universität innsbruck

Machine Learning for Theorem Proving

Lecture 10 (VU)

Cezary Kaliszyk

Overview

Last Lecture

- ATP foundations
- learning for superposition calculus
- Enigma
- deep learning for E-prover

Today

- lean connection calculus
- reinforcement learning for leanCoP

Automated Theorem Proving

Historical dispute already from the times of Gentzen and Hilbert

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

Possible answer: What is better in practice?

- Say the competitions or on existing ITP libraries?
- Since the late 90s: resolution (superposition) has been winning

But still so far from humans?

- A human can more naturally think in tableaux style as the proof state is much smaller. Sometimes proofs are a bit bigger, but it does not matter if we can guide it better.
- Thus, we can do machine learning much better for Tableaux
- And with ML beating brute force search in games, maybe better than resolution?

leanCoP: Lean Connection Prover

[*Otten 2010*]

We introduce tableaux only by one single calculus and system

Connected tableaux calculus

- Goal oriented, good for large theories

Regularly beats Metis and Prover9 in CASC

(CADE ATP Systems 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

$$\frac{}{\{\},\ M,\ \mathit{Path}}\quad \mathit{Axiom}$$

$$\frac{C,\ M,\ \mathit{Path} \cup \{L_2\}}{C \cup \{L_1\},\ M,\ \mathit{Path} \cup \{L_2\}}\quad \mathit{Reduction}$$

$$\frac{C_2 \setminus \{L_2\},\ M,\ \mathit{Path} \cup \{L_1\} \qquad C,\ M,\ \mathit{Path}}{C \cup \{L_1\},\ M,\ \mathit{Path}}\quad \mathit{Extension}$$

Example lean connection proof

Clauses:

$c_1 : P(x)$

$c_2 : R(x, y) \vee \neg P(x) \vee Q(y)$

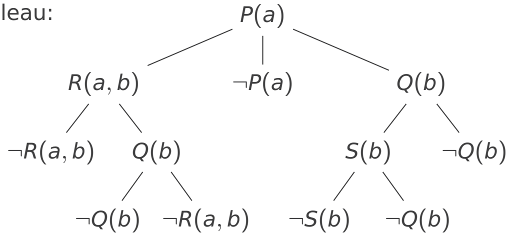
$c_3 : S(x) \vee \neg Q(b)$

$c_4 : \neg S(x) \vee \neg Q(x)$

$c_5 : \neg Q(x) \vee \neg R(a, x)$

$c_6 : \neg R(a, x) \vee Q(x)$

Tableau:



leanCoP proofs usually presented in DNF rather than CNF

leanCoP Example

[*Otten'15*]

- Formula to prove:

$$(\,((\exists x\, Q(x) \vee \neg Q(c)) \Rightarrow P) \;\wedge\; (P \Rightarrow (\exists y\, Q(y) \wedge R))\,) \;\Rightarrow\; (P \wedge R)$$

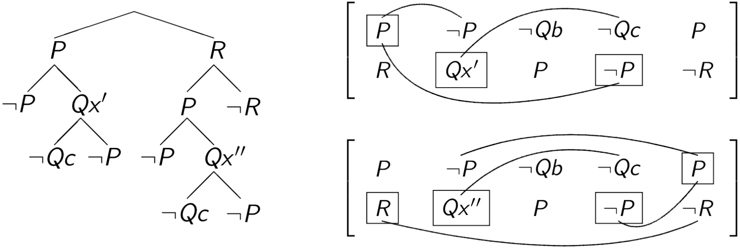
- DNF:

$$(P \wedge R) \vee (\neg P \wedge Qx) \vee (\neg Qb \wedge P) \vee (\neg Qc \wedge \neg P) \vee (P \wedge \neg R)$$

- Matrix:

$$\left[\; \begin{bmatrix} P \\ R \end{bmatrix} \begin{bmatrix} \neg P \\ Qx \end{bmatrix} \begin{bmatrix} \neg Qb \\ P \end{bmatrix} \begin{bmatrix} \neg Qc \\ \neg P \end{bmatrix} \begin{bmatrix} P \\ \neg R \end{bmatrix} \;\right]$$

- Tableaux:



LeanCoP implementation

Not only is the calculus very small, but it is possible to implement the prover, including its optimized version in less than 20 lines of code.

That is already working on a prepared normal form, but still it is impressive

leanCoP: Basic Code

```
prove([Lit|Cla],Path,PathLim,Lem,Set) :-
  (-NegLit=Lit;-Lit=NegLit) ->
    (
        member(NegL,Path),unify_with_occurs_check(NegL,NegLit)
    ;
        lit(NegLit,NegL,Cla1,Grnd1),
        unify_with_occurs_check(NegL,NegLit),
        prove(Cla1,[Lit|Path],PathLim,Lem,Set)
    ),
    prove(Cla,Path,PathLim,Lem,Set).
prove([],_,_,_,_).
```

leanCoP: Actual Code (Optimizations, No history)

```
prove([Lit|Cla],Path,PathLim,Lem,Set) :-
  \+ (member(LitC,[Lit|Cla]), member(LitP,Path),LitC==LitP),
  (-NegLit=Lit;-Lit=NegLit) ->
    (
      member(LitL,Lem), Lit==LitL
    ;
      member(NegL,Path),unify_with_occurs_check(NegL,NegLit)
    ;
      lit(NegLit,NegL,Cla1,Grnd1),
      unify_with_occurs_check(NegL,NegLit),
        ( Grnd1=g -> true ;
          length(Path,K), K<PathLim -> true ;
          \+ pathlim -> assert(pathlim), fail ),
      prove(Cla1,[Lit|Path],PathLim,Lem,Set)
    ),
    ( member(cut,Set) -> ! ; true ),
    prove(Cla,Path,PathLim,[Lit|Lem],Set).
prove([],_,_,_,_).
```

First ML experiment: MaLeCoP in Prolog

[Tableaux 2011]

Use ML to select next extension step

- Using external advice

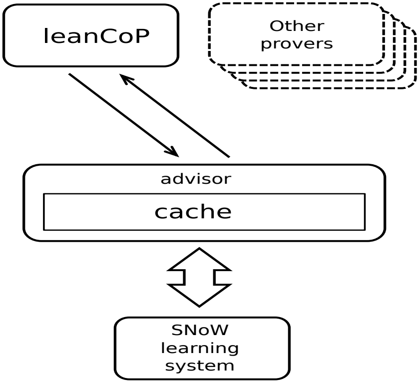
Slow implementation

- 1000 less inferences per second than the prolog version

Can avoid 90% inferences!

Important for this achievements:

- Caching of decisions
- Special strategies, such as only do learning in the first few steps



What about efficiency: FEMaLeCoP

[LPAR 2015]

Advise the:

- selection of clause for every tableau extension step

Proof state: weighted vector of symbols (or terms)

- extracted from all the literals on the active path
- Frequency-based weighting (IDF)
- Simple decay factor (using maximum)

Consistent clausification

- formula `?[X]: p(X)` becomes `p('skolem(?[A]:p(A),1)')`

Predictor: Custom sparse naive Bayes

- association of the features of the proof states
- with contrapositives used for the successful extension steps

FEMaLeCoP: Data Collection and Indexing

Extension of the saved proofs

- Training Data: pairs (path, used extension step)

External Data Indexing (incremental)

- `te_num`: number of training examples
- `pf_no`: map from features to number of occurrences $\in \mathbb{Q}$
- `cn_no`: map from contrapositives to numbers of occurrences
- `cn_pf_no`: map of maps of cn/pf co-occurrences

Problem Specific Data

- Upon start FEMaLeCoP reads
  - **only current-problem relevant** parts of the training data
- cn_no and cn_pf_no filtered by contrapositives in the problem
- pf_no and cn_pf_no filtered by possible features in the problem

Efficient Relevance (1/2)

Very similar to Naive Bayes for Premise selection

Estimate the relevance of each contrapositive φ by

$$\text{P}(\varphi \text{ is used in a proof in state } \psi \mid \psi \text{ has features F}(\gamma))$$

where $F(\gamma)$ are the features of the current path.

Efficient Relevance (1/2)

Very similar to Naive Bayes for Premise selection

Estimate the relevance of each contrapositive φ by

$$\mathrm{P}(\varphi \text{ is used in a proof in state } \psi \mid \psi \text{ has features } \mathrm{F}(\gamma))$$

where $F(\gamma)$ are the features of the current path.

Assuming the features are independent, this is:

$$\begin{aligned} &P(\varphi \text{ is used in } \psi\text{'s proof}) \\ &\quad \cdot \prod_{f \in F(\gamma) \cap F(\varphi)} P\big(\psi \text{ has feature } f \mid \varphi \text{ is used in } \psi\text{'s proof}\big) \\ &\quad \cdot \prod_{f \in F(\gamma) - F(\varphi)} P\big(\psi \text{ has feature } f \mid \varphi \text{ is not used in } \psi\text{'s proof}\big) \\ &\quad \cdot \prod_{f \in F(\varphi) - F(\gamma)} P\big(\psi \text{ does not have } f \mid \varphi \text{ is used in } \psi\text{'s proof}\big) \end{aligned}$$

Efficient Relevance (2/2)

All these probabilities can be estimated (using training examples):

$$\sigma_1 \ln t + \sum_{f \in (\bar{f} \cap \bar{s})} i(f) \ln \frac{\sigma_2 s(f)}{t} + \sigma_3 \sum_{f \in (\bar{f} - \bar{s})} i(f) + \sigma_4 \sum_{f \in (\bar{s} - \bar{f})} i(f) \ln(1 - \frac{s(f)}{t})$$

where

- $\bar{f}$ are the features of the path
- $\bar{s}$ are the features that co-occurred with φ
- $t = cn\_no(\varphi)$
- $s = cn\_fp\_no(\varphi)$
- i is the IDF
- σ_* are experimentally chosen parameters

Inference speed ... drops to about 40%

Which is not too bad. And the slower learning version can prove more problems in the same time:

Prover	Proved (%)
OCaml-leanCoP	574 (27.6%)
FEMaLeCoP	635 (30.6%)
together	664 (32.0%)

(evaluation on MPTP bushy problems, 60 s)

On various datasets, 3–15% problems more solved than leanCoP

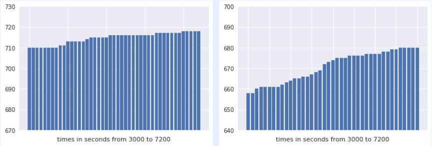
Note that the evaluation requires training data. So learned version together with data collection is compared to a non-learning version having more time.

What about stronger learning?

We have tried much stronger learning in the same setup:

XGboost helps minimally and makes it too slow

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



NBayes vs XGBoost on 2h timeout

Preliminary experiments with deep learning *[Olšak 2017]*

- Too slow to compare in meaningful scenarios.

So how we can use learning differently?

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.

Can we provide a tree search like interface?

- Two functions suffice

$$\text{start} : \text{problem} \rightarrow \text{state}$$

$$\text{action} : \text{action} \rightarrow \text{state}$$

- where

$$\text{state} = \langle \text{action list} \times \text{remaining goal-paths} \rangle$$

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!)

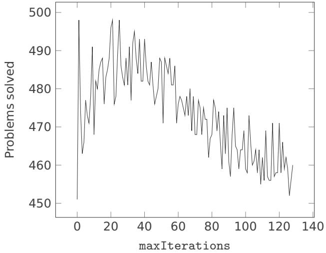
But with random playouts: depth hundreds of thousands!

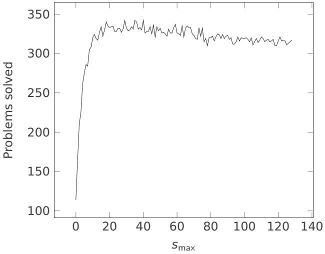
- Just unlikely to find a proof $\rightarrow$ learning

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 of the actual proof search





Sometimes the playouts randomly find proofs, but that’s not significant enough to be of major benefit. We need the actual playouts to be guided!

That’s what happens in modern AI for games. Can we take inspiration from these?

AlphaZero in a nutshell

Use two neural networks: One for the selection of moves (policy), second one for the evaluation of positions (value)

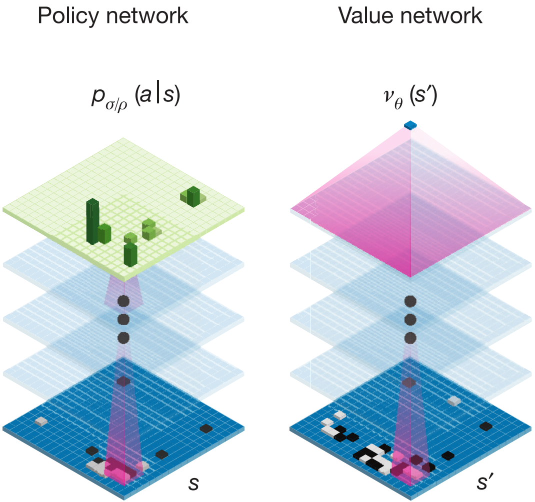
At any position perform playouts guided by the policy network that is slightly adjusted by the values in the explored subtree

Use these adjustments to train (improve) the policy network and use final game scores to train the value network

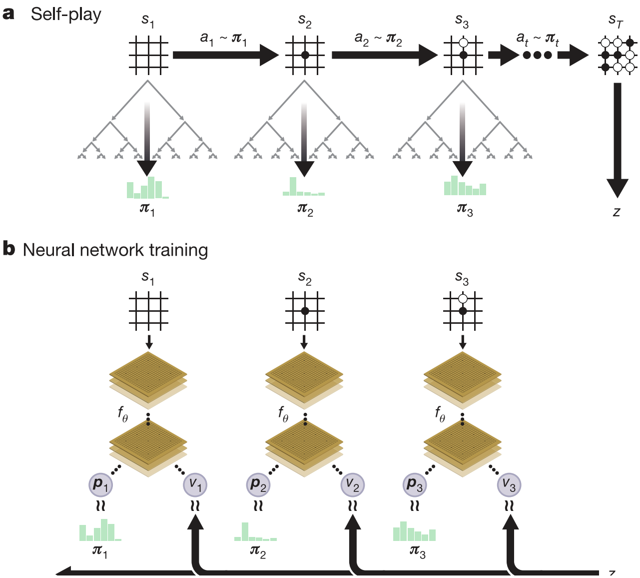
These intuitions on images based on the [Silver et al] paper on next three slides

AlphaZero (1/3)

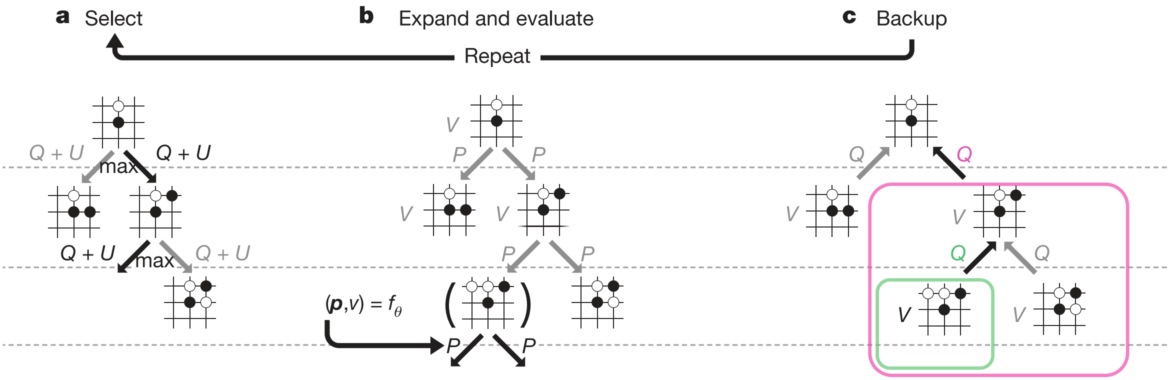
[*Silver et al.*]



AlphaZero (2/3)



AlphaZero (3/3)



How to select the best action?

[*Szepesvari 2006*]

Intuition

- Given some prior probabilities
- And having done some experiments
- Which action to take?
- (later extended to sequences of actions in a tree)

How to select the best action?

[*Szepesvari 2006*]

Intuition

- Given some prior probabilities
- And having done some experiments
- Which action to take?
- (later extended to sequences of actions in a tree)

Monte Carlo Tree Search with Upper Confidence Bounds for Trees

- Select node *n* maximizing

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

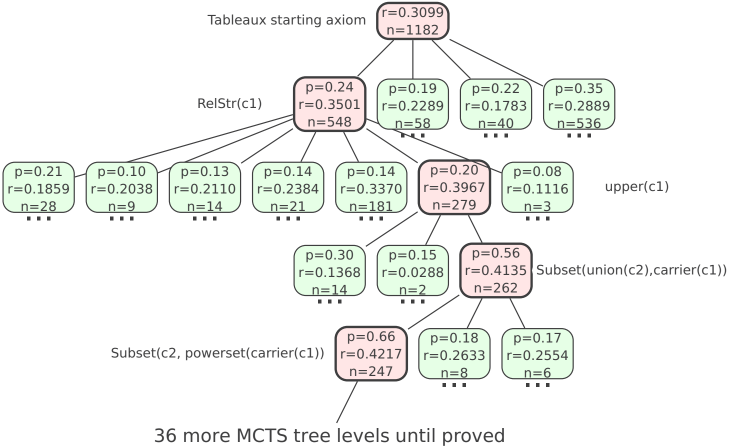
- where

$\frac{w_i}{n_i}$ average reward p_i action i prior

N number of experiments n_i action i experiments

MCTS tree for `WAYBEL_0:28`

How does this work for theorem proving? An example tree with priors, average rewards and visit counts for an ATP problem:



Learn Policy and Value in theorem proving

Policy: Which actions to take?

- Proportions predicted based on proportions in similar states
- Explore less the actions that were “bad” in the past
- Explore more and earlier the actions that were “good”

Value: How good (close to a proof) is a state?

- Intuitively reward states that have few goals or easy goals

Where to get training data?

- Explore 1000 nodes using UCT
- Select the most visited action and focus on it for this proof
- A sequence of selected actions can train both policy and value

Mizar TPTP problems

Split problems in training (29272) and test (3252) sets

Baseline: Non-learned UCT (uniform policy and value)

System	leanCoP	playouts	UCT
Test	**1143**	431	804

Mizar TPTP problems

Split problems in training (29272) and test (3252) sets

Baseline: Non-learned UCT (uniform policy and value)

System	leanCoP	playouts	UCT
Test	**1143**	431	804

10 training iterations

Iteration	1	2	3	4	5
Test	1354	1519	1566	1595	**1624**

Mizar TPTP problems

Split problems in training (29272) and test (3252) sets

Baseline: Non-learned UCT (uniform policy and value)

System	leanCoP	playouts	UCT
Train	10438	4184	7348
Test	**1143**	431	804

10 training iterations

Iteration	1	2	3	4	5	6	7	8	9	10
Train	12325	13749	14155	14363	14403	14431	14342	**14498**	14481	14487
Test	1354	1519	1566	1595	**1624**	1586	1582	1591	1577	1621

Mizar TPTP problems

Split problems in training (29272) and test (3252) sets

Baseline: Non-learned UCT (uniform policy and value)

System	leanCoP	playouts	UCT
Train	10438	4184	7348
Test	**1143**	431	804

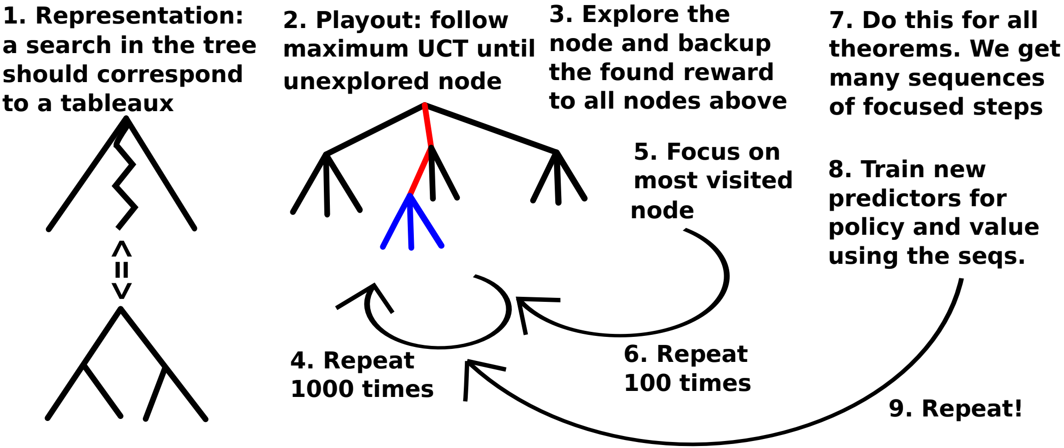
10 training iterations

Iteration	1	2	3	4	5	6	7	8	9	10
Train	12325	13749	14155	14363	14403	14431	14342	**14498**	14481	14487
Test	1354	1519	1566	1595	**1624**	1586	1582	1591	1577	1621

Original LeanCoP with same time as all iterations

leanCoP, 20 times more inferences, strategies	1396
rlCoP union	1839

RL-CoP setup summary



Nice, but theorem proving requiring significant hardware!

Additional Literature (not required)

Details on most important optimizations in leanCoP

Jens Otten.
Restricting backtracking in connection calculi.
AI Commun., 23(2-3):159–182, 2010.

Main paper describing AlphaGo

David Silver et al.
Mastering the game of Go with deep neural networks and tree search.
Nature, 529(7587):484–489, 2016.

Graph neural networks used to guide leanCoP

Miroslav Olsák, Cezary Kaliszyk, and Josef Urban.
Property invariant embedding for automated reasoning.
ECAI, 2020.

Summary

This Lecture

- lean connection calculus
- reinforcement learning for leanCoP

Next

- Proof Library Alignment
- Auto-formalization
- Final presentations/test!