Descending to Complementarity

Martin Suda*

Czech Technical University in Prague, Czech Republic

AITP, September 2023

*Supported by the project RICAIP no. 857306 under the EU-H2020 programme and the Cost action CA20111 EuroProofNet.

Strategies in modern Automatic Theorem Provers

- The use of proving strategies is an essential element in high-performance ATPs such as E, iProver, or Vampire.

Strategies in modern Automatic Theorem Provers

- The use of proving strategies is an essential element in high-performance ATPs such as E, iProver, or Vampire.
- They are combined into schedules (sequential or parallel) empirically selected to have complementary powers

Descending to Complementarity

Strategies in modern Automatic Theorem Provers

- The use of proving strategies is an essential element in high-performance ATPs such as E, iProver, or Vampire.
- They are combined into schedules (sequential or parallel) empirically selected to have complementary powers

Neural network guided theorem proving

- In recent years, systems like ENIGMA or Deepire were able to dramatically improve the performance of an ATP by integrating neural networks and learning appropriate guiding heuristics

Descending to Complementarity

Strategies in modern Automatic Theorem Provers

- The use of proving strategies is an essential element in high-performance ATPs such as E, iProver, or Vampire.
- They are combined into schedules (sequential or parallel) empirically selected to have complementary powers

Neural network guided theorem proving

- In recent years, systems like ENIGMA or Deepire were able to dramatically improve the performance of an ATP by integrating neural networks and learning appropriate guiding heuristics
- However: always only improved a single base strategy!

Descending to Complementarity

Strategies in modern Automatic Theorem Provers

- The use of proving strategies is an essential element in high-performance ATPs such as E, iProver, or Vampire.
- They are combined into schedules (sequential or parallel) empirically selected to have complementary powers

Neural network guided theorem proving

- In recent years, systems like ENIGMA or Deepire were able to dramatically improve the performance of an ATP by integrating neural networks and learning appropriate guiding heuristics
- However: always only improved a single base strategy!

➥ Could we train a full host of heuristics using gradient descent?

Descending to Complementarity

Strategies in modern Automatic Theorem Provers

- The use of proving strategies is an essential element in high-performance ATPs such as E, iProver, or Vampire.
- They are combined into schedules (sequential or parallel) empirically selected to have complementary powers

Neural network guided theorem proving

- In recent years, systems like ENIGMA or Deepire were able to dramatically improve the performance of an ATP by integrating neural networks and learning appropriate guiding heuristics
- However: always only improved a single base strategy!

➥ Could we train a full host of heuristics using gradient descent?

- Breed specialisations to problems, complementary by design?

Outline

Outline

Embedding problems into a latent space of strategies:

Embedding problems into a latent space of strategies:

- a latent "tweak" variable v_p for every training problem p

Embedding problems into a latent space of strategies:

- a latent “tweak” variable v_p for every training problem p
- initially unknown: e.g., $v_p = \vec{0}$ for every $p \in Train$

The Idea In One Slide

Embedding problems into a latent space of strategies:

- a latent "tweak" variable v_p for every training problem p
- initially unknown: e.g., $v_p = \vec{0}$ for every $p \in \mathit{Train}$
- eventually to represent the "ideal strategy for proving p"

The Idea In One Slide

Embedding problems into a latent space of strategies:

- a latent "tweak" variable v_p for every training problem p
- initially unknown: e.g., $v_p = \vec{0}$ for every $p \in Train$
- eventually to represent the "ideal strategy for proving p"

Condition guidance on v_p:

The Idea In One Slide

Embedding problems into a latent space of strategies:

- a latent "tweak" variable v_p for every training problem p
- initially unknown: e.g., $v_p = \vec{0}$ for every $p \in Train$
- eventually to represent the "ideal strategy for proving p"

Condition guidance on v_p:

- Instead of $N_\theta(input)$, let's use $N_\theta(input, v_p)$ on p

The Idea In One Slide

Embedding problems into a latent space of strategies:

- a latent "tweak" variable v_p for every training problem p
- initially unknown: e.g., $v_p = \vec{0}$ for every $p \in Train$
- eventually to represent the "ideal strategy for proving p"

Condition guidance on v_p:

- Instead of $N_\theta(input)$, let's use $N_\theta(input, v_p)$ on p
- Gradient formula becomes:

$$\nabla_{\theta, v_p} Loss(N_\theta(input, v_p), target)$$

The Idea In One Slide

Embedding problems into a latent space of strategies:

- a latent "tweak" variable v_p for every training problem p
- initially unknown: e.g., $v_p = \vec{0}$ for every $p \in Train$
- eventually to represent the "ideal strategy for proving p"

Condition guidance on v_p:

- Instead of $N_\theta(input)$, let's use $N_\theta(input, v_p)$ on p
- Gradient formula becomes:

$$\nabla_{\theta, v_p} Loss(N_\theta(input, v_p), target)$$

- In training, v_p "travels" in the strategy space to encode a specialization of the general guiding heuristic suitable for p

The Idea In One Slide

Embedding problems into a latent space of strategies:

- a latent "tweak" variable v_p for every training problem p
- initially unknown: e.g., $v_p = \vec{0}$ for every $p \in Train$
- eventually to represent the "ideal strategy for proving p"

Condition guidance on v_p:

- Instead of $N_\theta(input)$, let's use $N_\theta(input, v_p)$ on p
- Gradient formula becomes:

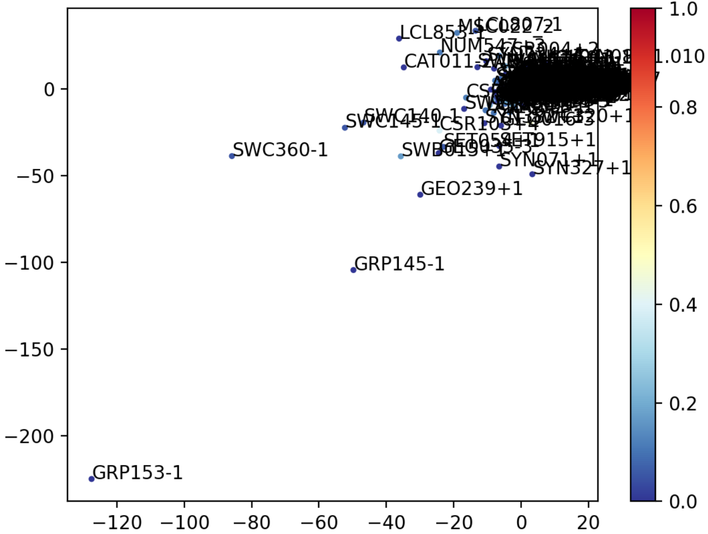
$$\nabla_{\theta, v_p} Loss(N_\theta(input, v_p), target)$$

- In training, v_p "travels" in the strategy space to encode a specialization of the general guiding heuristic suitable for p

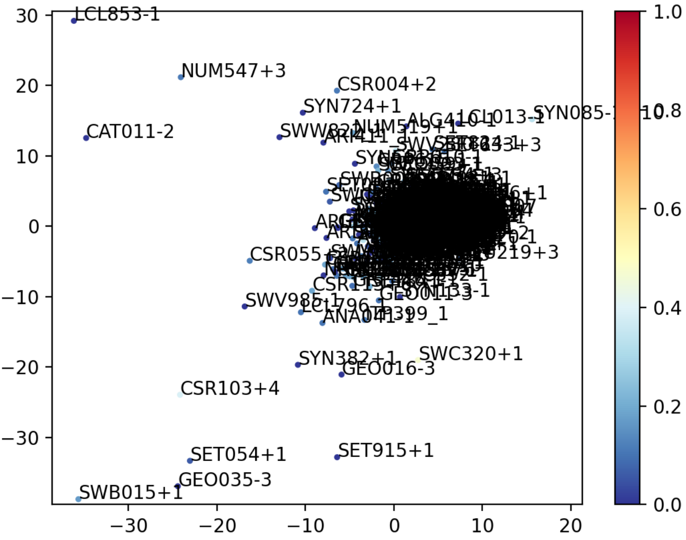
Pick the embedding's dimension well!

A Strategy Map of TPTP?

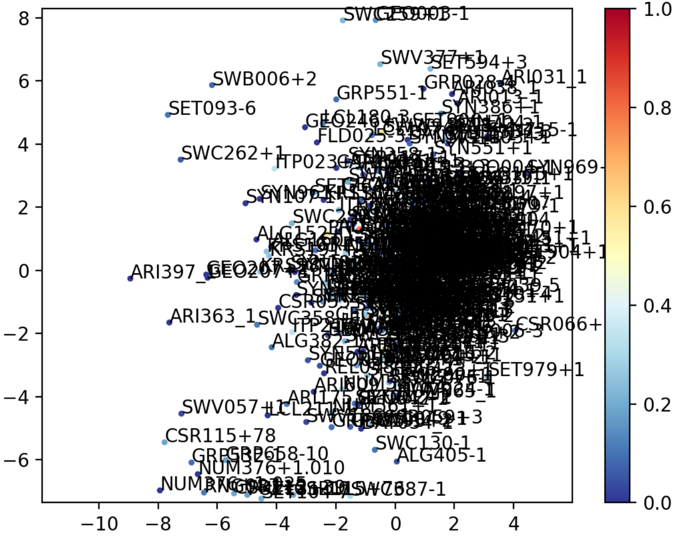
A Strategy Map of TPTP?



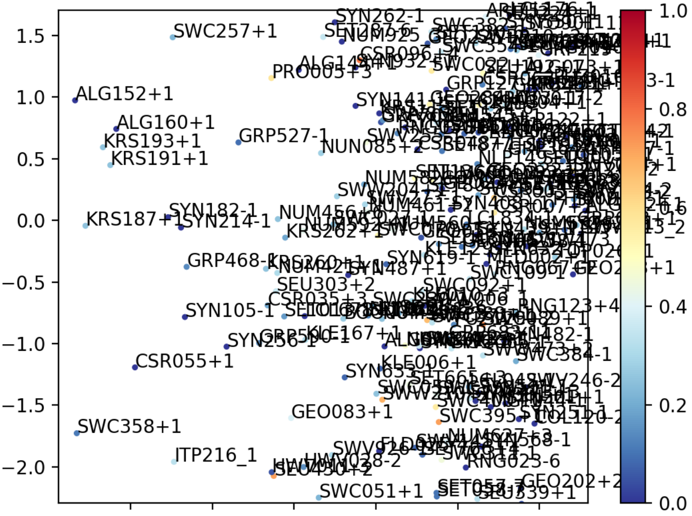
A Strategy Map of TPTP?



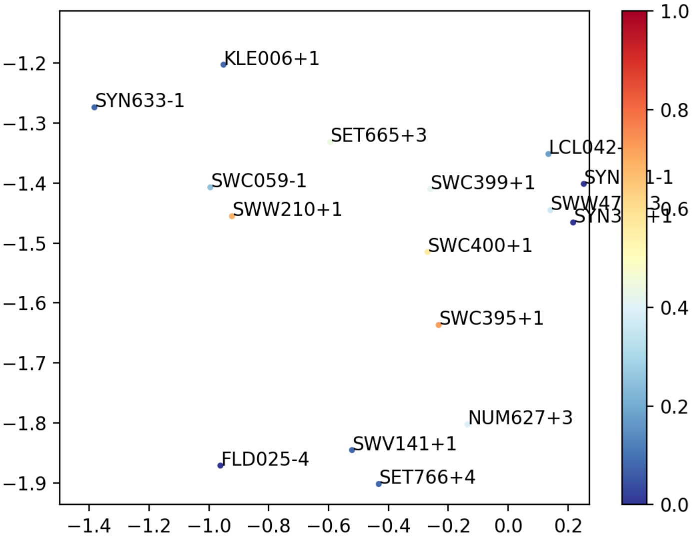
A Strategy Map of TPTP?



A Strategy Map of TPTP?



A Strategy Map of TPTP?



Outline

A Bit of Background

Clause selection

A Bit of Background

Clause selection

- arguably the most important choice point in saturation-based proving

$$\textit{Passive} \rightarrow \boxed{\text{?givenClause?}} \rightarrow \textit{Active}$$

A Bit of Background

Clause selection

- arguably the most important choice point in saturation-based proving

$$\mathit{Passive} \rightarrow \boxed{\text{?givenClause?}} \rightarrow \mathit{Active}$$

"What is the next likeliest clause to participate in the proof?"

A Bit of Background

Clause selection

- arguably the most important choice point in saturation-based proving

$$\textit{Passive} \rightarrow \boxed{\text{?givenClause?}} \rightarrow \textit{Active}$$

"What is the next likeliest clause to participate in the proof?"

Traditional clause selection heuristics

- simple criteria: age, weight, ...
- have a <u>priority queue</u> ordering *Passive* for each criterion
- <u>alternate</u> between selecting from the queues using a fixed <u>ratio</u>

A Bit of Background

Clause selection

- arguably the most important choice point in saturation-based proving

$$\textit{Passive} \rightarrow \boxed{\text{?givenClause?}} \rightarrow \textit{Active}$$

"What is the next likeliest clause to participate in the proof?"

Traditional clause selection heuristics

- simple criteria: age, weight, ...
- have a priority queue ordering *Passive* for each criterion
- alternate between selecting from the queues using a fixed ratio

> "Thus, there is real synergy in the interleaved strategies, which beat not only the individual components but also their union."
>
> — Schulz & Möhrmann, 2016

AITP 2022: Can we, through deep reinforcement learning, somehow re-discover this age/weight-queue alternation?

AITP 2022: Can we, through deep reinforcement learning, somehow re-discover this age/weight-queue alternation?

Architecture design

AITP 2022: Can we, through deep reinforcement learning, somehow re-discover this age/weight-queue alternation?

Architecture design

- simple clause features: age, weight, pos/neg-length, justEq/justNeq, varOcc, goalDist, numSplits

AITP 2022: Can we, through deep reinforcement learning, somehow re-discover this age/weight-queue alternation?

Architecture design

- simple clause features: age, weight, pos/neg-length, justEq/justNeq, varOcc, goalDist, numSplits
- the neural part: $\mathrm{MLP}(\mathit{features}_C) \to \mathit{logit}$

AITP 2022: Can we, through deep reinforcement learning, somehow re-discover this age/weight-queue alternation?

Architecture design

- simple clause features: age, weight, pos/neg-length, justEq/justNeq, varOcc, goalDist, numSplits
- the neural part: $\mathrm{MLP}(\mathit{features}_C) \to \mathit{logit}$
- stateless: no conjecture dependence, no proof planning

AITP 2022: Can we, through deep reinforcement learning, somehow re-discover this age/weight-queue alternation?

Architecture design

- simple clause features: age, weight, pos/neg-length, justEq/justNeq, varOcc, goalDist, numSplits
- the neural part: $\mathrm{MLP}(\mathit{features}_C) \to \mathit{logit}$
- stateless: no conjecture dependence, no proof planning
- yet learning from traces: $\mathit{Passive}_1, \mathit{Passive}_2, \ldots, \mathit{Passive}_n$

AITP 2022: Can we, through deep reinforcement learning, somehow re-discover this age/weight-queue alternation?

Architecture design

- simple clause features: age, weight, pos/neg-length, justEq/justNeq, varOcc, goalDist, numSplits
- the neural part: $\mathrm{MLP}(\mathit{features}_C) \rightarrow \mathit{logit}$
- stateless: no conjecture dependence, no proof planning
- yet learning from traces: $\mathit{Passive}_1, \mathit{Passive}_2, \ldots, \mathit{Passive}_n$
- at each step (e.g., step i), the agent is thought to sample:

$$\mathrm{softmax}\left([\mathrm{MLP}(\mathit{features}_C)]_{C \in \mathit{Passive}_i}\right)$$

AITP 2022: Can we, through deep reinforcement learning, somehow re-discover this age/weight-queue alternation?

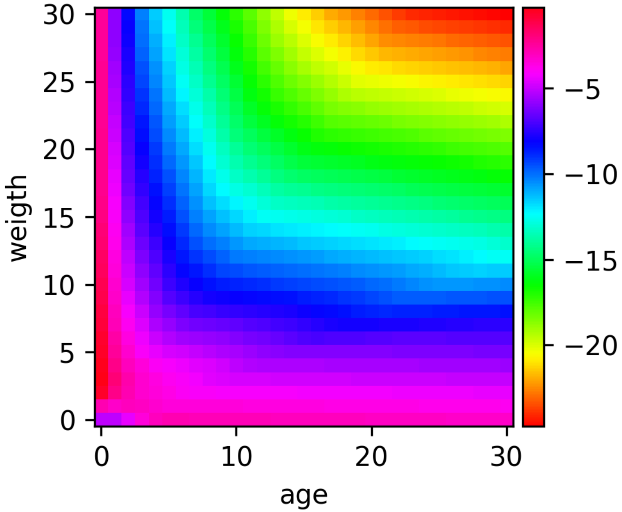
Architecture design

- simple clause features: age, weight, pos/neg-length, justEq/justNeq, varOcc, goalDist, numSplits
- the neural part: $\mathrm{MLP}(\mathit{features}_C) \rightarrow \mathit{logit}$
- stateless: no conjecture dependence, no proof planning
- yet learning from traces: $\mathit{Passive}_1, \mathit{Passive}_2, \ldots, \mathit{Passive}_n$
- at each step (e.g., step i), the agent is thought to sample:

$$\mathrm{softmax}\left([\mathrm{MLP}(\mathit{features}_C)]_{C \in \mathit{Passive}_i}\right)$$

- "reward": learn from proof clauses at each step

One queue to rule them all!



Outline

Early stopping in RL?

- keep some traces aside for validation purposes
- suddenly, a more careful learning rate does not hurt (reevaluation with the updated policy is the costly bit)

Early stopping in RL?

- keep some traces aside for validation purposes
- suddenly, a more careful learning rate does not hurt (reevaluation with the updated policy is the costly bit)

➥ Huge speedups through this!

Challenges of Breeding Strategies for This RL Setup

Early stopping in RL?

- keep some traces aside for validation purposes
- suddenly, a more careful learning rate does not hurt (reevaluation with the updated policy is the costly bit)

➥ Huge speedups through this!

Maintaining the strategy space embeddings v_p:

- for $p \in \mathit{Train}$: updated with gradient descent

Challenges of Breeding Strategies for This RL Setup

Early stopping in RL?

- keep some traces aside for validation purposes
- suddenly, a more careful learning rate does not hurt (reevaluation with the updated policy is the costly bit)

➥ Huge speedups through this!

Maintaining the strategy space embeddings v_p:

- for $p \in \mathit{Train}$: updated with gradient descent
- for $p \in \mathit{Valid}$:

Early stopping in RL?

- keep some traces aside for validation purposes
- suddenly, a more careful learning rate does not hurt (reevaluation with the updated policy is the costly bit)

➥ Huge speedups through this!

Maintaining the strategy space embeddings v_p:

- for $p \in \mathit{Train}$: updated with gradient descent
- for $p \in \mathit{Valid}$: try finding best v_p before contributing to loss

Challenges of Breeding Strategies for This RL Setup

Early stopping in RL?

- keep some traces aside for validation purposes
- suddenly, a more careful learning rate does not hurt (reevaluation with the updated policy is the costly bit)

➥ Huge speedups through this!

Maintaining the strategy space embeddings v_p:

- for $p \in \mathit{Train}$: updated with gradient descent
- for $p \in \mathit{Valid}$: try finding best v_p before contributing to loss
- for $p \in \mathit{Unseen}$: ?

Challenges of Breeding Strategies for This RL Setup

Early stopping in RL?

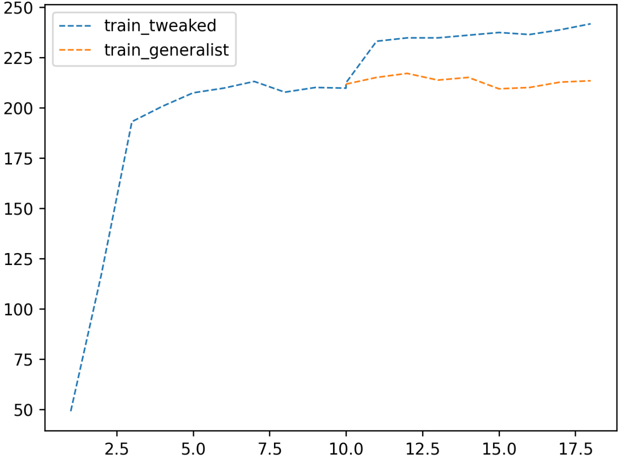
- keep some traces aside for validation purposes
- suddenly, a more careful learning rate does not hurt (reevaluation with the updated policy is the costly bit)

➥ Huge speedups through this!

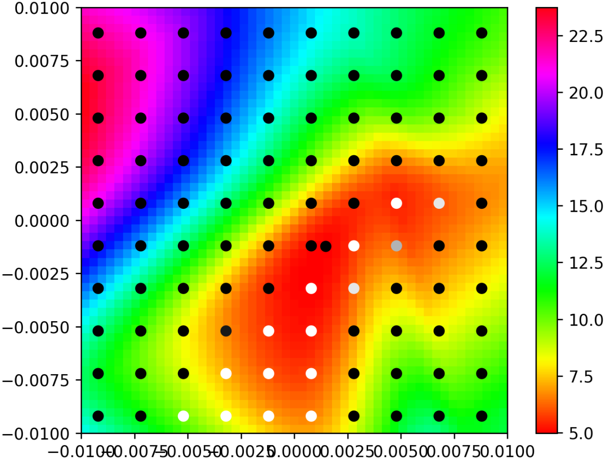
Maintaining the strategy space embeddings v_p:

- for $p \in \mathit{Train}$: updated with gradient descent
- for $p \in \mathit{Valid}$: try finding best v_p before contributing to loss
- for $p \in \mathit{Unseen}$: ?
  ➥ Unconditional schedule like with Snake?

How Well Does It Work?



How Well Does the Reward/Loss Capture the “True Game”?



A trace collected from solving the TPTP problem `GRA002+1`

Concluding Remarks

Summary:

- Neural guiding models for ATPs can be "tweaked" to create targeted proving strategies
- This is basically just gradient descent, but a few aspects require extra care (picking the right dimension, validation, . . .)

Concluding Remarks

Summary:

- Neural guiding models for ATPs can be "tweaked" to create targeted proving strategies
- This is basically just gradient descent, but a few aspects require extra care (picking the right dimension, validation, ...)

Related:

- If we were to add an inference model (i.e., a $IM : p \to v_p$) then that's what, e.g., ENIGMA is already doing with conjecture-conditioned guidance

Concluding Remarks

Summary:

- Neural guiding models for ATPs can be “tweaked” to create targeted proving strategies
- This is basically just gradient descent, but a few aspects require extra care (picking the right dimension, validation, ...)

Related:

- If we were to add an inference model (i.e., a $IM : p \to v_p$) then that’s what, e.g., ENIGMA is already doing with conjecture-conditioned guidance $\Rightarrow$ These are strategies too!

Concluding Remarks

Summary:

- Neural guiding models for ATPs can be "tweaked" to create targeted proving strategies
- This is basically just gradient descent, but a few aspects require extra care (picking the right dimension, validation, ...)

Related:

- If we were to add an inference model (i.e., a $IM : p \to v_p$) then that's what, e.g., ENIGMA is already doing with conjecture-conditioned guidance $\Rightarrow$ These are strategies too!

Thank you!