Reinforcement Learning in E

Jack McKeown and Geoff Sutcliffe

University of Miami, Miami, Florida, U.S.A.
jam771@miami.edu, geoff@cs.miami.edu

1 Introduction

Modern saturation based theorem provers such as E [8] and Vampire [5] rely on heuristics to guide their search. Historically, manually engineered heuristics have been the norm. More recently, advances in machine learning have inspired attempts to leverage that power for guiding search. While most efforts have used supervised learning, reinforcement learning (RL) has also been successfully applied [3, 1, 6]. This work aims to incorporate RL into E in a way that generalizes the approach taken by E's `--auto` mode. In the words of E's documentation:

> Clause selection is determined by a heuristic evaluation function, which conceptually sets up a set of priority queues and a weighted round robin scheme that determines from which queue the next clause is to be picked.

The order of each priority queue is dictated by its corresponding Clause Evaluation Function (CEF). When E is invoked on a problem in `--auto` mode, E analyzes the problem and selects a set of CEFs and a weighted round-robin *CEF-schedule*. The CEF-schedule is used for all given clause selections throughout the proof attempt. The approach taken in this work is to replace the CEF-schedule with an external mapping from the state of E to a CEF to use for the next given clause selection. When this mapping is allowed to be stochastic, it can be thought of as mapping the state of E to a probability distribution over CEFs.

2 Reinforcement Learning Framing

The *policy* of an RL agent is a mapping from the *state* of the RL *environment* to a categorical probability distribution over the available *action*s. When queried with a state, s, the agent samples an action, a, from the distribution $\pi_\theta(s)$, and receives a *reward* from the environment.

Previous approaches to RL for ATP have mostly been *tableaux-based* where the state features are derived from the tableaux and actions are tableaux extension steps [3]. The latest *saturation-based* approaches to RL for ATP essentially represent state as the clauses in the processed set and represent actions as the potential given clauses from the unprocessed set [1, 6]. These sets of clauses are encoded via neural networks such as Graph Neural Networks. Rewards are typically given for completing proofs.

In this work, reinforcement learning is incorporated into E as follows: E is invoked on a problem with a fixed set of CEFs. The state of E is sent to the agent each time E needs to select a given clause. At the time of writing, the policy being used, π_θ, is implemented by a shallow neural network with ReLU activation on the hidden layers and a softmax activation function on the output layer. The state consists of 4 features: the number of clauses and average clause weight within the unprocessed and processed sets, but many possible features exist. The agent responds with one of the CEFs as its action. E chooses a given clause using the corresponding CEF. If the selection completes a proof then the reward is one, otherwise the reward is zero. The

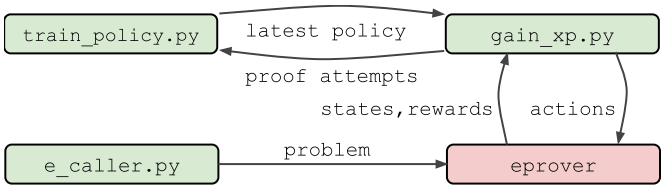

Figure 1: Experimental Setup and Training Architecture

parameters of the policy are learned via Monte-Carlo policy gradients (aka REINFORCE [9]). Originally, an epsilon-greedy strategy with Q-learning was considered, but was rejected due to concerns that the agent would heavily favor one CEF in underexplored regions of state-space.

3 Architecture

The architecture shown in Figure 1 is used to train π_θ:

1. A python script `e_caller.py` repeatedly invokes E on problems from the "bushy" problems of the MPTP2078 dataset [2]. This script allows for control of the sampling distribution over problems and can be used to focus training on easy or hard problems.
2. A python script `gain_xp.py` receives states and rewards from E and sends actions to E. It chooses actions via a saved policy and saves completed proof attempts to disk. All communication between E and `gain_xp.py` is performed using named pipes.
3. A python script `train_policy.py` continually updates the policy used by `gain_xp.py` using the saved proof attempts.

Multiple instances of `e_caller` and `gain_xp` are run in parallel to speed up learning.

4 Initial Results

For the results shown in Table 1, a constant list of 75 CEFs was extracted from the CEF-schedules employed by E's `--auto` mode. `e_caller.py` was configured to try all MPTP2078 bushy problems, and E is invoked with a timeout of 20 seconds. To see if any learning occurs, a uniform distribution over the 75 CEFs (ignoring state) is used as a baseline policy. The simplest extension of this policy is to learn constant probabilities with which to choose each CEF (still ignoring state). Next, a simple neural network that takes state into account is considered. Finally, these policies are compared to E's `--auto` mode.

Approach	**Solved**
Uniform Distribution	1105
Learned Distribution	1105
Simple Neural Network	1110
E –auto	1156

Table 1: Number of MPTP2078 Bushy Problems solved by different approaches

5 Concerns and Future Work

There are various reasons why E's auto mode still has an advantage over the approaches presented here. The most apparent reasons are that E's auto mode has a much richer space of features, and that the neural network used in these experiments is very simple. A more insidious reason is that the CEF-schedules used in E's `--auto` mode were explicitly evolved to solve more problems, whereas the policy gradient training directly favors finding proofs more quickly. From an RL perspective, it seems that the goal of quickly solving easy problems may be in conflict with the goal of solving hard problems. Perhaps oversampling hard problems in `e_caller.py` could help with this. Alternatively, the policy gradient loss could be scaled by some notion of problem difficulty. Future work will also explore Actor-Critic [4] models, which typically outperform REINFORCE.

It is unclear whether CEF choices are an expressive enough action space for improving guidance using RL. *This* work opted for CEFs as actions because it is convenient, and perhaps more sample efficient, to have a consistent set of available actions. Another reason for using CEFs is that it side-steps the issue of having to represent clauses as input for neural networks. (Graph Neural Networks, Recurrent Neural Networks, and Recursive Neural Networks have all been used with varying degrees of success for this [7].) Despite these theoretical benefits to using CEFs as actions, with the exponential growth of the unprocessed set during a proof attempt, the CEFs can only represent a small fraction of the available given clause selections. If the best clause to select is not preferred by any of the CEFs, then it is impossible to select. One might also be concerned that 75 CEFs is too many for a first attempt at learning a CEF-schedule. As a response to this, a smaller list of 7 CEFs was established. These 7 CEFs were chosen to be very different from one-another in order to retain an expressive choice of action. Despite this, the results were qualitatively similar to the results in Table 1. The only appreciable difference was that with 7 CEFs, the models peaked at a much worse performance of around 43% problems proved.

References

[1] I. Abdelaziz, M. Crouse, et al. Learning to Guide a Saturation-Based Theorem Prover. *CoRR*, abs/2106.03906, 2021.

[2] J. Alama, D. Kühlwein, E. Tsivtsivadze, J. Urban, and T. Heskes. Premise Selection for Mathematics by Corpus Analysis and Kernel Methods. *CoRR*, abs/1108.3446, 2011.

[3] C. Kaliszyk, J. Urban, H. Michalewski, and M. Olsák. Reinforcement Learning of Theorem Proving. *CoRR*, abs/1805.07563, 2018.

[4] V. Konda and J. Tsitsiklis. Actor-Critic Algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[5] L. Kovacs and A. Voronkov. First-Order Theorem Proving and Vampire. In *Proceedings of the 25th International Conference on Computer Aided Verification*, number 8044 in Lecture Notes in Artificial Intelligence, pages 1–35. Springer-Verlag, 2013.

[6] G. Reger M. Rawson, A. Bhayat. Reinforced External Guidance for Theorem Provers. 2020.

[7] J. McKeown. Clause Representation for Proof Guidance using Neural Networks, 2021.

[8] S. Schulz, S. Cruanes, and P. Vukmirovic. Faster, Higher, Stronger: E 2.3. In *Proceedings of the 27th International Conference on Automated Deduction*, number 11716 in Lecture Notes in Computer Science, pages 495–507. Springer-Verlag, 2019.

[9] R. J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8:229–256, 2004.