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

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

Contents

Schur Numbers Week 13 Report

May 16, 2025

1 Overview

In the previous week, we examined theoretical reinforcement learning strategies for addressing the Schur numbers problem, without committing to a specific direction. This week, we shift focus to a novel modeling perspective that leverages sequence modeling techniques to frame the problem in a new way. Inspired by recent advances in machine learning, we explore a transformer-based approach to learn from data generated through a parallelized search. Preliminary results suggest that this modeling strategy captures meaningful patterns, setting the stage for further enhancement using reinforcement learning in future work.

2 Data Generation

To generate the data used in training our model, we reused our parallelized brute-force CUDA search algorithm (developed in Week 10). As our modeling strategy suggested, the coloring is represented as binary masks, where each mask corresponds to a specific color. The data generation process involves the following steps:

1. Start with a fixed initial coloring.
2. Sample a certain number (`sample_size`) of samples from each depth level randomly to have a balanced dataset.
3. The data is in binary mask format (5 masks each one with three 64-bit integers), so it's converted to a sequence format. For example, the binary masks `0b1001,0b0110` represent a 2-color coloring of size 4 where numbers 1 and 4 have one color, and numbers 2 and 3 have another color. More data can be found in the `data` directory as `states_zi.txt`.

4. We store the pairs of each depth level data in a separate txt file as `pairs_zi.txt` where `i` is the depth level (The length of the colored sequence). All th data can be found in the `data` directory. Along with the CUDA code used to generate the data.

There was a lot of sequence data format we could have used, but the one we settled for looks like this:

```
[(1,1),(2,2),(3,2),(4,1),(5,3), ...]
```

Where in each tuple, the first element is the number and the second entry is its color. We believed this might be the best representation as it captures the order of the numbers and their colors. Other modifications can be made later on. The code used for the conversion process can be found in the `code` directory as `utils.py`.

3 Transformer Model Overview

With the data formatted as colored sequences, we turn our "attention" to the learning model itself. In this week's work, we introduce a Transformer-based neural network, inspired by its recent success in modeling sequences with complex dependencies. Unlike traditional models like RNNs or LSTMs, Transformers handle sequences in parallel, which fits naturally with our parallel data generation pipeline.

3.1 Model Architecture

Our model, which we refer to as the `SchurTransformer`, is a relatively compact encoder-only Transformer built using PyTorch. It is designed to process variable-length sequences of number-color pairs and predict the color assignment for the next number in the sequence, all while respecting the additive Schur constraints.

The core components of the architecture are:

- **Embeddings:** The input to the model consists of two sequences: the list of numbers and their associated colors. Each is passed through a separate embedding layer: one for the numbers (`num_embed`) and one for the colors (`color_embed`). These embeddings are then concatenated to form a unified representation of each token.
- **Positional Encoding:** Since Transformers have no inherent notion of sequence order, we add a learned positional encoding tensor to the input embeddings. This allows the model to understand the position of each element in the sequence and learn positional dependencies.

- **Transformer Encoder:** The heart of the model is a multi-layer Transformer encoder, implemented using PyTorch's `nn.TransformerEncoder`. We use a relatively small number of heads and layers (e.g., 4 heads, 2 layers) to maintain training efficiency while still capturing meaningful structure.
- **Output Layer:** After encoding, we take the output corresponding to the last token in the sequence and pass it through a linear layer followed by a softmax to produce a probability distribution over the 5 possible colors.

3.2 Constraint-Aware Prediction

To ensure that the model adheres to the additive constraint of the Schur problem, we apply a masking strategy. For each prediction, we construct a binary mask that marks colors violating the constraint $x+y=z$, where both x and y exist in the same color class. This mask is applied to the output logits before the softmax operation to zero out invalid color choices.

3.3 Training Strategy

The model is trained using a standard cross-entropy loss on valid color predictions. Each training example is a prefix of a sequence, and the target is the color of the next number. Padding is applied during batching to handle variable sequence lengths, and training is done using Adam optimizer with default parameters.

3.4 Evaluation

We had no concrete evaluation metrics for the model, but we ran the model on a few sequences and kept predicting the next color and it did incredibly well for values under $z = 80$. We would start with a sequence of length 7 and keep predicting the next color until no more valid colors are left. The model was able to reach sequences of length $40 \sim 50$. Testing code can be found in the `code` directory as `testing.py`. We took 500 samples of sequence for each length from 7 to 70 and averaged the length of the final sequences that's resulted by iteratively predicting the next color. The results are showm in Figure 1. For input < 40 the transformer did a great job, to reach ~ 50 but for larger inputs the model only predicted about 5 valid colors. What's weird is the downward trend from 10 to 30. We think it's a good start for the fine-tuning process that we would discuss in the next section

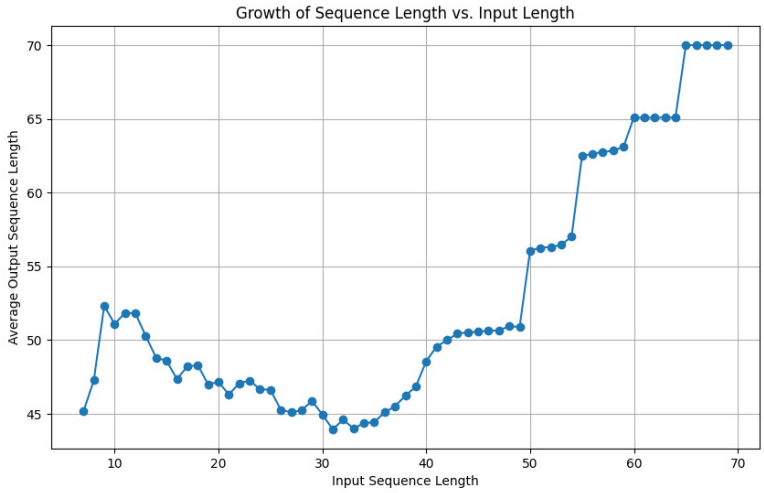

Figure 1: Average length of the final sequences generated by the model for different initial sequence lengths.

4 Future Work

We already implemented our custom RL gym environment and are now looking to integrate it with the transformer model. The idea is to use the transformer model as a policy network and fine-tune it using reinforcement learning. The RL agent will explore the space of colorings and learn to make decisions that maximize the length of the sequence while adhering to the Schur constraints. This approach is expected to yield better performance than the current supervised learning strategy, as it allows for exploration and exploitation of the solution space.