Efficient Neural Clause-Selection Reinforcement

Martin Suda*

Czech Technical University in Prague, Czech Republic

NaFoMa 2025, Bonn, June 2025

*Supported by the Czech Science Foundation standard project 24-12759S and the Cost action CA20111 EuroProofNet.

Machine-Learning-Boosted Automated Theorem Proving

ATP technology:

ATP technology: saturation-based

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, . . .

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, . . .

Heuristic to boost:

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, ...

Heuristic to boost: clause selection

Machine-Learning-Boosted Automated Theorem Proving

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, . . .

Heuristic to boost: clause selection

- arguably the most important choice point

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, . . .

Heuristic to boost: clause selection

- arguably the most important choice point
- "selecting just the proof clauses" intuition

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, . . .

Heuristic to boost: clause selection

- arguably the most important choice point
- "selecting just the proof clauses" intuition

Three main topics for today:

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, . . .

Heuristic to boost: clause selection

- arguably the most important choice point
- "selecting just the proof clauses" intuition

Three main topics for today:

- a RL-inspired learning operator

Machine-Learning-Boosted Automated Theorem Proving

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, ...

Heuristic to boost: clause selection

- arguably the most important choice point
- "selecting just the proof clauses" intuition

Three main topics for today:

- a RL-inspired learning operator
- a new neural architecture (GNN + RvNNs + MLP)

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, . . .

Heuristic to boost: clause selection

- arguably the most important choice point
- "selecting just the proof clauses" intuition

Three main topics for today:

- a RL-inspired learning operator
- a new neural architecture (GNN + RvNNs + MLP)
- put into Vampire

Machine-Learning-Boosted Automated Theorem Proving

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, ...

Heuristic to boost: clause selection

- arguably the most important choice point
- "selecting just the proof clauses" intuition

Three main topics for today:

- a RL-inspired learning operator
- a new neural architecture (GNN + RvNNs + MLP)
- put into Vampire $\Rightarrow$ Deepire 2.0

ATP technology: saturation-based

- state of the art (cf. CASC)
- E, iProver, Vampire, ...

Heuristic to boost: clause selection

- arguably the most important choice point
- "selecting just the proof clauses" intuition

Three main topics for today:

- a RL-inspired learning operator
- a new neural architecture (GNN + RvNNs + MLP)
- put into Vampire $\Rightarrow$ Deepire 2.0 $\Rightarrow$ 20 % performance boost

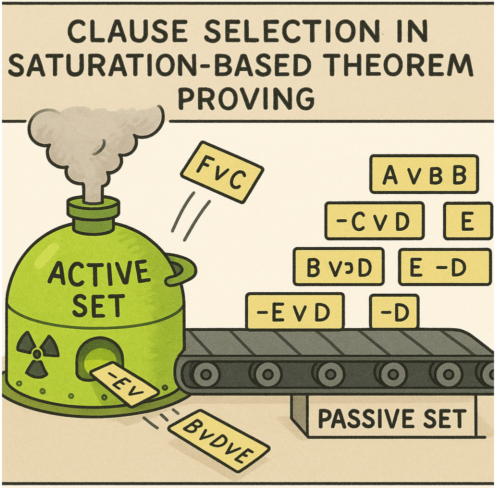
Outline

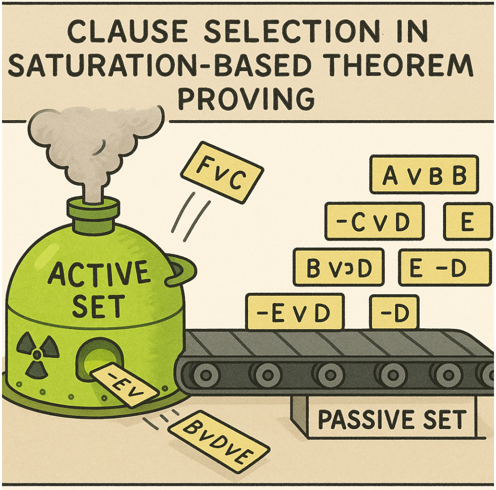
1. Saturation and Clause Selection
2. RL-Inspired Guidance
3. Neural Clause Evaluation
4. Deepire 2.0

Outline

1. Saturation and Clause Selection
2. RL-Inspired Guidance
3. Neural Clause Evaluation
4. Deepire 2.0

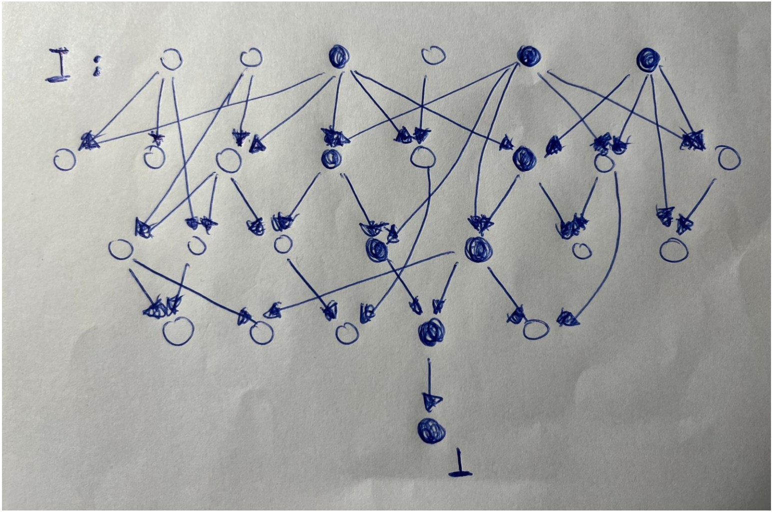
Saturation-based Theorem Proving

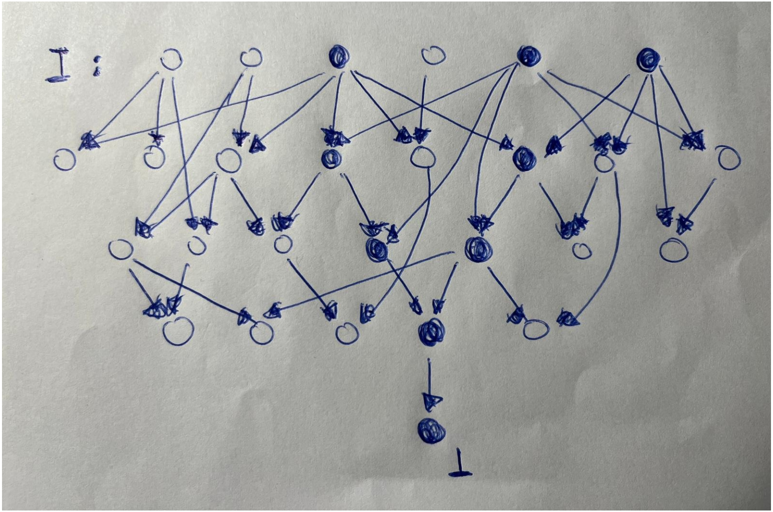




At a typical successful end: $|\mathit{Passive}| \gg |\mathit{Active}| \gg |\mathit{Proof}|$

The Proof Is Often Just A Tiny Part





How close can we actually hope get to the perfect clause selection?

How is clause selection traditionally done?

Take simple clause evaluation criteria:

- age: prefer clauses that were generated long time ago
- weight: prefer clauses with fewer symbols

How is clause selection traditionally done?

Take simple clause evaluation criteria:

- age: prefer clauses that were generated long time ago
- weight: prefer clauses with fewer symbols

Combine them into a single scheme:

- have a priority queue ordering *Passive* for each criterion
- alternate between selecting from the queues using a fixed ratio

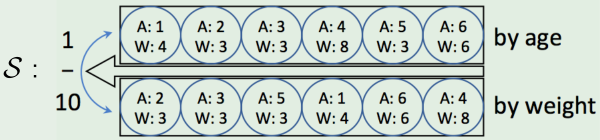
Take simple clause evaluation criteria:

- age: prefer clauses that were generated long time ago
- weight: prefer clauses with fewer symbols

Combine them into a single scheme:

- have a priority queue ordering *Passive* for each criterion
- alternate between selecting from the queues using a fixed ratio

Example (Organizing *Passive* via two priority queues)



Outline

1. Saturation and Clause Selection
2. RL-Inspired Guidance
3. Neural Clause Evaluation
4. Deepire 2.0

Why Reinforcement Learning?

Why Reinforcement Learning?

Inspired by the great successes:

- ATARI games (DQN)
  V. Mnih et al. Playing ATARI with deep reinforcement learning. CoRR, 2013.
- Board games (AlphaZero)
  D. Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 2018.
- ...

Why Reinforcement Learning?

Inspired by the great successes:

- ATARI games (DQN)
  V. Mnih et al. Playing ATARI with deep reinforcement learning. CoRR, 2013.
- Board games (AlphaZero)
  D. Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 2018.
- ...
- "I wan't to try it on my pet problem too!"

Why Reinforcement Learning?

Inspired by the great successes:

- ATARI games (DQN)
  V. Mnih et al. Playing ATARI with deep reinforcement learning. CoRR, 2013.
- Board games (AlphaZero)
  D. Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 2018.
- ...
- "I wan't to try it on my pet problem too!"

What's really unique about RL?

- It programs itself

Why Reinforcement Learning?

Inspired by the great successes:

- ATARI games (DQN)
  V. Mnih et al. Playing ATARI with deep reinforcement learning. CoRR, 2013.
- Board games (AlphaZero)
  D. Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 2018.
- ...
- "I wan't to try it on my pet problem too!"

What's really unique about RL?

- It programs itself (sometimes even optimally, in the limit)

Inspired by the great successes:

- ATARI games (DQN)
  V. Mnih et al. Playing ATARI with deep reinforcement learning. CoRR, 2013.
- Board games (AlphaZero)
  D. Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 2018.
- ...
- "I wan't to try it on my pet problem too!"

What's really unique about RL?

- It programs itself (sometimes even optimally, in the limit)
- It could discover fundamentally novel tricks and hacks!

Key Reinforcement Learning Concepts

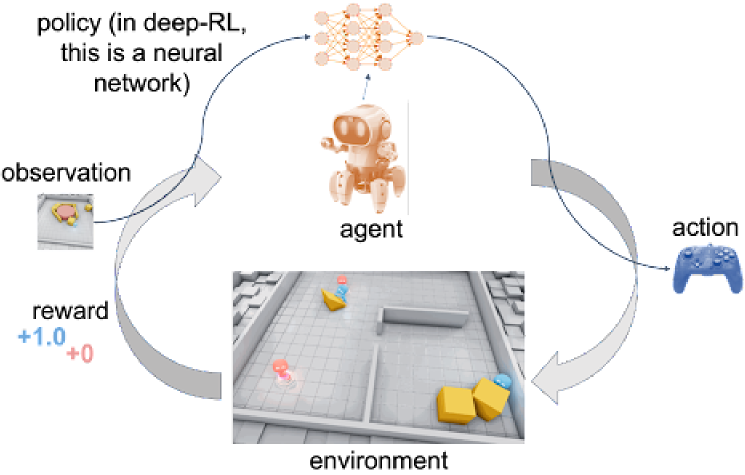


*Illustration from anyscale.com.

Saturation as an Reinforcement-Learning Environment

Agent

- the clause selection heuristic

Action

- the next clause to select <u>from the current passive set</u>

Saturation as an Reinforcement-Learning Environment

Agent

- the clause selection heuristic

Action

- the next clause to select <u>from the current passive set</u>

State / Observation

- <u>static</u> - the conjecture we are trying to prove
- <u>evolving</u> - the internal state of the prover at particular moment

Saturation as an Reinforcement-Learning Environment

Agent

- the clause selection heuristic

Action

- the next clause to select <u>from the current passive set</u>

State / Observation

- <u>static</u> - the conjecture we are trying to prove
- <u>evolving</u> - the internal state of the prover at particular moment

Reward

- Score 1 point for solving a problem (within the time limit)

Saturation as an Reinforcement-Learning Environment

Agent

- the clause selection heuristic

Action

- the next clause to select from the current passive set

State / Observation

- static - the conjecture we are trying to prove
- evolving - the internal state of the prover at particular moment

Reward

- Score 1 point for solving a problem (within the time limit) ???

Saturation as an Reinforcement-Learning Environment

Agent

- the clause selection heuristic

Action

- the next clause to select <u>from the current passive set</u>

State / Observation

- <u>static</u> - the conjecture we are trying to prove
- <u>evolving</u> - the internal state of the prover at particular moment

Reward

- Score 1 point for solving a problem (within the time limit) ???

➥ TRAIL [Crouse et al.'21], [McKeown'23], [Shminke'23], ...

Design Decisions

Guiding Principle

The new design accommodates the old heuristic as an attainable point in the space of possible solutions.

Guiding Principle

The new design accommodates the old heuristic as an attainable point in the space of possible solutions.

State / Observation

- the evolving state of an ATP is a large amorphous blob

Design Decisions

Guiding Principle

The new design accommodates the old heuristic as an attainable point in the space of possible solutions.

State / Observation

- the evolving state of an ATP is a large amorphous blob
- there is no state in the SoTA clause-selection heuristics

Design Decisions

Guiding Principle

The new design accommodates the old heuristic as an attainable point in the space of possible solutions.

State / Observation

- the evolving state of an ATP is a large amorphous blob
- there is no state in the SoTA clause-selection heuristics
- let's ditch state too

Design Decisions

Guiding Principle

The new design accommodates the old heuristic as an attainable point in the space of possible solutions.

State / Observation

- the evolving state of an ATP is a large amorphous blob
- there is no state in the SoTA clause-selection heuristics
- let's ditch state too $\Rightarrow$ assumption of <u>state-less</u> environment

Design Decisions

Guiding Principle

The new design accommodates the old heuristic as an attainable point in the space of possible solutions.

State / Observation

- the evolving state of an ATP is a large amorphous blob
- there is no state in the SoTA clause-selection heuristics
- let's ditch state too $\Rightarrow$ assumption of <u>state-less</u> environment

Reward

- refusing the play the honest, super-sparse reward game

Design Decisions

Guiding Principle

The new design accommodates the old heuristic as an attainable point in the space of possible solutions.

State / Observation

- the evolving state of an ATP is a large amorphous blob
- there is no state in the SoTA clause-selection heuristics
- let's ditch state too $\Rightarrow$ assumption of <u>state-less</u> environment

Reward

- refusing the play the honest, super-sparse reward game
- like in ENIGMA: a proof clause is a good clauses

Towards the RL-Inspired Learning Operator

A trace of a successful proof attempt on problem P is a tuple

$$T = (P, \mathcal{C}, \mathcal{C}^+, \{\mathcal{P}_i\}_{i \in I_T}).$$

Towards the RL-Inspired Learning Operator

A trace of a successful proof attempt on problem P is a tuple

$$T = (P, \mathcal{C}, \mathcal{C}^+, \{\mathcal{P}_i\}_{i \in I_T}).$$

Learning operator (for clause selection)

- input: neural network $N_{\boldsymbol{\theta}}$ (learnable params $\boldsymbol{\theta}$), set of traces $\mathcal{T}$
- output: updated parameters $\boldsymbol{\theta}'$,
  such that $N_{\boldsymbol{\theta}'}$ is better at solving problems like those from $\mathcal{T}$

Towards the RL-Inspired Learning Operator

A trace of a successful proof attempt on problem P is a tuple

$$T = (P, \mathcal{C}, \mathcal{C}^+, \{\mathcal{P}_i\}_{i \in I_T}).$$

Learning operator (for clause selection)

- input: neural network $N_{\boldsymbol{\theta}}$ (learnable params $\boldsymbol{\theta}$), set of traces $\mathcal{T}$
- output: updated parameters $\boldsymbol{\theta}'$,
  such that $N_{\boldsymbol{\theta}'}$ is better at solving problems like those from $\mathcal{T}$

Logits and Policy

Assuming $N_{\boldsymbol{\theta}}$ produces a score $N_{\boldsymbol{\theta}}(C) = l_C$ for each clause C, then

A trace of a successful proof attempt on problem P is a tuple

$$T = (P, \mathcal{C}, \mathcal{C}^+, \{\mathcal{P}_i\}_{i \in I_T}).$$

Learning operator (for clause selection)

- input: neural network $N_{\boldsymbol{\theta}}$ (learnable params $\boldsymbol{\theta}$), set of traces $\mathcal{T}$
- output: updated parameters $\boldsymbol{\theta}'$,
  such that $N_{\boldsymbol{\theta}'}$ is better at solving problems like those from $\mathcal{T}$

Logits and Policy
Assuming $N_{\boldsymbol{\theta}}$ produces a score $N_{\boldsymbol{\theta}}(C) = l_C$ for each clause C, then

$$\pi_{C,\boldsymbol{\theta}} = \mathrm{softmax}_C\left(\{l_D\}_{D \in \mathcal{P}}\right) = \frac{e^{l_C}}{\sum_{D \in \mathcal{P}} e^{l_D}}$$

is the (stochastic) clause selection policy defined by $N_{\boldsymbol{\theta}}$

The RL-Inspired Operator

Policy Gradient Theorem [Williams'92]

To improve a policy in terms of the expected return we update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha r_C \nabla_{\boldsymbol{\theta}} \log \pi_{C,\boldsymbol{\theta}},$$

where r_C is the return / reward at the corresponding step.

The RL-Inspired Operator

Policy Gradient Theorem [Williams'92]

To improve a policy in terms of the expected return we update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha r_C \nabla_{\boldsymbol{\theta}} \log \pi_{C,\boldsymbol{\theta}},$$

where r_C is the return / reward at the corresponding step.

Our Operator:

Each moment in time i is an independent opportunity to improve, with

$$\delta_i^T = \mathrm{mean}_{C \in \mathcal{P}_i^+} \nabla_{\boldsymbol{\theta}} \log \pi_{C,\boldsymbol{\theta}},$$

for a trace $T = (P, \mathcal{C}, \mathcal{C}^+, \{\mathcal{P}_i\}_{i \in I_T})$ and $\mathcal{P}_i^+ = \mathcal{P}_i \cap \mathcal{C}^+$.

The RL-Inspired Operator

Policy Gradient Theorem [Williams'92]

To improve a policy in terms of the expected return we update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha r_C \nabla_{\boldsymbol{\theta}} \log \pi_{C,\boldsymbol{\theta}},$$

where r_C is the return / reward at the corresponding step.

Our Operator:

Each moment in time i is an independent opportunity to improve, with

$$\delta_i^T = \mathrm{mean}_{C \in \mathcal{P}_i^+} \nabla_{\boldsymbol{\theta}} \log \pi_{C,\boldsymbol{\theta}},$$

for a trace $T = (P, \mathcal{C}, \mathcal{C}^+, \{\mathcal{P}_i\}_{i \in I_T})$ and $\mathcal{P}_i^+ = \mathcal{P}_i \cap \mathcal{C}^+$. Then

$$\delta^T = \mathrm{mean}_{i \in I_T} \delta_i^T \text{ and } \delta = \mathrm{mean}_{T \in \mathcal{T}} \delta^T.$$

Outline

Neural Clause Evaluation

Aim for a balance between expressivity and speed of inference!

Neural Clause Evaluation

Aim for a balance between expressivity and speed of inference!

One-off GNN Invocation:

- Graph Neural Networks

Neural Clause Evaluation

Aim for a balance between expressivity and speed of inference!

One-off GNN Invocation:

- Graph Neural Networks
- name-invariant formula representations

Neural Clause Evaluation

Aim for a balance between expressivity and speed of inference!

One-off GNN Invocation:

- Graph Neural Networks
- name-invariant formula representations
- relatively expensive; the more context the better

Neural Clause Evaluation

Aim for a balance between expressivity and speed of inference!

One-off GNN Invocation:

- Graph Neural Networks
- name-invariant formula representations
- relatively expensive; the more context the better
- here: only apply to the input CNF

Neural Clause Evaluation

Aim for a balance between expressivity and speed of inference!

One-off GNN Invocation:

- Graph Neural Networks
- name-invariant formula representations
- relatively expensive; the more context the better
- here: only apply to the input CNF

Generalizing Age and Weight with RvNNs:

- Recursive Neural Networks

Neural Clause Evaluation

Aim for a balance between expressivity and speed of inference!

One-off GNN Invocation:

- Graph Neural Networks
- name-invariant formula representations
- relatively expensive; the more context the better
- here: only apply to the input CNF

Generalizing Age and Weight with RvNNs:

- Recursive Neural Networks
- g-age: grow along the clause derivation tree

Neural Clause Evaluation

Aim for a balance between expressivity and speed of inference!

One-off GNN Invocation:

- Graph Neural Networks
- name-invariant formula representations
- relatively expensive; the more context the better
- here: only apply to the input CNF

Generalizing Age and Weight with RvNNs:

- Recursive Neural Networks
- g-age: grow along the clause derivation tree
- g-weight: grow along the clause syntax tree

Neural Clause Evaluation

Aim for a balance between expressivity and speed of inference!

One-off GNN Invocation:

- Graph Neural Networks
- name-invariant formula representations
- relatively expensive; the more context the better
- here: only apply to the input CNF

Generalizing Age and Weight with RvNNs:

- Recursive Neural Networks
- g-age: grow along the clause derivation tree
- g-weight: grow along the clause syntax tree
- share substructures (dag) and cache results

Aim for a balance between expressivity and speed of inference!

One-off GNN Invocation:

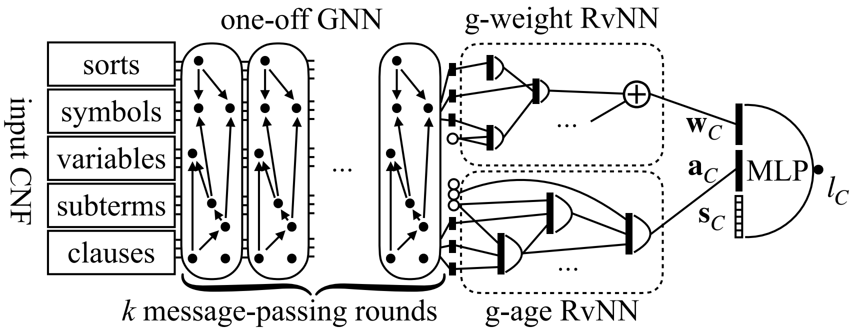
- Graph Neural Networks
- name-invariant formula representations
- relatively expensive; the more context the better
- here: only apply to the input CNF

Generalizing Age and Weight with RvNNs:

- Recursive Neural Networks
- g-age: grow along the clause derivation tree
- g-weight: grow along the clause syntax tree
- share substructures (dag) and cache results

Simple Hand-Crafted Features on Top!

Architecture Diagram



Note on Efficient RvNNs

Strive to compute as much as possible in one bulk!

Note on Efficient RvNNs

Strive to compute as much as possible in one bulk!

```
def gage_insert(cl_num:int,
                inf_rule:int,
                parents:list[int]):
  level = max(base_level,
                1 + max(height[p] for p in parents))
  height[cl_num] = level
  index = level - base_level
  if len(todo_layers) == index:
    todo_layers.append([])
  todo_layers[index].append((cl_num, inf_rule, parents))
```

Note on Efficient RvNNs

Strive to compute as much as possible in one bulk!

```
def gage_insert(cl_num:int,
                inf_rule:int,
                parents:list[int]):
  level = max(base_level,
              1 + max(height[p] for p in parents))
  height[cl_num] = level
  index = level - base_level
  if len(todo_layers) == index:
    todo_layers.append([])
  todo_layers[index].append((cl_num, inf_rule, parents))
```

I still need to try out how much GPUs could help here . . .

Outline

1. Saturation and Clause Selection
2. RL-Inspired Guidance
3. Neural Clause Evaluation
4. Deepire 2.0

Implementation

Single Clause Queue:

- ordered by the computed logits $N_\theta(C) = l_C$

Implementation

Single Clause Queue:

- ordered by the computed logits $N_\theta(C) = l_C$
- Could we also sample?

Implementation

Single Clause Queue:

- ordered by the computed logits $N_\theta(C) = l_C$
- Could we also sample?
- Gumbel-max trick:

Implementation

Single Clause Queue:

- ordered by the computed logits $N_\theta(C) = l_C$
- Could we also sample?
- Gumbel-max trick: add some Gumbel noise

$$g = -\log(-\log(u)) \text{ for } u \sim \mathrm{Uniform}(0,1)$$

Implementation

Single Clause Queue:

- ordered by the computed logits $N_\theta(C) = l_C$
- Could we also sample?
- Gumbel-max trick: add some Gumbel noise

$$g = -\log(-\log(u)) \text{ for } u \sim \mathrm{Uniform}(0, 1)$$

Delayed Insertion Buffer:

Implementation

Single Clause Queue:

- ordered by the computed logits $N_\theta(C) = l_C$
- Could we also sample?
- Gumbel-max trick: add some Gumbel noise

$$g = -\log(-\log(u)) \text{ for } u \sim \mathrm{Uniform}(0,1)$$

Delayed Insertion Buffer:

- insertions into passive are lazy
- only evaluate things in buffer when selection is called

Implementation

Single Clause Queue:

- ordered by the computed logits $N_\theta(C) = l_C$
- Could we also sample?
- Gumbel-max trick: add some Gumbel noise

$$g = -\log(-\log(u)) \text{ for } u \sim \mathrm{Uniform}(0, 1)$$

Delayed Insertion Buffer:

- insertions into passive are lazy
- only evaluate things in buffer when selection is called

Iterative Improvement Loop:

- run prover, collect traces, train from traces, run improved prover, repeat

Implementation

Single Clause Queue:

- ordered by the computed logits $N_\theta(C) = l_C$
- Could we also sample?
- Gumbel-max trick: add some Gumbel noise

$$g = -\log(-\log(u)) \text{ for } u \sim \mathrm{Uniform}(0, 1)$$

Delayed Insertion Buffer:

- insertions into passive are lazy
- only evaluate things in buffer when selection is called

Iterative Improvement Loop:

- run prover, collect traces, train from traces, run improved prover, repeat
- little trick; despite the RL heritage:
  inner loop trains until validation loss does not improve

Experiments

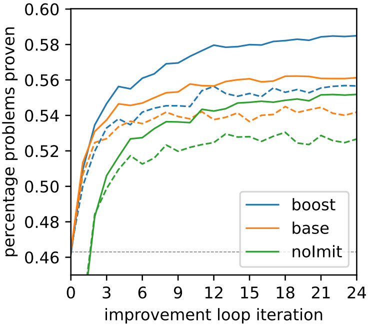
Setup:

- TPTP v9 CNF+FOF, 19 477 problems (train/test split)
- Vampire's default strategy (1:1 age-weight alternation)
- limit of 30 000 Mi (∼10 s) per proof attempt

Experiments

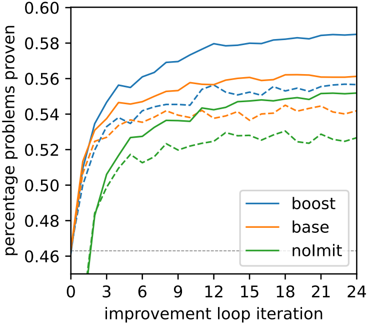
Setup:

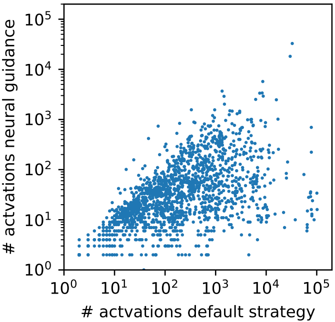
- TPTP v9 CNF+FOF, 19 477 problems (train/test split)
- Vampire's default strategy (1:1 age-weight alternation)
- limit of 30 000 Mi (∼10 s) per proof attempt



Setup:

- TPTP v9 CNF+FOF, 19 477 problems (train/test split)
- Vampire's default strategy (1:1 age-weight alternation)
- limit of 30 000 Mi (∼10 s) per proof attempt





Experiments II

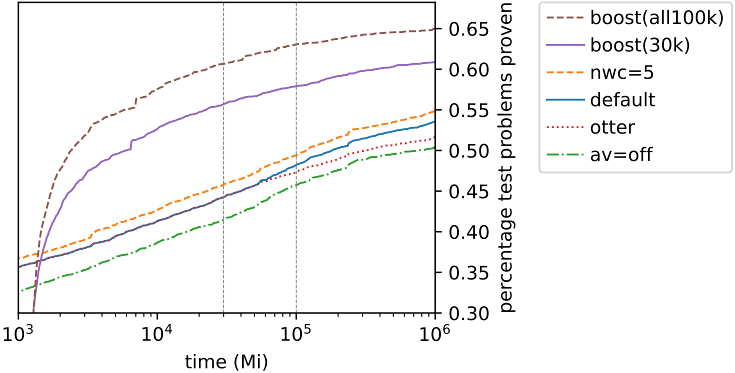
Solving Hard Problems:

- overfit to TPTP with 100 000 Mi-limited runs
- ran for 12.4 days
- solved 130 rating 1.0 (49 never solved, 8 status UNK)

Solving Hard Problems:

- overfit to TPTP with 100 000 Mi-limited runs
- ran for 12.4 days
- solved 130 rating 1.0 (49 never solved, 8 status UNK)

Put Into Perspective:



Conclusion

Summary:

- new efficient name-invariant neural architecture
- new learning operator inspired by reinforcement learning
- implementation in Vampire
  - 20 % performance boost of the default strategy
  - trained model can solve many very hard (previously unsolved) TPTP problems

Conclusion

Summary:

- new efficient name-invariant neural architecture
- new learning operator inspired by reinforcement learning
- implementation in Vampire
  - 20 % performance boost of the default strategy
  - trained model can solve many very hard (previously unsolved) TPTP problems

Outlook:

- ENIGMA-style vs RL-inspired learning
- other benchmarks than TPTP; e.g. Mizar40; transfer learning
- neural guidance and theorem proving strategies

Conclusion

Summary:

- new efficient name-invariant neural architecture
- new learning operator inspired by reinforcement learning
- implementation in Vampire
  - 20 % performance boost of the default strategy
  - trained model can solve many very hard (previously unsolved) TPTP problems

Outlook:

- ENIGMA-style vs RL-inspired learning
- other benchmarks than TPTP; e.g. Mizar40; transfer learning
- neural guidance and theorem proving strategies

Thank you!

ENIGMA-style

The core idea

Learn to recognize and prefer for selection clauses that look like those that contributed to a proof in past successful runs.

➥ [Schulz00], ENIGMA [Jakubův&Urban17], [Loos et al.'17], ...

The core idea

Learn to recognize and prefer for selection clauses that look like those that contributed to a proof in past successful runs.

➥ [Schulz00], ENIGMA [Jakubův&Urban17], [Loos et al.'17], ...

The "pos/neg"'s of E:

E prover can be asked to output, for every clause selected in a run, whether it ended up in the final proof (pos) or not (neg)

ENIGMA-style

The core idea

Learn to recognize and prefer for selection clauses that look like those that contributed to a proof in past successful runs.

➥ [Schulz00], ENIGMA [Jakubův&Urban17], [Loos et al.'17], ...

The "pos/neg"'s of E:

E prover can be asked to output, for every clause selected in a run, whether it ended up in the final proof (pos) or not (neg)

Next comes the ML:

- represent those clauses somehow (features, NNs, ...)
- train a binary classifier on the task
- integrate back with the prover:

ENIGMA-style

The core idea

Learn to recognize and prefer for selection clauses that look like those that contributed to a proof in past successful runs.

➥ [Schulz00], ENIGMA [Jakubův&Urban17], [Loos et al.'17], ...

The "pos/neg"'s of E:
E prover can be asked to output, for every clause selected in a run, whether it ended up in the final proof (pos) or not (neg)

Next comes the ML:

- represent those clauses somehow (features, NNs, ...)
- train a binary classifier on the task
- integrate back with the prover: "try to do more of the pos"

Possible Ways of Integrating the Learnt Advice

Priority:

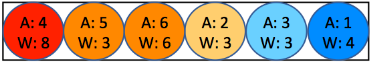
- sort by model's Y/N and tiebreak by age
- 

Possible Ways of Integrating the Learnt Advice

Priority:

- sort by model's Y/N and tiebreak by age
- 

Logits:

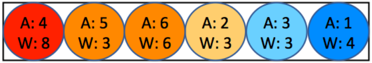
- even a binary classifier internally uses a real value
- 

Possible Ways of Integrating the Learnt Advice

Priority:

- sort by model's Y/N and tiebreak by age
- 

Logits:

- even a binary classifier internally uses a real value
- 

Combine with the original strategy

$\mathcal{S} \oplus \mathcal{M}^{1,0}$: