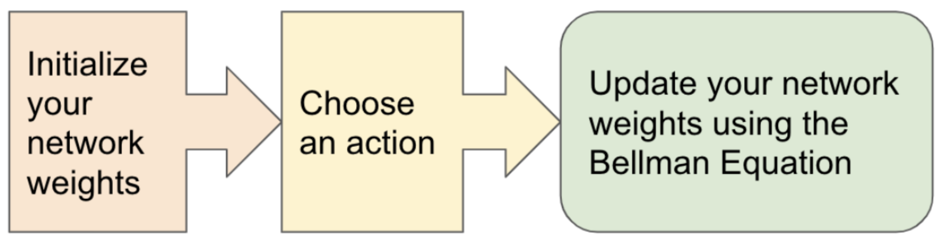


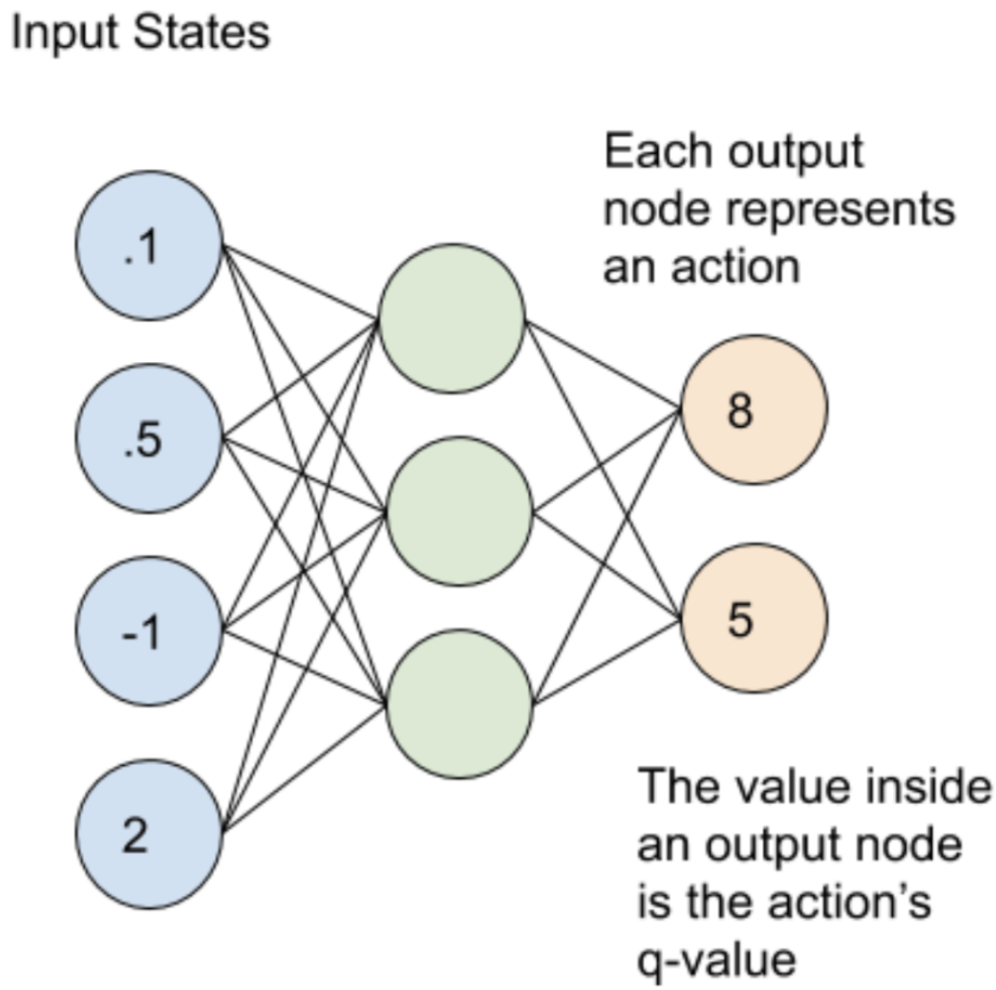
Figure 5: The Deep Q-Network Algorithm (Image by Author)

1. Initialize your Main and Target neural networks
2. Choose an action using the Epsilon-Greedy Exploration Strategy
3. Update your network weights using the Bellman Equation

A core difference between Deep Q-Learning and Vanilla Q-Learning is the implementation of the Q-table. Critically, Deep Q-Learning replaces the regular Q-table with a neural network. Rather than mapping a state-action pair to a q-value, a neural network maps input states to (action, Q-value) pairs.

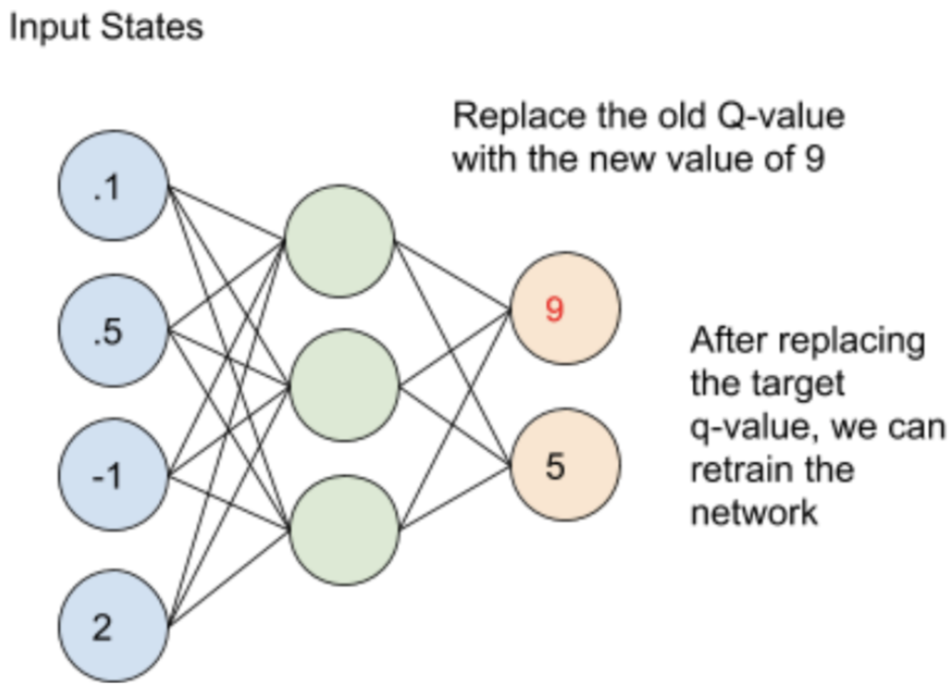
One of the interesting things about Deep Q-Learning is that the learning process uses 2 neural networks. These networks have the same architecture but different weights. Every N steps, the weights from the main network are copied to the target network. Using both of these networks leads to more stability in the learning process and helps the algorithm to learn more effectively. In our implementation, the main network weights replace the target network weights every 100 steps.

Experience Replay is the act of storing and replaying game states (the state, action, reward, next-state) that the RL algorithm is able to learn from. Experience Replay can be used in Off-Policy algorithms to learn in an offline fashion. Off-policy methods are able to update the algorithm's parameters using saved and stored information from previously taken actions. Deep Q-Learning uses Experience Replay to learn in small batches in order to avoid skewing the dataset distribution of different states, actions, rewards, and next-states that the neural network will see. Importantly, the agent doesn't need to train after each step. In our implementation, we use Experience Replay to train on small batches once every 4 steps rather than every single step. We found this trick to really help speed up our Deep Q-Learning implementation.


Input States
Each output node represents an action
.1
.5
-1
2
8
5
The value inside an output node is the action's q-value

Bellman Equation

Just like with vanilla Q-Learning, the agent still needs to update our model weights according to the Bellman Equation.



$$(R_t + \lambda * max_a Q(S_{t+1}, a))$$

State: One-hot color encoding per node.

```
 [0. 0. 0. ... 0. 1. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]]
[[0. 0. 1. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
```

Action: **Legal** color to choose from. Example: pick color 3 from legal [1, 3, 5, 6]
Reward framework:

- Reward -2 for coloring a node with a new color introduced to the graph.
- Reward 0 for coloring a node with a color that's not new to the graph.
- Reward -10 for coloring with an illegal color.

Experience replay: Array of past 10000 events. That being state for the current node, action, reward, next-state, edge-index.

2 Integer Linear Programming Models

2.1 Classical Assignment Model

The assignment model defines binary variables $x_{i,c}$ indicating whether vertex i is assigned color c. The constraints ensure that each vertex receives exactly one color and that adjacent vertices do not share the same color.

$$
\begin{aligned}
& \min z_{max} && && (1)\\
\text{s.t.} \\
& \sum_{c=1}^{H} x_{i,c} = 1 && \forall i \in V && (2)\\
& x_{i,c} + x_{j,c} \leq 1 && \forall (i,j) \in E; 1 \leq c, e \leq H && (3)\\
& z_{max} \geq c x_{i,c} && \forall i \in V; 1 \leq c \leq H && (4)\\
& x_{i,c} \in \{0,1\} && \forall i \in V; 1 \leq c \leq H && (5)\\
& z_{max} \in \mathbb{R} && && (6)
\end{aligned}
$$

2.2 Hybrid Partial-Ordering Based ILP Model

The Hybrid Partial Order Method is a novel Integer Linear Programming (ILP) formulation introduced to solve the vertex coloring problem by combining aspects of the classical assignment model and partial ordering approaches. The goal is to minimize the number of colors required to properly color the graph while maintaining a partial ordering structure.

$$
\begin{aligned}
& \min 1 + \sum_{i=1}^{H} y_{i,q} && && (1)\\
\text{s.t.} \\
& z_{v,1} = 0 && \forall v \in V && (2)\\
& y_{H,v} = 0 && \forall v \in V && (3)\\
& y_{i,v} - y_{i+1,v} \geq 0 && \forall v \in V,\ i = 1, \ldots, H-1 && (4)\\
& y_{i,v} + z_{v,i+1} = 1 && \forall v \in V,\ i = 1, \ldots, H-1 && (5)\\
& x_{v,i} = 1 - (y_{i,v} + z_{v,i}) && \forall v \in V, i = 1, \ldots, H-1 && (6)\\
& x_{u,i} + x_{v,i} \leq 1 && \forall (u,v) \in E,\ i = 1, \ldots, H && (7)\\
& y_{i,q} - y_{i,v} \geq 0 && \forall v \in V,\ i = 1, \ldots, H-1 && (8)\\
& y_{i,v}, z_{v,i} \in \{0,1\} && \forall v \in V,\ i = 1, \ldots, H && (9)
\end{aligned}
$$

3 Experimental results

The graphs used in the experiments are taken from the work of Trick, M.A., 2002, titled "Computational Symposium: Graph Coloring and Its Generalization." These graphs are used to evaluate the performance of graph coloring algorithms. The table below summarizes the solutions and experimental results for the presented models.

Filename	Graph		GCA		GCMO		Q_GNN_50	
	$\|V\|$	$\|E\|$	k^*	Avg Time [s]	k^*	Avg Time [s]	k^*	Avg Time [s]
1-FullIns_4.col	93	593	5	551.927	5	**220.613**	10	0.0432
1-FullIns_5.col	282	3247	-	-	-	-	13	0.2366
2-FullIns_4.col	212	1621	6	1542.793	6	**1405.000**	12	0.2386
3-FullIns_3.col	80	346	6	79.358	6	**35.660**	10	0.0304
3-FullIns_4.col	405	3524	7	6860.887	7	**40189.066**	12	0.3405
4-FullIns_3.col	114	541	7	74.884	7	**224.230**	10	0.04972
4-FullIns_4.col	690	6650	8	**139707.639**	8	326228.070	15	0.7301
4-Insertions_3.col	79	156	4	1500.394	4	891.687	4	**0.045**
5-FullIns_3.col	154	792	8	**156.001**	8	171.058	10	0.1124
DSJC125.1.col	125	736	5	**1481.863**	5	336.751	9	0.1001
DSJC125.5.col	125	3891	-	-	-	-	28	0.1273
DSJC125.9.col	125	6961	-	-	-	-	-	-
DSJC250.1.col	250	3218	-	-	-	-	-	-
anna.col	138	986	11	49.938	11	**38.422**	12	0.1034
david.col	87	812	11	**16.208**	11	69.604	12	0.0592
fpsol2.i.2.col	451	8691	30	8647.931	30	**18091.708**	40	0.5674
fpsol2.i.3.col	425	8688	30	**10971.131**	30	22261.259	39	0.5319
games120.col	120	1276	9	**30.828**	9	45.411	10	0.0917
homer.col	561	3258	-	-	-	-	19	0.3158
huck.col	74	602	11	17.492	11	50.015	11	**0.0427**
jean.col	80	508	10	19.249	10	34.108	10	**0.0567**
le450_15a.col	450	8168	-	-	-	-	23	0.5641
le450_15b.col	450	8169	15	**293544.982**	-	-	23	0.8354
le450_25a.col	450	8260	25	4797.352	25	**16937.937**	30	0.8444
le450_25b.col	450	8263	25	**1109.989**	25	6593.416	30	0.856
le450_5a.col	450	5714	5	91082.342	5	**132220.429**	14	0.5794
le450_5b.col	450	5734	5	**101281.902**	5	330164.498	14	0.5934
miles1000.col	128	6432	42	5535.919	42	**52164.028**	43	0.1001
miles250.col	128	774	8	**24.868**	8	48.515	10	0.0784
miles500.col	128	2340	20	55.109	20	**130.001**	22	0.1115
miles750.col	128	4226	31	722.343	31	**1036.207**	34	0.09801
mug100_1.col	100	166	4	464.081	4	**227.726**	6	0.0233
mug100_25.col	100	166	4	504.788	4	**256.657**	6	0.0220
mulsol.i.1.col	197	3925	49	**5354.243**	49	8128.514	49	0.1796
mulsol.i.2.col	188	3885	31	773.851	31	**1722.966**	36	0.1825
mulsol.i.3.col	184	3916	31	**1465.258**	31	2615.020	36	0.1763
mulsol.i.4.col	185	3946	31	**1460.488**	31	2159.745	36	0.1771
mulsol.i.5.col	186	3973	31	**1443.901**	31	1470.731	36	0.1710
myciel3.col	11	20	4	8.612	4	96.649	4	**0.0045**
myciel4.col	23	71	5	253.726	5	**239.840**	6	0.0067
myciel5.col	47	236	6	**1238.089**	6	1260.944	7	0.0162
myciel6.col	95	755	-	-	-	-	10	0.0327
myciel7.col	191	2360	-	-	-	-	9	0.113
queen10_10.col	100	2940	-	-	-	-	21	0.0633
queen11_11.col	121	3960	-	-	-	-	23	0.0723
queen12_12.col	144	5192	-	-	-	-	23	0.1179
queen13_13.col	169	6656	-	-	-	-	25	0.1637
queen14_14.col	196	8372	-	-	-	-	28	0.1947
queen5_5.col	25	320	5	**1.621**	5	2.311	6	0.0097
queen6_6.col	36	580	7	330.278	7	**236.386**	10	0.0146
queen7_7.col	49	952	7	**392.637**	7	434.787	12	0.0233
queen8_12.col	96	2736	12	658.831	12	**614.457**	19	0.0808
queen8_8.col	64	1456	9	**2397.146**	9	5499.739	16	0.0382
queen9_9.col	81	2112	10	**170747.021**	-	-	20	0.0542
zeroin.i.1.col	211	4100	49	**2705.986**	49	7396.734	49	0.1451
zeroin.i.2.col	211	3541	30	**2299.254**	30	3689.648	39	0.1357
zeroin.i.3.col	206	3540	30	**2416.741**	30	3761.858	39	0.1265

Table 1: Performance of Graph Coloring Methods with Predicted Chromatic Number K and Runtime (s)