Reinforcement Learning for leanCoP

Cezary Kaliszyk    Josef Urban
Henryk Michalewski    Mirek Olšák

AITP 2018

March 28, 2018
Automated Theorem Proving

Historical dispute: Gentzen and Hilbert

- Today two communities: Resolution (-style) and Tableaux

Possible answer: What is better in practice?

- Say the CASC competition or ITP libraries?
- Since the late 90s: resolution (superposition)

But still so far from humans?

- We can do learning much better for Tableaux
- And with ML beating brute force search in games, maybe?
leanCoP: Lean Connection Prover

Connected tableaux calculus
  - Goal oriented, good for large theories

Regularly beats Metis and Prover9 in CASC (CADE ATP competition)
  - despite their much larger implementation

Compact Prolog implementation, easy to modify
  - Variants for other foundations: iLeanCoP, mLeanCoP
  - First experiments with machine learning: MaLeCoP

Easy to imitate
  - leanCoP tactic in HOL Light
Lean connection Tableaux

Very simple rules:

- **Extension** unifies the current literal with a copy of a clause
- **Reduction** unifies the current literal with a literal on the path

\[
\begin{align*}
\text{Axiom} \quad \emptyset, M, \text{Path} \\
\text{Reduction} \quad C, M, \text{Path} \cup \{L_2\} & \rightarrow C \cup \{L_1\}, M, \text{Path} \cup \{L_2\} \\
\text{Extension} \quad C_2 \setminus \{L_2\}, M, \text{Path} \cup \{L_1\} & \rightarrow C \cup \{L_1\}, M, \text{Path}
\end{align*}
\]
First experiment: MaLeCoP in Prolog

Select extension steps
- Using external advice

Slow implementation
- 1000 less inf per second

Can avoid 90% inferences!
What about efficiency: FEMaLeCoP

Very simple but very fast classifier
- Naive Bayes (with optimizations)

Approximate features and multi-level indexing
- Offline indexing
- Indexing for the current problem
- Discrimination tree stores NB data

Consistent clausification and skolemization

Performance is about 40% of non-learning leanCoP speed
- A few more theorems proved (3–15%)
What about stronger learning?

Yes, but...

- If put directly, huge times needed
- Still improvement small

NBayes vs XGBoost on 2h timeout

Preliminary experiments with deep learning

- So far quite slow
Is theorem proving just a maze search?
Is theorem proving just a maze search?
Is theorem proving just a maze search?

Yes and NO!

- The proof search tree is not the same as the tableau tree!
- Unification can cause other branches to disappear.

Provide an external interface to proof search

- Usable in OCaml, C++, and Python
- Two functions suffice
  
  \[
  \text{start} : \text{problem} \rightarrow \text{state}
  \]
  
  \[
  \text{action} : \text{action} \rightarrow \text{state}
  \]

- where

  \[
  \text{state} = \langle \text{action list} \times \text{goal} \times \text{path} \times \text{remaining} \rangle
  \]
Is it ok to change the tree?

Most learning for games sticks to game dynamics
- Only tell it how to do the moves

Why is it ok to skip other branches
- Theoretically ATP calculi are complete
- Practically most ATP strategies incomplete

In usual 30s – 300s runs
- Depth of proofs with backtracking: 5–7 (complete)
- Depth with restricted backtracking: 7–10 (more proofs found!)

But with random playouts: depth hundreds of thousands!
- Just unlikely to find a proof → learning
Monte Carlo First Try: MonteCoP

Use Monte Carlo playouts to guide restricted backtracking

- Improves on leanCoP, but not by a big margin
- Potential still limited by depth

Can we do better?

- Arbitrarily long playouts
- Learn from the playouts
How to search a tree?

- Given some prior probabilities
- Given success (fail) rates so far

**UCT: Select node $n$ maximizing**

$$\frac{w_i}{n_i} + c \cdot p_i \cdot \sqrt{\frac{\ln N}{n_i}}$$

**Intuition**

- Initially proportional to the prior
- Later win ratio dominates
- We will learn the win ratio
Monte Carlo Tree Search + Upper Confidence Bounds for Trees

[Szepesvari 2006]

How to search a tree?

- Given some prior probabilities
- Given success (fail) rates so far

**UCT:** Select node $n$ maximizing

$$\frac{w_i}{n_i} + c \cdot p_i \cdot \sqrt{\frac{\ln N}{n_i}}$$

Intuition

- Initially proportional to the prior
- Later win ratio dominates
- We will learn the win ratio

---

MCTS tree for t9_zfmisc_1

<table>
<thead>
<tr>
<th>prior, $p_i$</th>
<th>$\frac{w_i}{n_i}$</th>
<th>visits, $n_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>0.799</td>
<td>10000</td>
</tr>
<tr>
<td>0.17</td>
<td>0.606</td>
<td>5625</td>
</tr>
<tr>
<td>0.64</td>
<td>0.719</td>
<td>4713</td>
</tr>
<tr>
<td>0.36</td>
<td>0.023</td>
<td>912</td>
</tr>
<tr>
<td>0.08</td>
<td>0.013</td>
<td>622</td>
</tr>
<tr>
<td>0.20</td>
<td>0.014</td>
<td>76</td>
</tr>
<tr>
<td>0.32</td>
<td>0.024</td>
<td>113</td>
</tr>
<tr>
<td>0.08</td>
<td>0.011</td>
<td>68</td>
</tr>
<tr>
<td>0.10</td>
<td>0.007</td>
<td>5</td>
</tr>
</tbody>
</table>
Learn Policy: Which actions to take?

Even for a single problem

- If we know that some branches failed
- We can avoid such branches in other parts of the “maze”

Playouts following UCT

- After a number of playouts, select the most visited branch
- Only continue inside that branch (called **big step**)

A sequence of big steps ends in a proof, dead end, or is too long

- We can either way learn which actions were chosen
- With some initial win heuristic (remaining goals, size, constant)
Learn Value: How likely is a proof state to be provable?

Learn from all bigstep states

- One if theorem, zero otherwise
Learn Value: How likely is a proof state to be provable?

Learn from all bigstep states

- One if theorem, zero otherwise

With 150K good value training samples and 250K good policy training samples

- XGBoost policy train time: 4 min, Value train time: 8 min
- 2000 problems run with 100K inferences, no bigsteps

<table>
<thead>
<tr>
<th></th>
<th>time (min)</th>
<th>Theorems</th>
</tr>
</thead>
<tbody>
<tr>
<td>No learning</td>
<td>1.5</td>
<td>440</td>
</tr>
<tr>
<td>Only learn values</td>
<td>5.0</td>
<td>535</td>
</tr>
<tr>
<td>Only learn policy</td>
<td>10.5</td>
<td>790</td>
</tr>
<tr>
<td>Learn both</td>
<td>11.5</td>
<td>871</td>
</tr>
</tbody>
</table>
Reinforcement from scratch

Starting with no data, and with 1500 playouts per bigstep

<table>
<thead>
<tr>
<th>round</th>
<th>thms</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td>665</td>
</tr>
<tr>
<td>1</td>
<td>654</td>
</tr>
<tr>
<td>2</td>
<td>718</td>
</tr>
<tr>
<td>3</td>
<td>727</td>
</tr>
<tr>
<td>4</td>
<td>754</td>
</tr>
<tr>
<td>5</td>
<td>748</td>
</tr>
<tr>
<td>6</td>
<td>769</td>
</tr>
<tr>
<td>7</td>
<td>760</td>
</tr>
<tr>
<td>8</td>
<td>776</td>
</tr>
<tr>
<td>9</td>
<td>776</td>
</tr>
<tr>
<td></td>
<td>.......</td>
</tr>
<tr>
<td>10</td>
<td>782</td>
</tr>
<tr>
<td>11</td>
<td>797</td>
</tr>
<tr>
<td>12</td>
<td>796</td>
</tr>
<tr>
<td>13</td>
<td>800</td>
</tr>
<tr>
<td>14</td>
<td>795</td>
</tr>
<tr>
<td>15</td>
<td>794</td>
</tr>
<tr>
<td>16</td>
<td>792</td>
</tr>
<tr>
<td>17</td>
<td>804</td>
</tr>
<tr>
<td></td>
<td>......</td>
</tr>
<tr>
<td>29</td>
<td>815</td>
</tr>
<tr>
<td>30</td>
<td>820</td>
</tr>
</tbody>
</table>
Conclusion

- Reinforcement learning on small Mizar data project
  - UCT, action, value work in connection based setup
  - Learning from scratch can work even for a single problem

- Lots of things to try
  - Other cost functions
  - Other learning frameworks
  - Larger experiments