Robust Strategy Schedule Optimization for an Automatic Theorem Prover

Filip Bártek and Martin Suda

Czech Technical University in Prague, Czech Republic

AITP, September 2023

ATPs and Strategies

State-of-the-art automatic theorem provers for FOL:

- e.g.: E, iProver, Vampire

ATPs and Strategies

State-of-the-art automatic theorem provers for FOL:

- e.g.: E, iProver, Vampire
- many, many ways to configure the search for a proof

ATPs and Strategies

State-of-the-art automatic theorem provers for FOL:

- e.g.: E, iProver, Vampire
- many, many ways to configure the search for a proof
- one such configuration = strategy

ATPs and Strategies

State-of-the-art automatic theorem provers for FOL:

- e.g.: E, iProver, Vampire
- many, many ways to configure the search for a proof
- one such configuration = strategy
  "We define a strategy by fixing values of proof search options"

ATPs and Strategies

State-of-the-art automatic theorem provers for FOL:

- e.g.: E, iProver, Vampire
- many, many ways to configure the search for a proof
- one such configuration = strategy
  "We define a strategy by fixing values of proof search options"

Already Gandalf [Tammet98] knew that:

- there is no single best universal strategy

ATPs and Strategies

State-of-the-art automatic theorem provers for FOL:

- e.g.: E, iProver, Vampire
- many, many ways to configure the search for a proof
- one such configuration = strategy
  "We define a strategy by fixing values of proof search options"

Already Gandalf [Tammet98] knew that:

- there is no single best universal strategy
- it pays off to prepare a whole schedule of strategies to try on a problem and execute them in succession (or in parallel)

ATPs and Strategies

State-of-the-art automatic theorem provers for FOL:

- e.g.: E, iProver, Vampire
- many, many ways to configure the search for a proof
- one such configuration = strategy
  "We define a strategy by fixing values of proof search options"

Already Gandalf [Tammet98] knew that:

- there is no single best universal strategy
- it pays off to prepare a whole schedule of strategies to try on a problem and execute them in succession (or in parallel)
- many short runs of complementary strategies will usually beat a single long run of a in-theory-complete single strategy

ATPs and Strategies

State-of-the-art automatic theorem provers for FOL:

- e.g.: E, iProver, Vampire
- many, many ways to configure the search for a proof
- one such configuration = strategy
  "We define a strategy by fixing values of proof search options"

Already Gandalf [Tammet98] knew that:

- there is no single best universal strategy
- it pays off to prepare a whole schedule of strategies to try on a problem and execute them in succession (or in parallel)
- many short runs of complementary strategies will usually beat a single long run of a in-theory-complete single strategy

DEMO: `./vampire Problems/PUZ/PUZ039-1.p`

CASC 2022 Competition Result Summary (partial)

Typed First-order Theorems +*-/	SnakeFor 1.0	cvc5 1.0	*Vampire* 4.5	Vampire 4.7	iProver 3.6
Solved/250	218/250	195/250	192/250	187/250	138/250
Solutions	218 87%	195 78%	192 76%	187 74%	137 54%

First-order Theorems	SnakeFor 1.0	Vampire 4.7	*Vampire* 4.6	E 3.0	iProver 3.6
Solved/500	460/500	451/500	448/500	384/500	365/500
Solutions	460 92%	451 90%	448 89%	384 76%	365 73%

First-order Non-theorems	*Vampire* 4.6	Vampire 4.7	SnakeFor 1.0	cvc5 1.0	iProver 3.6
Solved/250	167/250	160/250	159/250	78/250	63/250
Solutions	167 66%	160 64%	159 63%	78 31%	63 25%

Unit Equality CNF	Twee 2.4.1	*Twee* 2.4	E 3.0	SnakeFor 1.0	Vampire 4.7
Solved/250	216/250	215/250	212/250	207/250	152/250
Solutions	216 86%	215 86%	212 84%	207 82%	152 60%

SnakeForV4.7: a strategy discovery and schedule construction tool applied to Vampire 4.7

CASC 2022 Competition Result Summary (partial)

Typed First-order Theorems +*-/	SnakeFor 1.0	cvc5 1.0	Vampire 4.5	Vampire 4.7	iProver 3.6
Solved/250	218/250	195/250	192/250	187/250	138/250
Solutions	218 87%	195 78%	192 76%	187 74%	137 54%

First-order Theorems	SnakeFor 1.0	Vampire 4.7	Vampire 4.6	E 3.0	iProver 3.6
Solved/500	460/500	451/500	448/500	384/500	365/500
Solutions	460 92%	451 90%	448 89%	384 76%	365 73%

First-order Non-theorems	Vampire 4.6	Vampire 4.7	SnakeFor 1.0	cvc5 1.0	iProver 3.6
Solved/250	167/250	160/250	159/250	78/250	63/250
Solutions	167 66%	160 64%	159 63%	78 31%	63 25%

Unit Equality CNF	Twee 2.4.1	Twee 2.4	E 3.0	SnakeFor 1.0	Vampire 4.7
Solved/250	216/250	215/250	212/250	207/250	152/250
Solutions	216 86%	215 86%	212 84%	207 82%	152 60%

SnakeForV4.7: a strategy discovery and schedule construction tool applied to Vampire 4.7 (and running in demonstration-only mode)

A Tribute to Spider

Spider [Voronkov2013]

"Spider was used since 2010 and has been a secret weapon behind Vampire's success at the CASC competitions."

A Tribute to Spider

Spider [Voronkov2013]

"Spider was used since 2010 and has been a secret weapon behind Vampire's success at the CASC competitions."

What we know about Spider:

A Tribute to Spider

Spider [Voronkov2013]

"Spider was used since 2010 and has been a secret weapon behind Vampire's success at the CASC competitions."

What we know about Spider:

- tries out random strategies to solve medium to hard problems

A Tribute to Spider

Spider [Voronkov2013]

"Spider was used since 2010 and has been a secret weapon behind Vampire's success at the CASC competitions."

What we know about Spider:

- tries out random strategies to solve medium to hard problems
- locally optimizes a strategy just for its problem
  (preferring default values where the choice seems irrelevant)

A Tribute to Spider

Spider [Voronkov2013]

"Spider was used since 2010 and has been a secret weapon behind Vampire's success at the CASC competitions."

What we know about Spider:

- tries out random strategies to solve medium to hard problems
- locally optimizes a strategy just for its problem
  (preferring default values where the choice seems irrelevant)
- once done, evaluates a strategy on all problems

A Tribute to Spider

Spider [Voronkov2013]

"Spider was used since 2010 and has been a secret weapon behind Vampire's success at the CASC competitions."

What we know about Spider:

- tries out random strategies to solve medium to hard problems
- locally optimizes a strategy just for its problem
  (preferring default values where the choice seems irrelevant)
- once done, evaluates a strategy on all problems
- poses schedule construction as an optimization problem:
  "Cover all known-to-be-solvable problems by a subset of strategies with their time limits summing up to $\leq K$."

A Tribute to Spider

Spider [Voronkov2013]

"Spider was used since 2010 and has been a secret weapon behind Vampire's success at the CASC competitions."

What we know about Spider:

- tries out random strategies to solve medium to hard problems
- locally optimizes a strategy just for its problem
  (preferring default values where the choice seems irrelevant)
- once done, evaluates a strategy on all problems
- poses schedule construction as an optimization problem:
  "Cover all known-to-be-solvable problems by a subset of strategies with their time limits summing up to $\leq K$."

General mindset:

A Tribute to Spider

Spider [Voronkov2013]

"Spider was used since 2010 and has been a secret weapon behind Vampire's success at the CASC competitions."

What we know about Spider:

- tries out random strategies to solve medium to hard problems
- locally optimizes a strategy just for its problem
  (preferring default values where the choice seems irrelevant)
- once done, evaluates a strategy on all problems
- poses schedule construction as an optimization problem:
  "Cover all known-to-be-solvable problems by a subset of strategies with their time limits summing up to $\leq K$."

General mindset:

- Computationally not cheap: get some CPUs and use them!

A Tribute to Spider

Spider [Voronkov2013]

"Spider was used since 2010 and has been a secret weapon behind Vampire's success at the CASC competitions."

What we know about Spider:

- tries out random strategies to solve medium to hard problems
- locally optimizes a strategy just for its problem
  (preferring default values where the choice seems irrelevant)
- once done, evaluates a strategy on all problems
- poses schedule construction as an optimization problem:
  "Cover all known-to-be-solvable problems by a subset of strategies with their time limits summing up to $\leq K$."

General mindset:

- Computationally not cheap: get some CPUs and use them!
- Don't bother with any form of ML-powered "strategy selection"

So who is the Snake?

Our (my and Filip's) story:

So who is the Snake?

Our (my and Filip's) story:

- Andrei, as the sole operator of Spider, was getting increasingly busy with EasyChair in recent recent years

So who is the Snake?

Our (my and Filip's) story:

- Andrei, as the sole operator of Spider, was getting increasingly busy with EasyChair in recent recent years
- CASC schedule of Vampire 4.7 used in 2022 was from 2019!

So who is the Snake?

Our (my and Filip's) story:

- Andrei, as the sole operator of Spider, was getting increasingly busy with EasyChair in recent recent years
- CASC schedule of Vampire 4.7 used in 2022 was from 2019!
- so we decided to develop our own tool to try out new ideas and gain flexibility / independence

So who is the Snake?

Our (my and Filip's) story:

- Andrei, as the sole operator of Spider, was getting increasingly busy with EasyChair in recent recent years
- CASC schedule of Vampire 4.7 used in 2022 was from 2019!
- so we decided to develop our own tool to try out new ideas and gain flexibility / independence

What's new in Snake?

So who is the Snake?

Our (my and Filip's) story:

- Andrei, as the sole operator of Spider, was getting increasingly busy with EasyChair in recent recent years
- CASC schedule of Vampire 4.7 used in 2022 was from 2019!
- so we decided to develop our own tool to try out new ideas and gain flexibility / independence

What's new in Snake?

- stochastic view of strategies treated as Las Vegas algorithms

So who is the Snake?

Our (my and Filip's) story:

- Andrei, as the sole operator of Spider, was getting increasingly busy with EasyChair in recent recent years
- CASC schedule of Vampire 4.7 used in 2022 was from 2019!
- so we decided to develop our own tool to try out new ideas and gain flexibility / independence

What's new in Snake?

- stochastic view of strategies treated as Las Vegas algorithms
- local strategy improvement with many random probes

So who is the Snake?

Our (my and Filip's) story:

- Andrei, as the sole operator of Spider, was getting increasingly busy with EasyChair in recent recent years
- CASC schedule of Vampire 4.7 used in 2022 was from 2019!
- so we decided to develop our own tool to try out new ideas and gain flexibility / independence

What's new in Snake?

- stochastic view of strategies treated as Las Vegas algorithms
- local strategy improvement with many random probes
- greedy weighted cover for the schedule creation phase

So who is the Snake?

So who is the Snake?

*Illustration by Sibylle Ortner, used with permission.

Warning: CASC's TPTP problems threatened to be shuffled!

Warning: CASC's TPTP problems threatened to be shuffled!

A 2019 experiment

Use `tptp4X -trandomize` from the TPTP toolset to:

- randomize the order of commutative logical operations
- randomize the order of formulas

Warning: CASC's TPTP problems threatened to be shuffled!

A 2019 experiment

Use `tptp4X -trandomize` from the TPTP toolset to:

- randomize the order of commutative logical operations
- randomize the order of formulas

Can we solve more problems? (with a single strategy)

Warning: CASC's TPTP problems threatened to be shuffled!

A 2019 experiment

Use `tptp4X -trandomize` from the TPTP toolset to:

- randomize the order of commutative logical operations
- randomize the order of formulas

Can we solve more problems? (with a single strategy)

configuration	solved	uniques	additional
straight	8612	53	8612
shuffled1	8773	60	345
shuffled2	8788	85	128
shuffled3	8775	48	48

Warning: CASC's TPTP problems threatened to be shuffled!

A 2019 experiment

Use `tptp4X -trandomize` from the TPTP toolset to:

- randomize the order of commutative logical operations
- randomize the order of formulas

Can we solve more problems? (with a single strategy)

configuration	solved	uniques	additional
straight	8612	53	8612
shuffled1	8773	60	345
shuffled2	8788	85	128
shuffled3	8775	48	48

Can now be invoked from Vampire (`--shuffle_input on`)
as well as "internal" shuffling (`--random_traversals on`)

Step 1: Random Strategies to Discover New Solutions

Our Strategy Space

Step 1: Random Strategies to Discover New Solutions

Our Strategy Space

- $\sim$ 100 options: bool, categorical, numerical, ratios

Step 1: Random Strategies to Discover New Solutions

Our Strategy Space

- $\sim$ 100 options: bool, categorical, numerical, ratios
- not simply a cartesian product:
  - dependent options

Step 1: Random Strategies to Discover New Solutions

Our Strategy Space

- $\sim$ 100 options: bool, categorical, numerical, ratios
- not simply a cartesian product:
  - dependent options
  - conflicting combinations

Step 1: Random Strategies to Discover New Solutions

Our Strategy Space

- $\sim$ 100 options: bool, categorical, numerical, ratios
- not simply a cartesian product:
  - dependent options
  - conflicting combinations (please avoid "forbidden clauses")

Step 1: Random Strategies to Discover New Solutions

Our Strategy Space

- $\sim$ 100 options: bool, categorical, numerical, ratios
- not simply a cartesian product:
  - dependent options
  - conflicting combinations (please avoid "forbidden clauses")

How to sample good random strategies?

Step 1: Random Strategies to Discover New Solutions

Our Strategy Space

- $\sim$ 100 options: bool, categorical, numerical, ratios
- not simply a cartesian product:
  - dependent options
  - conflicting combinations (please avoid "forbidden clauses")

How to sample good random strategies?

- What does it mean in the first place?

Step 1: Random Strategies to Discover New Solutions

Our Strategy Space

- $\sim$ 100 options: bool, categorical, numerical, ratios
- not simply a cartesian product:
  - dependent options
  - conflicting combinations (please avoid "forbidden clauses")

How to sample good random strategies?

- What does it mean in the first place?
- Uniform was good enough. But expert knowledge helps!

Step 1: Random Strategies to Discover New Solutions

Our Strategy Space

- $\sim$ 100 options: bool, categorical, numerical, ratios
- not simply a cartesian product:
  - dependent options
  - conflicting combinations (please avoid "forbidden clauses")

How to sample good random strategies?

- What does it mean in the first place?
- Uniform was good enough. But expert knowledge helps!
- Something more principled: open research (mini)topic

Step 1: Random Strategies to Discover New Solutions

Our Strategy Space

- $\sim$ 100 options: bool, categorical, numerical, ratios
- not simply a cartesian product:
  - dependent options
  - conflicting combinations (please avoid "forbidden clauses")

How to sample good random strategies?

- What does it mean in the first place?
- Uniform was good enough. But expert knowledge helps!
- Something more principled: open research (mini)topic

➥ Snake: shuffling is on and we sample `--random_seed`

Step 1: Random Strategies to Discover New Solutions

Our Strategy Space

- $\sim$ 100 options: bool, categorical, numerical, ratios
- not simply a cartesian product:
  - dependent options
  - conflicting combinations (please avoid "forbidden clauses")

How to sample good random strategies?

- What does it mean in the first place?
- Uniform was good enough. But expert knowledge helps!
- Something more principled: open research (mini)topic

➥ Snake: shuffling is on and we sample `--random_seed`

How to pick the next problem?

Step 1: Random Strategies to Discover New Solutions

Our Strategy Space

- $\sim$ 100 options: bool, categorical, numerical, ratios
- not simply a cartesian product:
  - dependent options
  - conflicting combinations (please avoid "forbidden clauses")

How to sample good random strategies?

- What does it mean in the first place?
- Uniform was good enough. But expert knowledge helps!
- Something more principled: open research (mini)topic

➥ Snake: shuffling is on and we sample `--random_seed`

How to pick the next problem?

- focus on yet unsolved ones

Step 1: Random Strategies to Discover New Solutions

Our Strategy Space

- $\sim$ 100 options: bool, categorical, numerical, ratios
- not simply a cartesian product:
  - dependent options
  - conflicting combinations (please avoid "forbidden clauses")

How to sample good random strategies?

- What does it mean in the first place?
- Uniform was good enough. But expert knowledge helps!
- Something more principled: open research (mini)topic

➥ Snake: shuffling is on and we sample `--random_seed`

How to pick the next problem?

- focus on yet unsolved ones
- focus on speeding up the best known solution

Step 2: Improve the Strategy on its Problem

Local search for strategy improvement

- vary one option at a time
- iterate over all (non-default) options for several rounds

Step 2: Improve the Strategy on its Problem

Local search for strategy improvement

- vary one option at a time
- iterate over all (non-default) options for several rounds

Optimization criteria

Step 2: Improve the Strategy on its Problem

Local search for strategy improvement

- vary one option at a time
- iterate over all (non-default) options for several rounds

Optimization criteria

1. does the probability of solving the problem improve?

Step 2: Improve the Strategy on its Problem

Local search for strategy improvement

- vary one option at a time
- iterate over all (non-default) options for several rounds

Optimization criteria

1. does the probability of solving the problem improve?
2. does the time to solve the problem improve?

Step 2: Improve the Strategy on its Problem

Local search for strategy improvement

- vary one option at a time
- iterate over all (non-default) options for several rounds

Optimization criteria

1. does the probability of solving the problem improve?
2. does the time to solve the problem improve?
3. does the new value look more reasonable?

Step 2: Improve the Strategy on its Problem

Local search for strategy improvement

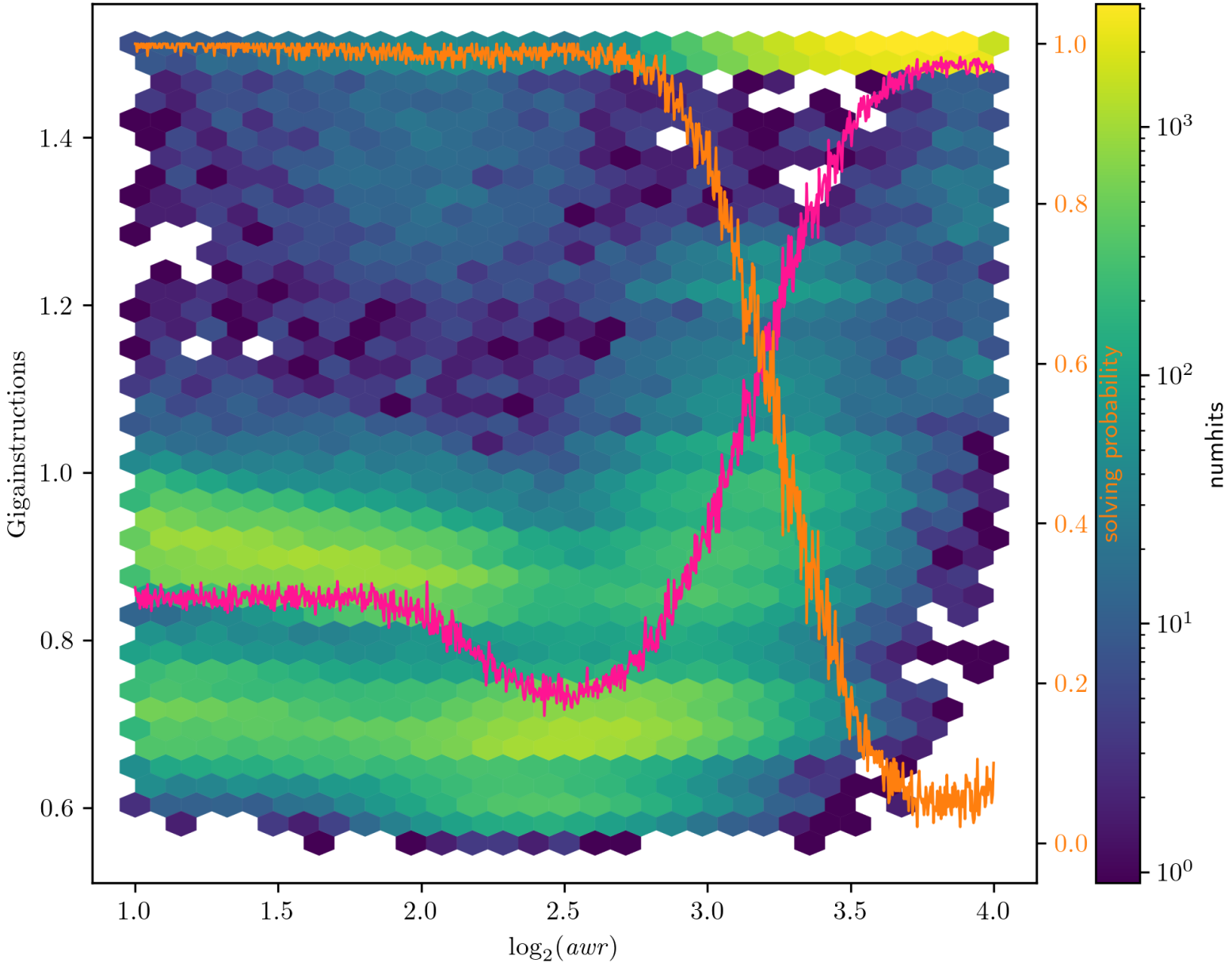
- vary one option at a time
- iterate over all (non-default) options for several rounds

Optimization criteria

1. does the probability of solving the problem improve?
2. does the time to solve the problem improve?
3. does the new value look more reasonable?

DEMO: `protocol.txt`

DEMO cont.: an AWR Plot



Step 3: Schedule Construction

So now we have all the discovered strategies evaluated on all the problems of interest: E_p^s, for $s \in \mathit{Strats}$ and $p \in \mathit{Probs}$.

Step 3: Schedule Construction

So now we have all the discovered strategies evaluated on all the problems of interest: E_p^s, for $s \in \mathit{Strats}$ and $p \in \mathit{Probs}$.

Weighted set cover formulation:

- define sets: $S_{(s,i)} = \{p \| E_p^s \leq i\}$ with weights $w_{(s,i)} = i$

Step 3: Schedule Construction

So now we have all the discovered strategies evaluated on all the problems of interest: E_p^s, for $s \in \mathit{Strats}$ and $p \in \mathit{Probs}$.

Weighted set cover formulation:

- define sets: $S_{(s,i)} = \{p \| E_p^s \leq i\}$ with weights $w_{(s,i)} = i$
- problems covered by a schedule $\mathcal{S}$: $c(\mathcal{S}) = \bigcup_{(s,i)\in\mathcal{S}} S_{(s,i)}$

Step 3: Schedule Construction

So now we have all the discovered strategies evaluated on all the problems of interest: E_p^s, for $s \in Strats$ and $p \in Probs$.

Weighted set cover formulation:

- define sets: $S_{(s,i)} = \{p \| E_p^s \leq i\}$ with weights $w_{(s,i)} = i$
- problems covered by a schedule $\mathcal{S}$: $c(\mathcal{S}) = \bigcup_{(s,i)\in\mathcal{S}} S_{(s,i)}$
- task: find a schedule $\mathcal{S}$ covering all solvable problems:

$$c(\mathcal{S}) = \bigcup_{s\in Strats} \{p \| E_p^s < \infty\},$$

minimizing the cost $\sum_{(s,i)\in\mathcal{S}} w_{(s,i)}$

Step 3: Schedule Construction

So now we have all the discovered strategies evaluated on all the problems of interest: E_p^s, for $s \in \mathit{Strats}$ and $p \in \mathit{Probs}$.

Weighted set cover formulation:

- define sets: $S_{(s,i)} = \{p \| E_p^s \leq i\}$ with weights $w_{(s,i)} = i$
- problems covered by a schedule $\mathcal{S}$: $c(\mathcal{S}) = \bigcup_{(s,i)\in\mathcal{S}} S_{(s,i)}$
- task: find a schedule $\mathcal{S}$ covering all solvable problems:

$$c(\mathcal{S}) = \bigcup_{s\in\mathit{Strats}} \{p \| E_p^s < \infty\},$$

minimizing the cost $\sum_{(s,i)\in\mathcal{S}} w_{(s,i)}$

Greedy Weighted Set Cover

Starting with the empty schedule $\mathcal{S} = \emptyset$
Loop adding to $\mathcal{S}$ an (s, i) maximizing $|S_{(s,i)} \setminus c(\mathcal{S})|/w_{(s,i)}$

Greedy Weighted Set Cover – "Advanced Topics"

Some observations:

Greedy Weighted Set Cover – "Advanced Topics"

Some observations:

- If, at the end, both $(s, i) \in \mathcal{S}$ and $(s, j) \in \mathcal{S}$ for $i < j$, we can drop (s, i) since, after all, $S_{(s,i)} \subseteq S_{(s,j)}$

Greedy Weighted Set Cover – "Advanced Topics"

Some observations:

- If, at the end, both $(s, i) \in \mathcal{S}$ and $(s, j) \in \mathcal{S}$ for $i < j$, we can drop (s, i) since, after all, $S_{(s,i)} \subseteq S_{(s,j)}$
- Redefine schedule to $\mathcal{S} : \mathit{Strats} \to \mathbb{N}$

Greedy Weighted Set Cover – "Advanced Topics"

Some observations:

- If, at the end, both $(s, i) \in \mathcal{S}$ and $(s, j) \in \mathcal{S}$ for $i < j$, we can drop (s, i) since, after all, $S_{(s,i)} \subseteq S_{(s,j)}$
- Redefine schedule to $\mathcal{S} : \mathit{Strats} \to \mathbb{N}$ (starting as constant 0)

Greedy Weighted Set Cover – "Advanced Topics"

Some observations:

- If, at the end, both $(s, i) \in \mathcal{S}$ and $(s, j) \in \mathcal{S}$ for $i < j$, we can drop (s, i) since, after all, $S_{(s,i)} \subseteq S_{(s,j)}$
- Redefine schedule to $\mathcal{S} : \mathit{Strats} \to \mathbb{N}$ (starting as constant 0)
- and adapt the costs as we go:

$$\text{if } \mathcal{S}(s) = i \text{ then set } w_{(s,j)} = \max(0, j - i)$$

Greedy Weighted Set Cover – "Advanced Topics"

Some observations:

- If, at the end, both $(s, i) \in \mathcal{S}$ and $(s, j) \in \mathcal{S}$ for $i < j$, we can drop (s, i) since, after all, $S_{(s,i)} \subseteq S_{(s,j)}$
- Redefine schedule to $\mathcal{S} : \mathit{Strats} \to \mathbb{N}$ (starting as constant 0)
- and adapt the costs as we go:

$$\text{if } \mathcal{S}(s) = i \text{ then set } w_{(s,j)} = \max(0, j - i)$$

Actually, can aim to construct a "probabilistic" schedule

Greedy Weighted Set Cover – "Advanced Topics"

Some observations:

- If, at the end, both $(s, i) \in \mathcal{S}$ and $(s, j) \in \mathcal{S}$ for $i < j$, we can drop (s, i) since, after all, $S_{(s,i)} \subseteq S_{(s,j)}$
- Redefine schedule to $\mathcal{S} : \mathit{Strats} \to \mathbb{N}$ (starting as constant 0)
- and adapt the costs as we go:

$$\text{if } \mathcal{S}(s) = i \text{ then set } w_{(s,j)} = \max(0, j - i)$$

Actually, can aim to construct a "probabilistic" schedule

- Collect more data:

Greedy Weighted Set Cover – "Advanced Topics"

Some observations:

- If, at the end, both $(s, i) \in \mathcal{S}$ and $(s, j) \in \mathcal{S}$ for $i < j$, we can drop (s, i) since, after all, $S_{(s,i)} \subseteq S_{(s,j)}$
- Redefine schedule to $\mathcal{S} : \mathit{Strats} \to \mathbb{N}$ (starting as constant 0)
- and adapt the costs as we go:

$$\text{if } \mathcal{S}(s) = i \text{ then set } w_{(s,j)} = \max(0, j - i)$$

Actually, can aim to construct a "probabilistic" schedule

- Collect more data: e.g., (s, i) solves p with probability 0.8

Greedy Weighted Set Cover – "Advanced Topics"

Some observations:

- If, at the end, both $(s, i) \in \mathcal{S}$ and $(s, j) \in \mathcal{S}$ for $i < j$, we can drop (s, i) since, after all, $S_{(s,i)} \subseteq S_{(s,j)}$
- Redefine schedule to $\mathcal{S} : \mathit{Strats} \to \mathbb{N}$ (starting as constant 0)
- and adapt the costs as we go:

$$\text{if } \mathcal{S}(s) = i \text{ then set } w_{(s,j)} = \max(0, j - i)$$

Actually, can aim to construct a "probabilistic" schedule

- Collect more data: e.g., (s, i) solves p with probability 0.8
- Assuming strategy independence: if current $\mathcal{S}$ solves p with prob. 0.5, adding (s, i) to $\mathcal{S}$ will improve by $0.8 \cdot 0.5$ to 0.9

Wrapping Up

Strategy schedules substantially boost prover performance

Wrapping Up

Strategy schedules substantially boost prover performance

- the "best known solutions" keep improving for very very long!

Wrapping Up

Strategy schedules substantially boost prover performance

- the "best known solutions" keep improving for very very long!
- then just evaluate everywhere and optimize out a schedule

Wrapping Up

Strategy schedules substantially boost prover performance

- the "best known solutions" keep improving for very very long!
- then just evaluate everywhere and optimize out a schedule

Are we just memorising solutions to win CASC?

Wrapping Up

Strategy schedules substantially boost prover performance

- the "best known solutions" keep improving for very very long!
- then just evaluate everywhere and optimize out a schedule

Are we just memorising solutions to win CASC?

- CASC is easy!

Wrapping Up

Strategy schedules substantially boost prover performance

- the "best known solutions" keep improving for very very long!
- then just evaluate everywhere and optimize out a schedule

Are we just memorising solutions to win CASC?

- CASC is easy! (Can cover all known solvable in 8*120s easily!)

Wrapping Up

Strategy schedules substantially boost prover performance

- the "best known solutions" keep improving for very very long!
- then just evaluate everywhere and optimize out a schedule

Are we just memorising solutions to win CASC?

- CASC is easy! (Can cover all known solvable in 8*120s easily!)
- We don't know how well this generalises.

Wrapping Up

Strategy schedules substantially boost prover performance

- the "best known solutions" keep improving for very very long!
- then just evaluate everywhere and optimize out a schedule

Are we just memorising solutions to win CASC?

- CASC is easy! (Can cover all known solvable in 8*120s easily!)
- We don't know how well this generalises. (Ablations needed!)

Wrapping Up

Strategy schedules substantially boost prover performance

- the "best known solutions" keep improving for very very long!
- then just evaluate everywhere and optimize out a schedule

Are we just memorising solutions to win CASC?

- CASC is easy! (Can cover all known solvable in 8*120s easily!)
- We don't know how well this generalises. (Ablations needed!)

Spider was doing this for years now!

Wrapping Up

Strategy schedules substantially boost prover performance

- the "best known solutions" keep improving for very very long!
- then just evaluate everywhere and optimize out a schedule

Are we just memorising solutions to win CASC?

- CASC is easy! (Can cover all known solvable in 8*120s easily!)
- We don't know how well this generalises. (Ablations needed!)

Spider was doing this for years now!

How could Snake be more robust?

Wrapping Up

Strategy schedules substantially boost prover performance

- the "best known solutions" keep improving for very very long!
- then just evaluate everywhere and optimize out a schedule

Are we just memorising solutions to win CASC?

- CASC is easy! (Can cover all known solvable in 8*120s easily!)
- We don't know how well this generalises. (Ablations needed!)

Spider was doing this for years now!

How could Snake be more robust?

- randomization during minimization (the step 2) tends to make a strategy s robustly successful on its p

Wrapping Up

Strategy schedules substantially boost prover performance

- the "best known solutions" keep improving for very very long!
- then just evaluate everywhere and optimize out a schedule

Are we just memorising solutions to win CASC?

- CASC is easy! (Can cover all known solvable in 8*120s easily!)
- We don't know how well this generalises. (Ablations needed!)

Spider was doing this for years now!

How could Snake be more robust?

- randomization during minimization (the step 2) tends to make a strategy s robustly successful on its p
- probabilistic schedule more robust as it is playing against (and not with) "the shuffler"

Wrapping Up

Strategy schedules substantially boost prover performance

- the "best known solutions" keep improving for very very long!
- then just evaluate everywhere and optimize out a schedule

Are we just memorising solutions to win CASC?

- CASC is easy! (Can cover all known solvable in 8*120s easily!)
- We don't know how well this generalises. (Ablations needed!)

Spider was doing this for years now!

How could Snake be more robust?

- randomization during minimization (the step 2) tends to make a strategy s robustly successful on its p
- probabilistic schedule more robust as it is playing against (and not with) "the shuffler"
- greedy weighted set cover

Wrapping Up

Strategy schedules substantially boost prover performance

- the "best known solutions" keep improving for very very long!
- then just evaluate everywhere and optimize out a schedule

Are we just memorising solutions to win CASC?

- CASC is easy! (Can cover all known solvable in 8*120s easily!)
- We don't know how well this generalises. (Ablations needed!)

Spider was doing this for years now!

How could Snake be more robust?

- randomization during minimization (the step 2) tends to make a strategy s robustly successful on its p
- probabilistic schedule more robust as it is playing against (and not with) "the shuffler"
- greedy weighted set cover (by the way, notice it is "anytime")

Wrapping Up

Strategy schedules substantially boost prover performance

- the "best known solutions" keep improving for very very long!
- then just evaluate everywhere and optimize out a schedule

Are we just memorising solutions to win CASC?

- CASC is easy! (Can cover all known solvable in 8*120s easily!)
- We don't know how well this generalises. (Ablations needed!)

Spider was doing this for years now!

How could Snake be more robust?

- randomization during minimization (the step 2) tends to make a strategy s robustly successful on its p
- probabilistic schedule more robust as it is playing against (and not with) "the shuffler"
- greedy weighted set cover (by the way, notice it is "anytime") should have harder time overfitting than an optimal schedule

Wrapping Up

Strategy schedules substantially boost prover performance

- the "best known solutions" keep improving for very very long!
- then just evaluate everywhere and optimize out a schedule

Are we just memorising solutions to win CASC?

- CASC is easy! (Can cover all known solvable in 8*120s easily!)
- We don't know how well this generalises. (Ablations needed!)

Spider was doing this for years now!

How could Snake be more robust?

- randomization during minimization (the step 2) tends to make a strategy s robustly successful on its p
- probabilistic schedule more robust as it is playing against (and not with) "the shuffler"
- greedy weighted set cover (by the way, notice it is "anytime") should have harder time overfitting than an optimal schedule

Thank you!

One more experiment with randomness

Clause Selection and Age-weight Ratio

Vampire alternates between selecting the next given clause by age (old first) and by weight (light first) under a given ratio.

One more experiment with randomness

Clause Selection and Age-weight Ratio

Vampire alternates between selecting the next given clause by age (old first) and by weight (light first) under a given ratio.

Normally, this alternation is regular. What if we change it to probabilistic?

One more experiment with randomness

Clause Selection and Age-weight Ratio

Vampire alternates between selecting the next given clause by age (old first) and by weight (light first) under a given ratio.

Normally, this alternation is regular. What if we change it to probabilistic?

configuration	solved	uniques	additional
base	8725	12	8725
rnd1	8747	8	91
rnd2	8744	16	37
rnd3	8768	23	37
rnd4	8735	14	21
rnd5	8741	16	16

base = `-sa discount -awr 1:1 -t 10`

Related work

Related work ATP:

- MaLeS [Kühlwein&Urban, 2015]
- BliStr → BliStrTune → EmpireTune → Grackle [Urban, Jakubův, . . .]
- HOS-ML [Holden& Korovin, 2021]
- Genetic breeding [Schäfer and Schulz, 2015]

Related work SMT:

- MachSMT [Scott et al., 2021]

Related work algorithm configuration etc:

- [Hoos,Hutter,. . .]