RL: LEARN TO CUT PROJECT REPORT

Leon Li
Columbia University
Computer Science Department
{al4263}@columbia.edu

ABSTRACT

Integer programming, in theory, is an NP-hard problem. In the paper Reinforcement Learning for Integer Programming: Learning to Cut [1], the author used a RL approach to solve the problem that significantly outperforms human-designed heuristics. In this report, we followed the work done by the authors, with some innovations and adaptations, to tackle IP. The adapted methods can (1) tackle even more difficult IP problems and (2) generalize to new instances.

1 Introduction and problems

The problem set up is the following:

$$\min\{c^T x : Ax \leq b, x \geq 0, x \in \mathbb{Z}^n\}$$

The Gomory cuts provide an effective method to solve integer programming. At each iteration, the Gomory cuts algorithm provides a set of possible cuts for us to cut the plane, which is adding more constraints. The goal is to find the optimality gap by adding more cuts, which is to optimize $OPT_{LP} - OPT_{Integer}$ [2] by adding more constraints to the integer programming.

Here, we can formulate the problem in the RL diagram: Let n be the number of variables:

At the $t = 0$, the state space is the set of all constraint: **S** = $\{A_t x \leq b_t\}$ where $\forall a_i \in A_t, a_i \in \mathbb{R}^n$ and $\forall b_i \in b_t, b_i \in \mathbb{R}$.

The action space is the possible new constraints/cuts generated by the Gomory cuts: **A** = $\{E_t x \leq d_t\}$ where $\forall e_i \in E_t, e_i \in \mathbb{R}^n$ and $\forall d_i \in d_t, d_i \in \mathbb{R}$.

The transition from s_t to s_{t+1} is simply adding the cut into the previous state as a new constraint.

Let x^*_{LP} be the optimal solution, the reward is $C^T x^*_{LP}(t+1) - C^T x^*_{LP}(t) \geq 0$. We also use discounted reward with γ to encourage the agent to find the solution early.

2 Algorithm and Training method

2.1 Policy Architecture: LSTM and Attention

The policy π samples an action from the action space each time: $a = \pi(s_t; \theta)$. The policy is a parametrized neural network.The core dilemma is that size of state and action space is not fixed. To solve this problem, we use the LSTM[3] and the Attention Mechanism[4][5]. First of all, we concatenate A and b and call it the constraint matrix, and concatenate E and d call it the candidate matrix. At time step $T = t$, the constraint matrix is a matrix with the shape of $\mathbb{R}^{(m+t)\times(n+1)}$, where $m + t$ is the number of constraints and n is the number of variables. Similarly, for the candidate matrix, it is a matrix with the shape of $\mathbb{R}^{(k)\times(n+1)}$, where k is the number of candidate Gomory cuts. The core idea of the algorithm is to calculate the attention score for each cuts in the candidate with all the constraint in the constraint matrix.[1] And the candidate with the highest attention score will be the most aligned with the set of constraints, which means the cuts with the highest reward.

The implementation is the following: we first use an LSTM with hidden dimension of 32 to embed the constraint matrix and candidate matrix. Here, we use the last hidden cell output of the LSTM as the final embedding. For the embedding, we pass it to a 2 hidden layer neural network with hidden unit of size 64 and Tanh nonlinear activation functions. The output dimension is $k = 16$. Let's call the embedded constraint matrix as H and the embedded candidate matrix as G. The attention score matrix is calculated as

$$S_j = \frac{1}{N_t} \sum_{i=1}^{N_t} g_j^T h_i \forall\, h_i, g_j \text{ in H, G}$$

The action is then simply the argmax over the Softmax of the attention score.

2.2 Policy Training: Policy Gradient

The policy gradient is:

$$\nabla_\phi \rho(\phi) = \mathbb{E}_\pi \big[\sum_{t=0}^{\infty} Q^{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi(s_t, a_t; \theta) \big] [6]$$

To compute the gradient and update $\theta \leftarrow \theta + \alpha \nabla_\theta \rho(\theta)$, where α is the learning rate, we need to find the value of $Q^{\pi_\theta}(s_t, a_t)$. However, it is not analytically solvable, and thus requires us to estimate with either Monte Carlo estimation, or train a Value function as the baseline(Actor-Critic). In this problem, we simply sample one trajectory of cuts, which is one episode with discounted reward with γ, and compute the loss and gradient estimator $\hat{g}$ as the following:

For one trajectory of collected estimated Q_s and a list of state action pairs $(s(t = i), a(t = i))$, the Loss is defined as $-\frac{1}{N} \sum_{i=1}^{N} Q_s(t = i) \log(\pi(s(t = i), a(t = i); \theta))$, and the gradient estimator $\hat{g}$ is:

$$-\frac{1}{N} \sum_{i=1}^{N} Q_s(t = i) \nabla_\theta \log(\pi(s(t = i), a(t = i); \theta) [6]$$

2.3 Evolution Strategies

For the evolution strategies, I did not strictly follow what was described in the paper [1]. The core idea of es is to calculate the noisy estimation of the performance of policy π_θ. Thus we use the following noisy estimation of the Q generated by policy π_θ:

$$Q_i \leftarrow Q_i + \alpha \epsilon_i, \text{ where } \alpha = 0.15 \text{ and } \epsilon_i \sim \mathcal{N}(0, 1)$$

2.4 Curriculum Training

The easy configuration has 10 instances, where the hard configuration has 100 instances. To directly train the policy network on the hard configuration is extremely challenging. Instead, we propose the method of curriculum training: we generate three curriculums consists of 10, 40, 70 instances separately, and we train the same policy network on those three curriculum dataset first, and then train it on the hard configuration.

2.5 Data Processing

Data processing is extremely useful for deep learning tasks. In this task, we want to process both the constraint matrix and the candidate matrix. However, we do not want to lose the relative information about the constraint and the candidate, as numerical relationships are extremely useful. So we concatenate both the constraint and candidate matrix together and then standardize the matrix by subtract the mean and then divide the variance.

2.6 Pseudo code

Here are the algorithm for both easy config training and hard config training. Some terminology: L_C, L_D, L_A, L_R refers to the list to record the constraint, candidates, actions, and rewards. E_r refers to the cumulative episode rewards without discounting. ND refers to not done, which is the condition that $x^*_{LP}(t)$ is not all integer value. C is the state, which is the constraint matrix, and D is the candidate space.

Algorithm 1 Training [1]

Input: initialize a policy π_θ, IP instance parameterized by c, A, b, timelimits T
Hyperparameter: learning rate α, ES random rate σ
for $e_t \in$ episodes **do**
    reset environment, initialize t = 0, E_r = 0, empty lists to record L_C, L_D, L_A, L_R
    $C(0) = [A, b]$, find $x^*_{LP}(0)$ and $D(0) = [E, d]$
    while ND and t < T **do**
        $s_t = \{C(t), D(t), x^*_{LP}(t)\}$
        standardize $C(t)$ and $D(t)$
        compute attention score with $att_{score}(t) = \pi_\theta(C(t), D(t))$
        sample action as $a(t)$ = argmax of softmax($att_{score}(t)$)
        find the reward $r(t)$
        generate $C(t+1)$ by append the cut to $C(t)$
        append $C(t), D(t), a(t), r(t)$ to L_C, L_D, L_A, L_R
        $E_r \leftarrow E_r + r(t)$
        $t \leftarrow t + 1$
    end while
    calculate discounted reward d_r
    calculate Q_s with d_r and ES
    use L_C, L_D, L_A, L_R, Q_s to compute the gradient estimator $\hat{g}$
    $\theta \leftarrow \theta + \alpha\hat{g}$
end for
return θ

The algorithm for curriculum training is:

Algorithm 2 Curriclum Traning

Input: initialize a policy π_θ, Curriclums, hard config
for curriculum$(i) \in$ Curriclums **do**
    Train the network π_θ with the Training algorithm above
end for
Train the network π_θ with hard config
return θ

The roll out of the policy:

Algorithm 3 Roll out of policy

Input: a trained policy π_θ, IP instance parameterized by c, A, b, time limits T
reset environment, initialize t = 0, E_r = 0
$C(0) = [A, b]$, find $x^*_{LP}(0)$ and $D(0) = [E, d]$
while ND and t < T **do**
    $s_t = \{C(t), D(t), x^*_{LP}(t)\}$
    standardize $C(t)$ and $D(t)$
    compute attention score with $att_{score}(t) = \pi_\theta(C(t), D(t))$
    sample action as $a(t)$ = argmax of softmax($att_{score}(t)$)
    find the reward $r(t)$
    generate $C(t+1)$ by append the cut to $C(t)$
    $E_r \leftarrow E_r + r(t)$
    $t \leftarrow t + 1$
end while
return E_r

3 Result

3.1 easy configuration

Here is the result for easy configuration:

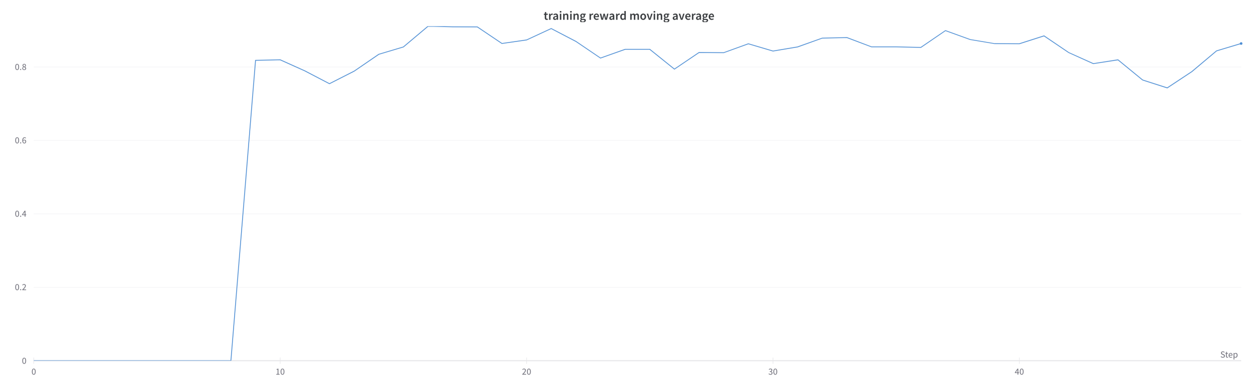

Figure 1: Averaged Moving Reward within 10 steps for easy configuration

3.2 hard configuration

Here is the result for easy configuration:

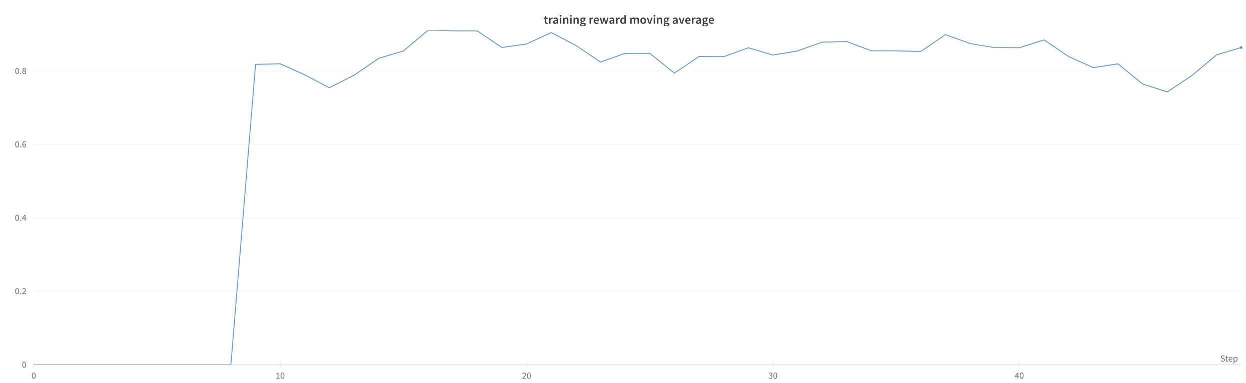

Figure 2: Averaged Moving Reward within 10 steps for easy configuration

3.3 test configuration

Here is the result for easy configuration:

References

[1] Shipra Agrawal Yunhao Tang and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. *International Conference on Machine Learning*, 2020.

[2] Yunhao Tang Shipra Agrawal, Abhi Gupta. Course project: Using rl to solve integer programming.

[3] Jürgen Schmidhuber Sepp Hochreiter. Long short-term memory. *Neural computation*, 1997.

[4] Niki Parmar Jakob Uszkoreit Llion Jones Aidan N Gomez Łukasz Kaiser Illia Polosukhin Ashish Vaswani, Noam Shazeer. Attention is all you need. *Advances in neural information processing systems*, 2017.

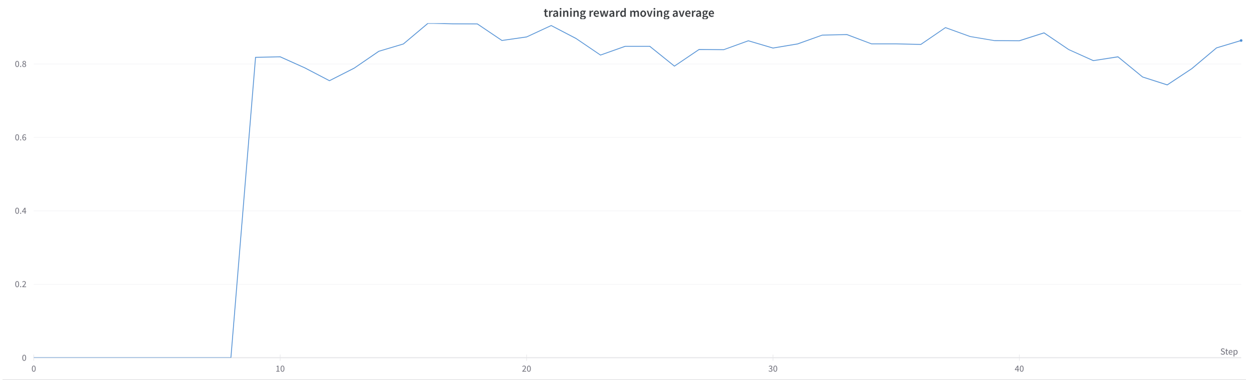

Figure 3: Averaged Moving Reward within 10 steps for easy configuration

[5] Yoshua Bengio Dzmitry Bahdanau, Kyunghyun Cho. Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations, ICLR 2015 - San Diego, United States*, 2015.

[6] Shipra Agrawal. Lecture 4 policy gradient methods. *IEOR 8100 Lecture notes*.