Learning to Cut with Policy Gradient

Shuibenyang Yuan

Abstract

Integer programming (IP) is a well known NP-complete problem which involves many applications like: production planning, scheduling, and territorial partitioning, etc. Solving such integer programming problems is hard, and modern IP solvers are often human-designed and tuned using experience and data. This paper presents a deep reinforcement learning (RL) algorithm to make selection of Gomory's Integer cuts to make solving IPs more efficiently(Gomory, 1960).

1 Introduction

Integer programming problems are NP-complete in theory and hard to solve in practice as they cannot be guaranteed solved with polynomial time. There are different IP solvers exist built upon efficient heuristics for solving some specific sub-problem of IPs. With arise of Machine Learning(ML) techniques, the application of ML to different optimization problems has been a topic of significant interests in recent years. In this paper, we build a reinforcement learning(RL) agent to routinely make selection of different available cuts provided by Gomory's cut(Gomory, 1960), which adds an appropriate Gomory's cut terminates in a finite number of iteration. We use tried two deep learning techniques, seq2seq and attention, to make the choice of Gomory's cuts to improve on performing cuts.

2 Related Work

The related work of this paper refers to everything on Learning to Cut(Tang, 2020).

3 Methods

3.1 Formulating Cutting Plane selection as RL

To formulate cutting plane selection as RL problem, we adopt ideas from Learning to Cut(Tang, 2020).
In brief, we convert cutting plane problem with an MDP: at time step $t \geq 0$, an agent is in a state $s_t \in S$, taking an action $a_t \in A$ with an instant reward $r_t \in \mathbb{R}$, and then, the agent transition to the next state $s_{t+1} \sim p(\cdot|s_t, a_t)$.
A policy $\pi : S \to P(A)$ gives a mapping from any state to a distribution over action $\pi(\cdot|s_t)$, and the objective of RL is to maximizes the expected cumulative rewards, i.e., $max_\pi J(\pi) := E_\pi(\sum_{t=1}^{T} \gamma^t r_t)$ where γ is the discounted factor of the expectation of cumulative rewards. The state space S is defined as $s_t = \{A_t, b_t, c, D_t, e_t\}$, where is the constraint matrix of current LP $\max_x c^T x$ s.t $A_t x \leq b_t$. D_t, e_t are Gomory cuts available in the current t of Gomory's cutting plane algorithm (Gomory, 1960).
The reward $r_t = c^T x^*_{LP}(t+1) - c^T x^*_{LP} \geq 0$, which can be described the gap between objective values of consecutive LP solutions.

3.2 States parametric function

We use the same parametric function design of Learning to Cut(Tang, 2020). We define our parametric function $F_\theta : R^{n+1} \to R^k$ for some given k. In specific, we compute

$$H_t = F_\theta([A_t, b_t]) \tag{1}$$

$$G_t = F_\theta([D_t, e_t]) \tag{2}$$

Here, $[\cdot, \cdot]$ denotes concatenation.

3.3 Network for order-agnostic cut selection

As we do not know for what instance of LP problem will be cut in next state, it is necessary that the architecture is agnostic to the ordering among the constraint as the ordering does not reflect the geometry of the feasible set.
We build two different networks to transform it as a sequence network to carry over state-by-state information.

3.3.1 seq2seq network

Inspired by Sequence to Sequence Learning (Sutskever, 2014), we aim to carry the weights of current state into the weights of next state, and parametric the weights with a function $G_\theta : R^{n+1} \to R^m$ for some given m.
We compute

$$W_t = G_\theta([A_t, b_t]), M_t = G_\theta([A_{t-1}, b_{t-1}]) \quad (3)$$

$$U_t = G_\theta([D_t, e_t]), V_t = G_\theta([D_{t-1}, e_{t-1}]) \quad (4)$$

Then, we compute

$$x_j^{(t)} = \sigma(\frac{1}{N_{t-1}} \sum_{i=1}^{N_{t-1}} W_j^{(t)T} M_j^{(t)}) \quad (5)$$

$$y_j^{(t)} = \sigma(\frac{1}{K_{t-1}} \sum_{i=1}^{K_{t-1}} U_j^{(t)T} V_j^{(t)}) \quad (6)$$

where N_{t-1} and K_{t-1} are the number of rows of A_{t-1}, D_{t-1} respectively, and σ indicates an activation function of choice. In our case, we use $softmax$ as our activation function.

3.3.2 recurrent attention network

We adopt ideas from the attention network (Vaswani, 2017) with a recurrent network to build the policy network.
The recurrent function is defined as $\phi(F_\theta(x), h_{t-1}) : (R^k, R^m) \to (R^k, R^m)$ where F_θ is the state parametric function defined above, and ϕ is the recurrent function, h_t is the hidden state of the recurrent neural network, then the function output can be computed as:

$$U^{(t)}, h_x^{(t)} = \phi(H^{(t)}, h_x^{(t-1)}) \quad (7)$$

$$V^{(t)}, h_y^{(t)} = \phi(G^{(t)}, h_y^{(t-1)}) \quad (8)$$

Note that, the hidden state weights are carried over on the whole process of training iteratively.
Then, We compute

$$W_t = G_\theta([A_t, b_t]), M_t = G_\theta([U^{(1)}, ..., U^{(t-1)}]^T) \quad (9)$$

$$U_t = G_\theta([D_t, e_t]), V_t = G_\theta([V^{(1)}, ..., V^{(t-1)}]^T) \quad (10)$$

Note that, M_t, V_t only contains concatenation of Us, Vs in this iteration.
Then, we compute

$$x_j^{(t)} = \sigma(\frac{1}{N_t} \sum_{i=1}^{N_t} W_j^{(t)T} M_j^{(t)}) \quad (11)$$

$$y_j^{(t)} = \sigma(\frac{1}{K_t} \sum_{i=1}^{K_t} U_j^{(t)T} V_j^{(t)}) \quad (12)$$

where $[\cdot, ..., \cdot]$ is the function of concatenation of all the matrices, and N_t is the sum of number of rows of $A_1, ..., A_{t-1}$, and K_t is the sum of number of rows of $D_1, ..., D_{t-1}$. σ indicates an activation function of choice. In our case, we use $tanh$ as our activation function.

3.4 Policy Network Architectures

As defined above, with policy $\pi_\theta(a_t|s_t)$, given state s_t, a policy π_θ will return a categorical probability distribution $P_t \in N^t$ where N^t is the number of available cuts at time t.
Compute P_t with seq2seq network how P_t is computed is specified in Algorithm 2 in Appendix A
Compute P_t with recurrent attention network how P_t is computed is specified in Algorithm 3 Appendix A
Policy rollout How policy network pull out probability distribution is specified Algorithm 4 Appendix A

3.5 Training: Policy Gradient

We train the RL agent using policy gradient(PG), which relies on an estimated $\hat{g}_t$ computed by

$$\hat{g}_t = \sum_{t=0}^{T-1} r_t \gamma^t + \epsilon_t \quad (13)$$

Algorithm 1: training on θ

Input: I: number of iterations ,N: number of trajectories
Result: θ
Initialize θ at random;
Initialize $i \leftarrow 0$;
while $i < I$ **do**
  $n \leftarrow 0$;
  $\nabla_\theta \leftarrow 0$;
  while $n < N$ **do**
    $P_n^{(i)}, R_n^{(i)} \leftarrow record$ P_t, R_t with policy rollout;
    $G_n^{(i)} \leftarrow$ record g_t with $R_n^{(i)}$;
    $\nabla_\theta \leftarrow \nabla_\theta + \alpha(-\log P_n^{(i)} \cdot G_n^{(i)} - \log P_n^{(i)} \cdot P_n^{(i)})$;
    $n \leftarrow n + 1$
  end
  $\theta \leftarrow \theta + \frac{1}{N}\nabla_\theta$;
  $i \leftarrow i + 1$
end

Note that $\epsilon_t \sim N(0, \sigma)$, where σ is a tuned parameter for standard deviation of noise ϵ_t.
The process of training is described in Algorithm 1. It averages the gradient after number of trajectories reaches the limit N at each iteration.

4 Experiments

We train each policy network with fine tuned parameters for on training with $[1, 4, 12]$ trajectories, and we evaluate our approach with different metrics. Specifically, we evaluate our policy network with following aspects:

- The training curves of each policy network with training on different trajectories.
- The training best rewards of each policy network with training on different trajectories.
- The test rewards for unseen instances of each policy network with training on different trajectories.
- The training time cost due to model complexity.

Baseline We compare the performance of our RL agents with uniform random policy: select cut randomly in every state. As all the available cuts are generated by Gomory's, which are guaranteed with a theoretical convergence in finite time.
Setup We train our policy network on two instances: a easy instance with 10 different IP problems with 60 variables to solve, and a hard instance with 100 different IP problems with 60 variables to solve. For evaluation, we test our policy network based on the weights trained in the last iteration. We evaluate easy-instance-trained policy on hard instance and another test instance with 100 different unseen IP problems. Similarily, we evaluate hard-instance-trained policy on easy instance and another test instance with 100 different unseen IP problems.
Implementation detail We used pre-build MDP simulation environment for RL using Gurobi (Gurobi Optimization, 2015) developed by the authors of learning to cut (Tang, 2020). For training, we sample actions from the categorical distribution with the current policy, save the policy netowork's weights with 20 iterations and 50 cuts per iteration. For testing, we process 12 iterations to evaluate each trained policy with greedy action $a^* = argmax_i p_i$ on seen instance (training instance) and unseen instances, and each iteration also performs 50 cuts. After the process, we record each policy's performance by taking mean of the 12 iterations dropping the maximum and minimum.

4.1 Training Curve

Figure 1 presents the plot of training reward on easy instance and hard instance of two policy networks and baseline in a 3 rolling average window. From the plot, we observe that both attention and seq2seq network outperform the random policy cuts significantly: random cuts' training rewards goes between $0.2 - 0.4$ training reward on both easy instance and hard instance, but others are significantly higher than $0.2 - 0.4$. On training easy instance, both of attention seq2seq net work converge to around 0.8 training reward. On training hard instance, we found that training on more trajectories will get worse training reward:

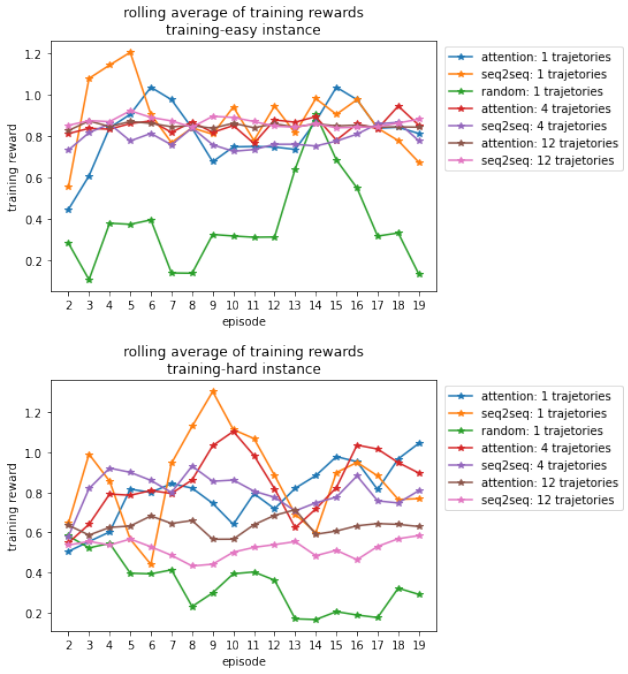

Figure 1: training curve

the lower the counts of trajectory is, the higher average the training reward can be, which contradicts our initial hypothesis that more trajectories may improve the policy network.

4.2 Best Training Rewards

With Figure 1, we observed both 1-trajectory seq2seq policy networks have the highest training rewards on both easy and hard instances, but both 12-trajectory seq2seq policy networks have the lowest training rewards on both easy and hard instances. We can also see the higher variance on training with seq2seq policy network. So, seq2seq policy networks tend to have higher variance on training stage than attention policy networks. We can see more details about this in Figure 3 in Appendix A.

4.3 Test Rewards

The Figure 2 presents the performance of each policy network on unseen instances. From the table, we observe that both policy network also outperform the random policy significantly: the average testing reward of random policy stays around 0.35, and all of other

			easy_config	hard_config	test_config
model	**training_instance**	**num_trajetories**			
seq2seq	**training-easy**	**1**	NaN	0.624571	0.495289
		4	NaN	0.653427	0.847902
		12	NaN	1.017011	0.592489
	training-hard	**1**	0.698237	NaN	0.654271
		4	0.677572	NaN	0.709378
		12	0.807179	NaN	0.685687
attention	**training-easy**	**1**	NaN	0.516764	0.776047
		4	NaN	0.702680	0.623918
		12	NaN	0.719894	0.761169
	training-hard	**1**	0.728543	NaN	0.753973
		4	0.489844	NaN	0.469277
		12	0.787593	NaN	0.797781
random	**N/A**	**1**	0.371029	0.348695	0.364410

Figure 2: Test Rewards

policy network have larger testing rewards for unseen instances. In addition, we see that seq2seq policy network have larger variances on unseen instances than attention policy network.

4.4 Training Time

Because attention policy network will traverse additional attention weights with all experiences in this iteration, the time cost on training of attention policy network will be significantly larger than that of seq2seq policy network. More detail of time comparison can be found in Figure 4 in Appendix A.

5 The Conclusions

We presented a policy gradient deep RL technique to learn from cutting plane strategy for solving IP instances, by solving a set of training instances iterative. The RL agent trained has significantly improvement on the strategy with uniform random cut provided by Gomory's cuts (Gomory, 1960).

References

Gomory, R. (1960). An algorithm for the mixed integer problem. Technical report, RAND CORP SANTA MONICA CA.

Gurobi Optimization, I. (2015). Gurobi optimizer reference manual. *URL http://www. gurobi. com.*

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112

Tang Y., Agrawal S., Faenza Y.(2020). Reinforcement Learning for Integer Programming: *Learning to Cut*. *arXiv: 1906.04859*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

A Appendix

Algorithm 2: compute P_t with seq2seq network

Input: $S_t = \{A_t, b_t, D_t, e_t\}$
Result: P_t
compute x_t, y_t;
$X_t \leftarrow [F_\theta([A_t, b_t]), x_t]$;
$Y_t \leftarrow [F_\theta([D_t, e_t]), y_t]$;
$S_t \leftarrow \frac{1}{N_t} \sum_{i=1}^{N_t} Y_t^T X_t$;
$P_t \leftarrow softmax(S_t)$;

Algorithm 3: compute P_t with attention network

Input: $\mathrm{S}_t = \{A_t, b_t, D_t, e_t\}$
Result: P_t
compute x_t, y_t;
$X_t, h_t \leftarrow \phi_\theta([A_t, b_t], h_{t-1})$;
$Y_t, h_t \leftarrow \phi_\theta([D_t, e_t], h_{t-1})$;
$X_t \leftarrow [X_t, x_t]$;
$Y_t \leftarrow [Y_t, y_t]$;
$S_t \leftarrow \frac{1}{N_t} \sum_{i=1}^{N_t} Y_t^T X_t$;
$P_t \leftarrow softmax(S_t)$;

Algorithm 4: policy rollout

Input: C for cutlimit, policy network $\pi(\theta)$
Result: Write here the result
$t \leftarrow 0$;
initialize A_t, b_t, D_t, e_t;
while $t < C$ **do**
  $s_t \leftarrow \{A_t, b_t, D_t, e_t\}$;
  $P_t \leftarrow \pi_\theta(s_t)$;
  $a_t \leftarrow$ sample based on P_t;
  $A_{t+1}, b_{t+1}, D_{t+1}, e_{t+1} \leftarrow$ based on a_t;
  Compute reward r_t;
  $t \leftarrow t + 1$
end

			best_training_rewards
model	**num_trajetories**	**training_instance**	
attention	**1**	**training-easy**	1.115349
		training-hard	1.305517
	4	**training-easy**	1.066773
		training-hard	1.188934
	12	**training-easy**	0.931515
		training-hard	0.856213
random	**1**	**training-easy**	1.115064
		training-hard	1.115064
seq2seq	**1**	**training-easy**	1.607248
		training-hard	1.767083
	4	**training-easy**	0.963572
		training-hard	1.173287
	12	**training-easy**	0.955852
		training-hard	0.707421

Figure 3: Best Training Rewards

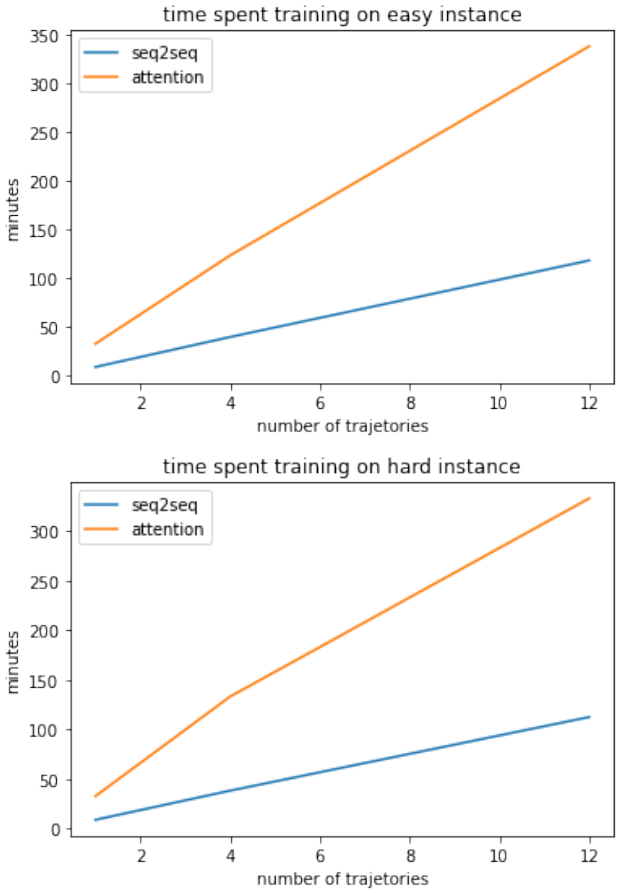

Figure 4: training time