# CSE 326 Analysis and Design of Algorithms

**Dr.Walid Gomaa**

| **Name** | **ID** |
|---|---|
| Mohamed Abdelmonem Makram | 120220055 |
| Abdelrahman Ahmed Shaheen | 120220228 |
| Abdelrhman Mohamed Eldenary | 120220253 |
| Anas Ihab Badr | 120220360 |

E-JUST

Computer Science Engineering Department
Egypt-Japan University of Science and Technology

# Contents

## 1 Introduction

Our initial attempt to apply reinforcement learning (RL) to the Schur number problem did not yield promising results. The agent struggled to find long valid colorings, indicating that our chosen RL setup and reward structure were not well-suited to this combinatorial problem. This suggests the need to identify an appropriate RL paradigm or network architecture that can handle large discrete state spaces. While the Schur problem itself has not been directly tackled with RL before, similar domains have seen success: for example, DeepMind's AlphaGo combined policy and value neural networks with self-play reinforcement learning to master the complex game of Go. More recently, researchers have applied RL to symbolic mathematics tasks, such as using an RL agent to solve symbolic equations, or solving math problems in an educational curriculum. These successes illustrate that trial-and-error RL methods can learn strategies in vast combinatorial or symbolic search spaces, motivating us to explore RL approaches for the Schur problem.

## 2 Integrating CUDA-Based DFS Exploration into Reinforcement Learning

In Week 8, we successfully implemented a DFS-based coloring search algorithm using CUDA. This approach represented color assignments compactly using binary strings, where each color was encoded using 2-bit values and stored in 64-bit memory blocks. Through shared-memory iterative DFS and bitmask-based conflict checks, we were able to efficiently explore valid colorings from $z = 6$ to $z = 44$ in parallel. The encoding and iterative structure allowed rapid validation and compact memory usage, achieving a 300ms runtime compared to over 200 seconds in Python.

Inspired by this result, we propose to frame our reinforcement learning setup around the CUDA-generated paths:

- **State Representation:** Each coloring sequence (e.g., for $z = 6$ to $z = 44$) is represented as a fixed-length binary string. Each 2-bit value denotes the color assigned to the number at that index.
- **Action Space:** At each timestep, the agent chooses a color (among $k$ options) for the next number.
- **Reward Function:** Each valid assignment yields a reward of $+1$, and a conflict yields a penalty (e.g., $-10$). The goal is to maximize the length of the conflict-free coloring.
- **Data Collection:** Using the CUDA-based code, we can generate thousands of long coloring sequences to bootstrap RL training episodes. These sequences can be labeled with rewards and used for supervised pretraining and policy bootstrapping.

We aim to train a neural network that mimics the structure of AlphaGo's policy-value architecture:

- **Input:** The binary string representing the current partial coloring.

- **Policy Head:** A softmax layer that outputs a probability distribution over color choices.
- **Value Head:** A scalar predicting the expected reward (i.e., how deep the sequence can go before a conflict).

We believe that combining our deterministic CUDA search with reinforcement learning creates a hybrid system: CUDA provides strong offline data for imitation learning, while RL fine-tunes policies through exploration and reward feedback. By leveraging the strengths of both methods, we hope to guide the agent toward generalizing long, conflict-free coloring strategies.

**Next Steps:**

- Refactor the CUDA code to output binary string colorings along with their conflict status and length.
- Use this dataset to pretrain the policy-value network.
- Implement a reinforcement learning training loop using PPO or REINFORCE to continue learning from the pretraining baseline.

## 3 Graph Neural Network (GNN) + Curriculum Learning

### 3.1 Motivation and Overview

The Schur Number problem involves coloring natural numbers $\{1, 2, ..., n\}$ into $k$ color classes such that no monochromatic solution exists for the equation $a + b = c$. Due to the exponential nature of the state space and the sparsity of valid configurations, standard deep RL techniques struggle to generalize. We propose a different approach: using a **Graph Neural Network (GNN)** to model the structural dependencies in the problem, along with **Curriculum Learning** to gradually scale up complexity.

### 3.2 Problem Representation as a Graph

We construct a graph $G = (V, E)$ where:

[itemsep=0pt]Nodes $V$ represent numbers $\{1, 2, ..., n\}$. Edges $E$ represent arithmetic constraints: $a + b = c$.

To simplify learning, we encode the constraint as undirected edges between $a$ and $b$ if $a + b = c$ exists.

### 3.3 Graph Encoding and Node Features

Each node $v_i$ has a feature vector $x_i$ composed of:

[itemsep=0pt]One-hot color assignment vector (size $k$). A binary flag indicating if the node is colored. A normalized position $i/n$. (Optional) A conflict count based on current violations.

The input graph to the GNN is $(X, A)$, where $X \in R^{n \times d}$ and $A$ is the adjacency matrix.

## 3.4 Model Architecture

The GNN has shared message-passing layers followed by two heads:

[itemsep=0pt]**Policy Head:** outputs a softmax distribution over color choices. **Value Head:** predicts the expected return from the current graph state.

Message-passing layers could use GCN, GAT, or MPNN.

## 3.5 Curriculum Learning Strategy

We define a sequence of training stages:

[itemsep=0pt]Stage 1: Solve Schur instance with $n = 9$ (S(3)). Stage 2: Move to $n = 13$ (S(4)). Stage 3: Train on $n = 16+$ (S(5) or synthetic graphs).

The model is progressively fine-tuned across stages, improving generalization.

## 3.6 Reinforcement Learning Setup

[itemsep=0pt]**State:** Current graph $G = (X, A)$ with partial coloring. **Action:** Choose a color for an uncolored node. **Reward:**[itemsep=0pt]

- – $+1$ for valid coloring.
- – $-1$ for invalid assignment.
- – $-0.1\times$ number of violations for partial mistakes.

- **Discount factor:** $\gamma = 0.99$.

## 3.7 Implementation and Training Loop

The training loop is actor-critic based. Below is the pseudocode:

```
initialize GNN_policy , GNN_value
curriculum_stages = [S3_graph , S4_graph , S5_graph]

for stage in curriculum_stages:
    for episode in range(E_max):
        G = deepcopy(stage)
        done = False
        buffer = []

        while not done:
            X, A = encode_graph(G)
            logits = GNN_policy(X, A)
            value = GNN_value(X, A)

            node, color = sample_action(logits, G)
            valid = G.assign_color(node, color)

            reward = compute_reward(G, node, color, valid)
            done = check_done(G)

            buffer.append((G.copy(), node, color, reward, value))

            if not valid:
                done = True

        update_actor_critic(buffer)
```

**Losses:** $\mathrm{L}_{policy} = -\log \pi(a|s) \cdot A(s)$
$\mathcal{L}_{value} = (V(s) - R)^2$
$\mathcal{L}_{entropy} = -\sum_a \pi(a|s) \log \pi(a|s)$ $\mathrm{L} = \mathrm{L}_{policy} + c_1\mathcal{L}_{value} - c_2\mathcal{L}_{entropy}, \quad c_1 = 0.5,\ c_2 = 0.01$

### 3.8 Anticipated Benefits

[itemsep=0pt]**Generalization:** GNNs can learn structural patterns across graphs. **Sample Efficiency:** Curriculum learning avoids early collapse. **Scalability:** Works on variable-sized graphs without padding.

## 4 PPO-Based Actor-Critic (Model-Free)

### 4.1 Motivation and Overview

The Schur Number problem requires assigning integers $\{1, 2, ..., n\}$ to $k$ color classes such that no monochromatic solution exists for $a + b = c$. The problem is combinatorial, with a large state space and sparse valid configurations. Traditional search-based approaches are computationally expensive and difficult to scale.

We explore a **fully model-free reinforcement learning** method using **Proximal Policy Optimization (PPO)** within an **actor-critic framework**. This approach avoids the complexity of tree search (e.g., MCTS), enables end-to-end training using shaped rewards, and allows rapid inference without external search structures.

### 4.2 Environment Setup and State Representation

Each episode corresponds to a single attempt to color numbers $\{1, ..., n\}$ under the Schur constraint.

**State Representation:**

[itemsep=0pt]A tensor $S \in R^{n \times k}$ where each row is a one-hot encoding of a node's color assignment, or all zeros if uncolored. Optional binary mask to track available numbers. Positional encoding: normalized position index $i/n$.

**Observation:** The full vectorized form of current coloring status.
**Action Space:**

[itemsep=0pt]Select a color from $\{1, ..., k\}$ for the next uncolored number in sequence. The policy is conditioned on the current graph state.

## 4.3 Reward Function Design

The reward structure is critical for guiding the agent:

[itemsep=0pt]+1 if all numbers are successfully colored with no violation. $-1$ if a coloring decision causes a monochromatic violation ($a+b=c$). $-0.1$ per constraint conflict at intermediate steps. Small bonus (e.g., +0.01) for every valid assignment.

This encourages the agent to explore deeper states without prematurely terminating.

## 4.4 PPO Architecture

[itemsep=0pt]**Actor Network:** Outputs probability distribution over $k$ color actions. **Critic Network:** Estimates state value for the current coloring. Shared base layers (MLP, CNN, or Transformer).

**Policy Output:** Softmax layer over actions.

**Losses:** $\mathrm{L}_{PPO} = -\min\left(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)$
$\mathcal{L}_{value} = (V(s_t) - R_t)^2$

## 4.5 Training Loop and Pseudocode

```
initialize actor_critic_policy
initialize environment (Schur coloring)

for update in range(num_updates):
    buffer = []
    for episode in range(num_episodes):
        state = env.reset()
        done = False
        while not done:
            action, log_prob, value = actor_critic_policy.act(
   state)
            next_state, reward, done = env.step(action)
            buffer.append((state, action, reward, value, log_prob
   ))
            state = next_state

    advantages = compute_gae(buffer)
    update_policy(buffer, advantages)
```

**Policy Update:** Uses PPO's clipped objective with advantage estimation.

## 4.6 Hyperparameters

[itemsep=0pt]Learning rate: 3e-4 Discount factor ($\gamma$): 0.99 Clipping threshold ($\epsilon$): 0.2 Batch size: 32 PPO update epochs: 4

### 4.7 Advantages of PPO-Based Method

[itemsep=0pt]**Model-Free:** Avoids complexity of MCTS or explicit transition models. **Fast Inference:** Single forward pass at test time. **Flexible Rewards:** Can accommodate shaped or sparse reward regimes. **Efficient Training:** PPO balances performance and stability.

## 5 Conclusion

Throughout our investigation, we explored multiple reinforcement learning paradigms to tackle the Schur Number coloring problem. We began by integrating a CUDA-based DFS search with RL, creating a hybrid strategy that generates valid colorings efficiently and can guide imitation learning.

Next, we proposed using Graph Neural Networks combined with curriculum learning. This approach allows the model to understand structural constraints in the coloring problem and incrementally scale from small to large instances, improving generalization and stability.

Finally, we implemented a fully model-free PPO-based actor-critic framework, leveraging reward shaping and advantage estimation to train efficiently in large combinatorial state spaces.

Each approach offers unique benefits and challenges. Our current progress demonstrates that reinforcement learning can provide valuable heuristics and general strategies for this complex symbolic problem. Further tuning, experimentation, and hybridization of these ideas will continue to push the boundaries of how RL can handle mathematical constraint problems like Schur coloring.