Regularization in Spider-Style Strategy Discovery and Schedule Construction

Filip Bártek, Karel Chvalovský, Martin Suda

Czech Technical University in Prague, Czech Republic

July 5, 2024

This work was supported by the Czech Science Foundation grants 20-06390Y and 24-12759S, the European Regional Development Fund under the Czech project AI&Reasoning no. CZ.02.1.01/0.0/0.0/15_003/0000466, the project RICAIP no. 857306 under the EU-H2020 programme, and the Grant Agency of the Czech Technical University in Prague, grant no. SGS20/215/OHK3/3T/37.

Automated theorem proving and strategies

- Automated theorem provers (Vampire, E, etc.) expose lots of parameters

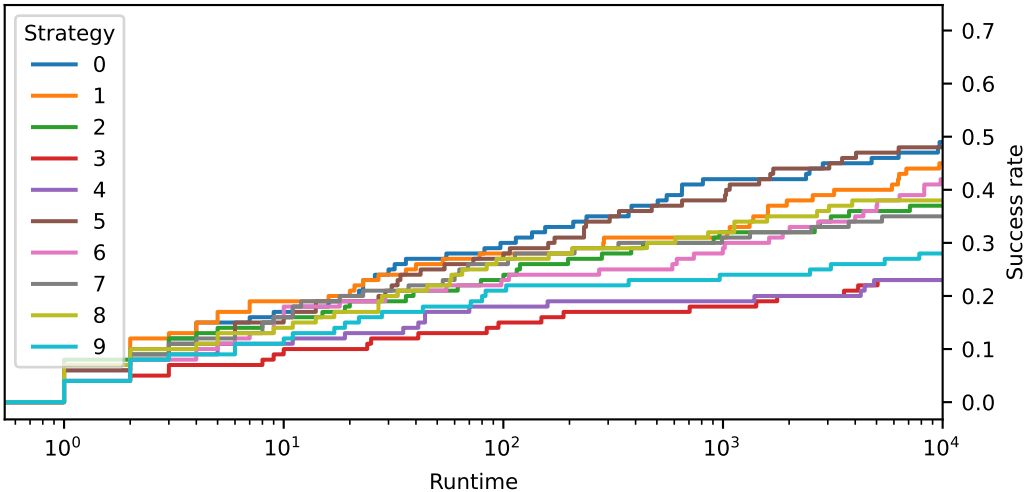
Automated theorem proving and strategies

- Automated theorem provers (Vampire, E, etc.) expose lots of parameters
- Different strategies (configurations) work well on different input problems

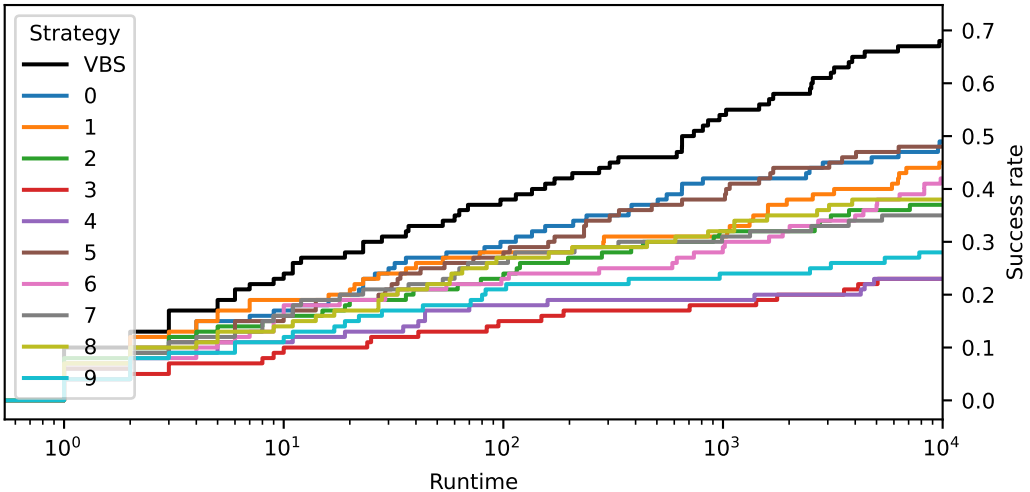
Automated theorem proving and strategies

- Automated theorem provers (Vampire, E, etc.) expose lots of parameters
- Different strategies (configurations) work well on different input problems
- Given a problem and a time limit, how to use the allocated time?

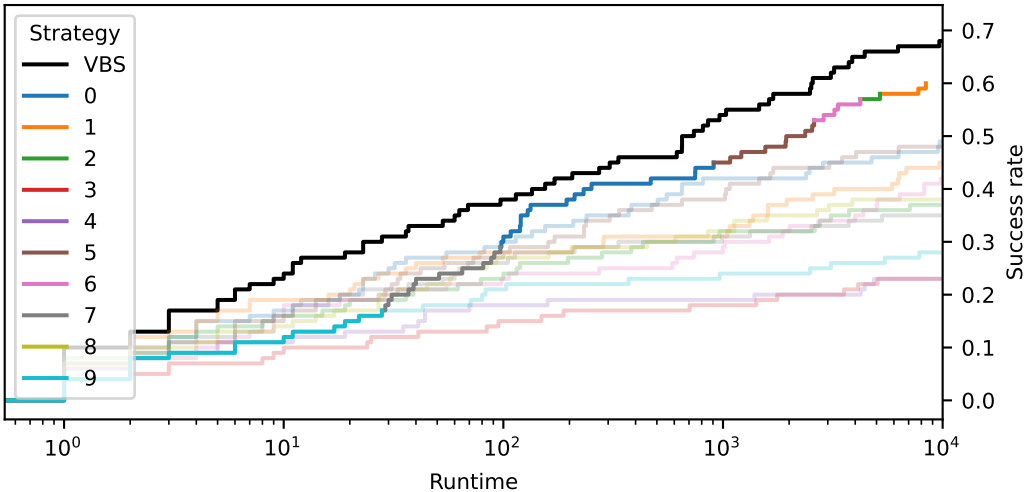
Vampire strategies



Vampire strategies



Vampire strategies



Plan of action

1. Strategy discovery: Collect strong complementary strategies

Plan of action

1. Strategy discovery: Collect strong complementary strategies
2. Schedule construction: Construct a strong strategy schedule

Spider-style strategy discovery

Spider-style strategy discovery

Repeat:

1. Sample Vampire strategy s

Spider-style strategy discovery

Repeat:

1. Sample Vampire strategy s
2. Determine time limit t

Spider-style strategy discovery

Repeat:

1. Sample Vampire strategy s
2. Determine time limit t
3. Sample problem p

Spider-style strategy discovery

Repeat:

1. Sample Vampire strategy s
2. Determine time limit t
3. Sample problem p
4. Attempt to solve p with s in time limit t. If success:

Spider-style strategy discovery

Repeat:

1. Sample Vampire strategy s
2. Determine time limit t
3. Sample problem p
4. Attempt to solve p with s in time limit t. If success:
   - Optimize s on p by local search to reduce runtime

Spider-style strategy discovery

Repeat:

1. Sample Vampire strategy s
2. Determine time limit t
3. Sample problem p
4. Attempt to solve p with s in time limit t. If success:
   - Optimize s on p by local search to reduce runtime
   - Evaluate s' on all problems

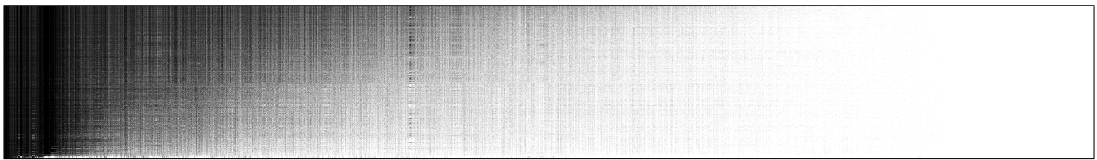
Spider-style strategy discovery

Repeat:

1. Sample Vampire strategy s
2. Determine time limit t
3. Sample problem p
4. Attempt to solve p with s in time limit t. If success:
   - Optimize s on p by local search to reduce runtime
   - Evaluate s' on all problems
   - Store s' and the evaluation results

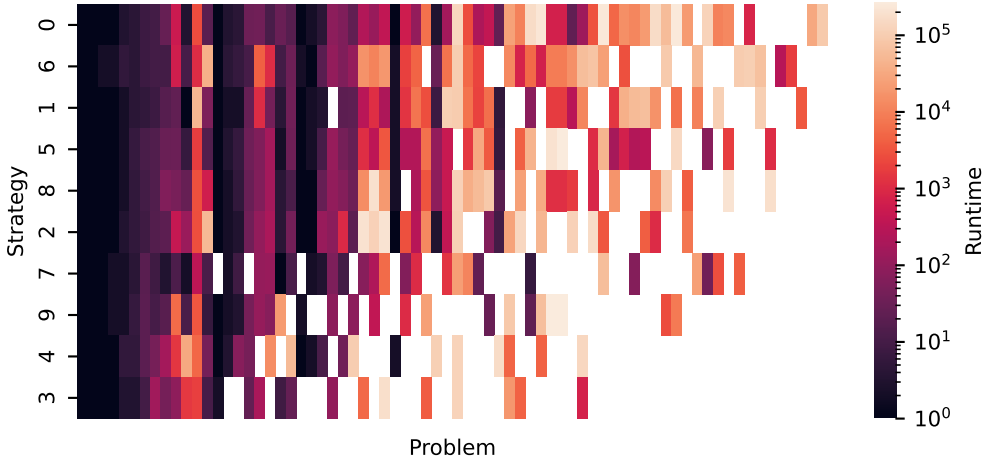
Public dataset: Vampire strategy performance measurements

- 1096 strategies (configurations of Vampire)
- 7866 first-order logic problems from TPTP
- $8\,621\,136 = 1096 \cdot 7866$ solver runs
- Time to compute: 21 days $\times$ 120 CPU cores

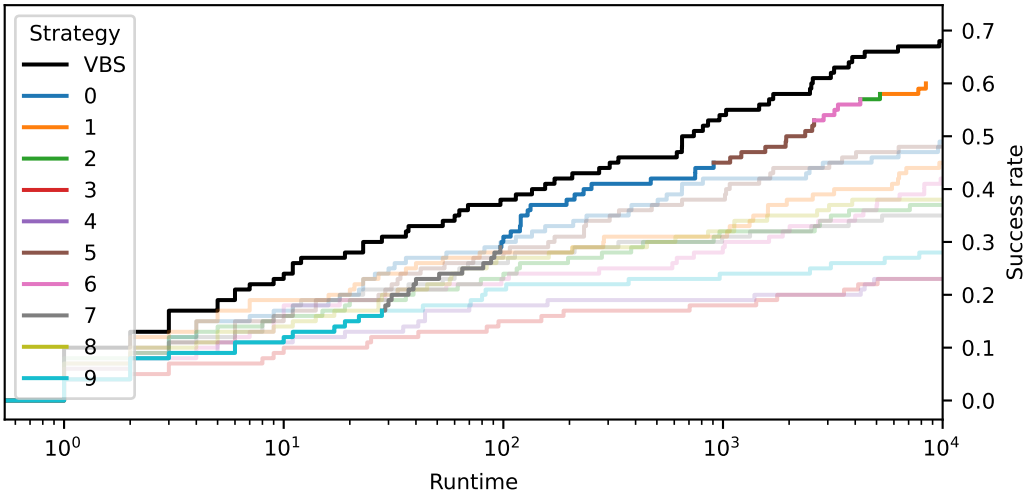


https://zenodo.org/records/10814478

Our dataset (detail)



Our dataset (detail)



Strategy schedule construction

Strategy schedule construction

Input

- Strategies (solvers) S
- Problems P
- Runtime measurements $E : P \times S \to \mathbb{N} \cup \{\infty\}$
- Runtime budget $T \in \mathbb{N}$

Strategy schedule construction

Input

- Strategies (solvers) S
- Problems P
- Runtime measurements $E : P \times S \to \mathbb{N} \cup \{\infty\}$
- Runtime budget $T \in \mathbb{N}$

Output

Schedule $t_s : S \to \mathbb{N}$ such that $\sum_{s \in S} t_s \leq T$

Strategy schedule construction

Input

- Strategies (solvers) S
- Problems P
- Runtime measurements $E : P \times S \to \mathbb{N} \cup \{\infty\}$
- Runtime budget $T \in \mathbb{N}$

Output

Schedule $t_s : S \to \mathbb{N}$ such that $\sum_{s \in S} t_s \leq T$

Maximize

$$\left| \bigcup_{s \in S} \{p \in P \mid E(p, s) \leq t_s\} \right|$$

Generalizing to unseen problems

Generalizing to unseen problems

For a fixed time budget T, repeat:

Generalizing to unseen problems

For a fixed time budget T, repeat:

1. Randomly split P into P_{train} and P_{test} (80:20)

Generalizing to unseen problems

For a fixed time budget T, repeat:

1. Randomly split P into P_{train} and P_{test} (80:20)
2. Construct schedule t_s using runtime measurements on P_{train}

Generalizing to unseen problems

For a fixed time budget T, repeat:

1. Randomly split P into P_{train} and P_{test} (80:20)
2. Construct schedule t_s using runtime measurements on P_{train}
3. Evaluate t_s using P_{test}: How many problems does the schedule solve in runtime T?

Generalizing to unseen problems

For a fixed time budget T, repeat:

1. Randomly split P into P_{train} and P_{test} (80:20)
2. Construct schedule t_s using runtime measurements on P_{train}
3. Evaluate t_s using P_{test}: How many problems does the schedule solve in runtime T?

Average the evaluation results.

Perfect schedule optimization

Perfect schedule optimization

NP-hard

Perfect schedule optimization

NP-hard

Straightforward encoding in integer programming

Perfect schedule optimization

NP-hard

Straightforward encoding in integer programming

Table: Problems covered out of 7866 for $T = 256\,000$ Mi

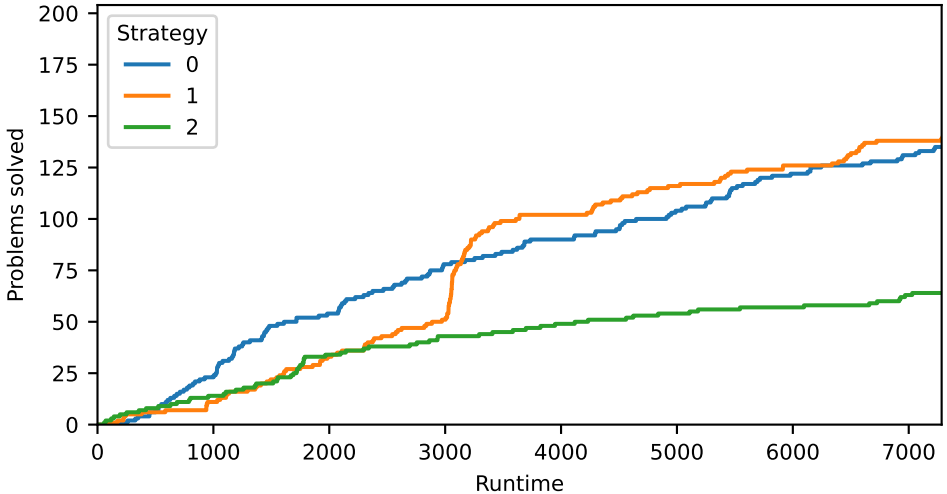
Schedule	Train	Test	Time to construct
Optimal	6536.7	6057.4	> 16 h (Gurobi)
Greedy	6491.7	6048.9	< 1 min

Greedy schedule construction

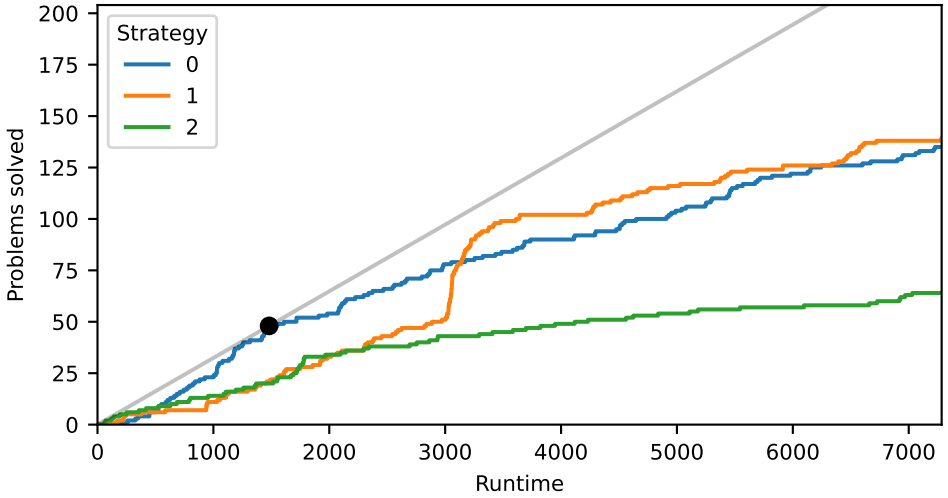
Slice extension criterion

$$s, t \leftarrow \operatorname{argmax}_{s \in S, 0 < t \leq T'} \frac{|\{p \in P' | E(p,s) \leq t_s + t\}|}{t}$$

Greedy schedule construction



Greedy schedule construction



Regularization methods

Regularization method	Parameter	Default
Additive slack	$b \in \mathbb{N}$	$b = 0$
Multiplicative slack	$w \geq 1$	$w = 1$
Diminishing problem rewards	$0 \leq \beta \leq 1$ (discount factor)	$\beta = 0$
Temporal reward adjustment	$0 \leq \alpha$ (reward exponent)	$\alpha = 1$

Slice extension criterion with reward adjustment

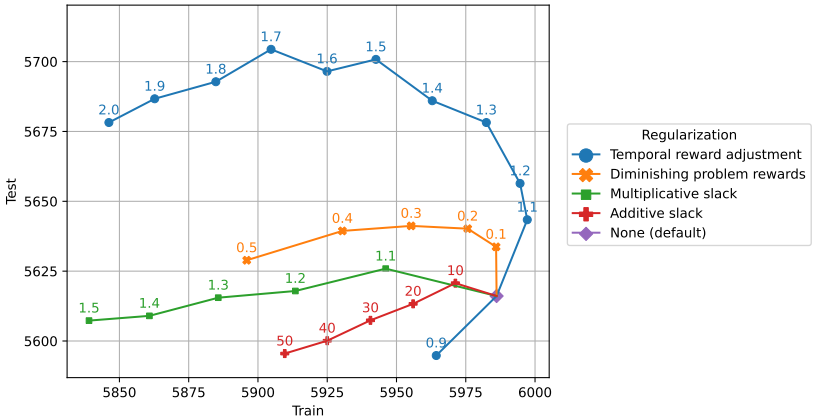
$$s, t \leftarrow \operatorname{argmax}_{s \in S, 0 < t \leq T'} \frac{\left(\sum_{p \in P} [\![t_s < E(p,s) \leq t_s + t]\!] \beta^{\#\text{covered}(p)}\right)^{\alpha}}{t}$$

Schedule post-processing with slack

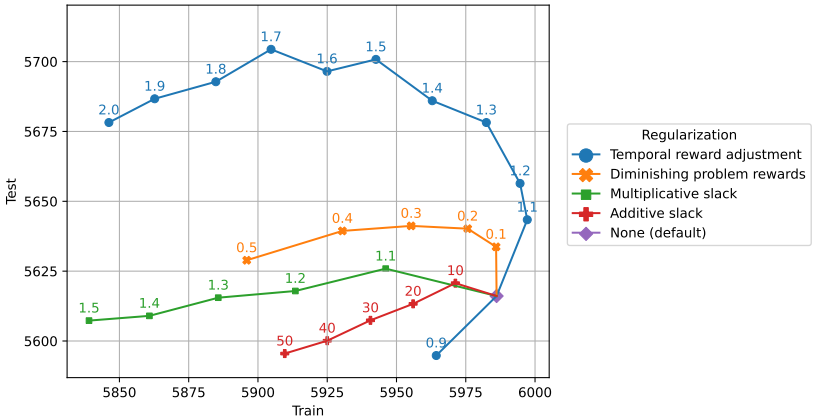
for all $s \in S$ such that $t_s > 0$ **do**

$t_s \leftarrow t_s \cdot w + b$

Regularization for budget 64 000 Mi



Regularization for budget 64 000 Mi



Thank you!

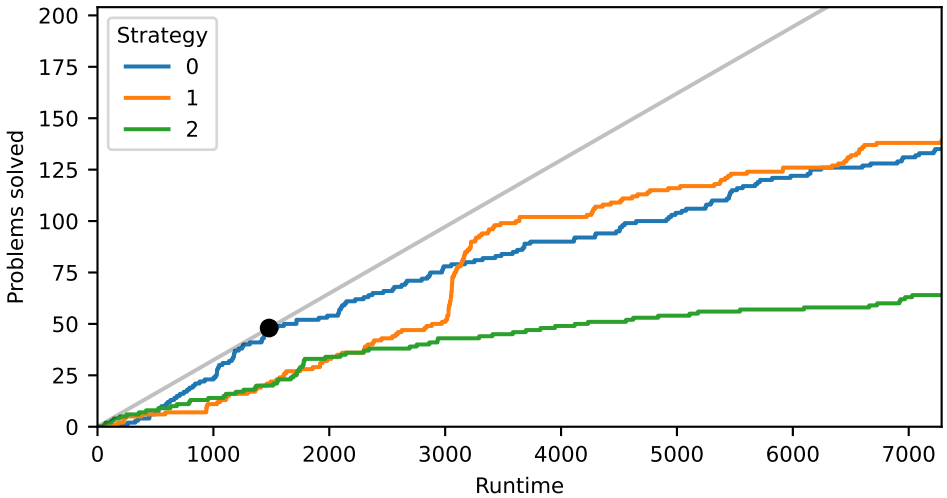
Bonus slides

Empirical results

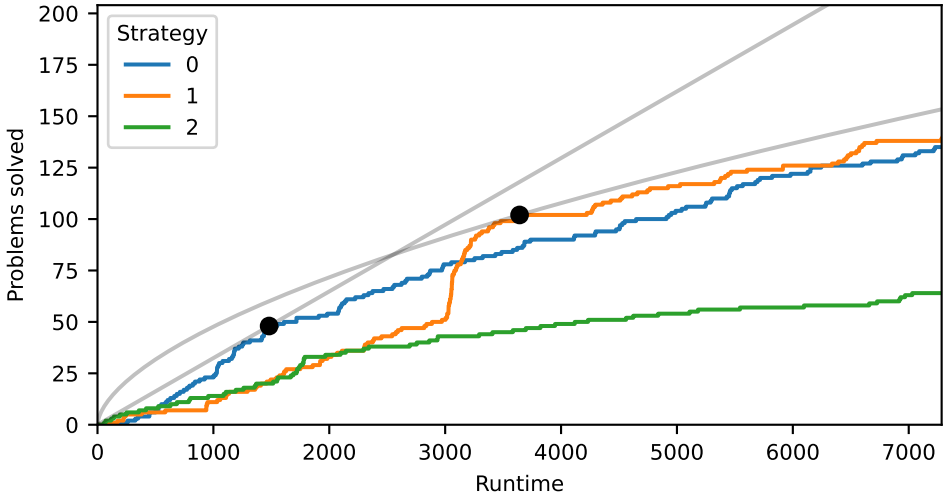
Table: Budget: 64 000 Mi. Performance is the mean number of problems solved out of 7866 across 50 splits. Time to fit is the mean time to construct a schedule in seconds.

Regularization	Performance		Time to fit [s]
	Test	Train	
$\alpha = 1.7$	5704.4	5904.7	4.3
$\beta = 0.3$	5641.2	5955.3	66.5
$w = 1.1$	5625.9	5946.1	19.3
$b = 10$	5620.7	5971.2	19.6
None (default)	5616.1	5986.1	19.9

Temporal reward adjustment ($\alpha = 1.7$)



Temporal reward adjustment ($\alpha = 1.7$)

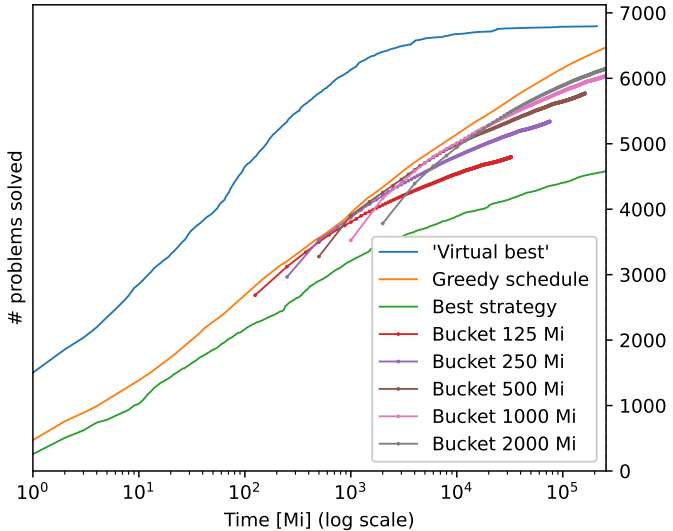


Greedy schedule optimization

Input: Problems P, strategies S, runtimes E, budget T
Output: Schedule $t_s : S \to \mathbb{N}$

$\forall s \in S : t_s \leftarrow 0$ ▷ Start with an empty schedule
$P' \leftarrow P$ ▷ Remaining problems
$T' \leftarrow T$ ▷ Remaining budget
repeat
  $s, t \leftarrow \operatorname{argmax}_{s \in S, 0 < t \leq T'} \frac{|\{p \in P' | E(p,s) \leq t_s + t\}|}{t}$ ▷ Maximize new problems per time
  $t_s \leftarrow t_s + t$ ▷ Extend the schedule
  $P' \leftarrow \{p \in P' | E(p,s) > t_s\}$ ▷ Remove the solved problems
  $T' \leftarrow T' - t$
until $T' = 0$ or $P' = \emptyset$
return t_s

VBS


problems solved
Time [Mi] (log scale)
'Virtual best'
Greedy schedule
Best strategy
Bucket 125 Mi
Bucket 250 Mi
Bucket 500 Mi
Bucket 1000 Mi
Bucket 2000 Mi
0
1000
2000
3000
4000
5000
6000
7000
10^0
10^1
10^2
10^3
10^4
10^5

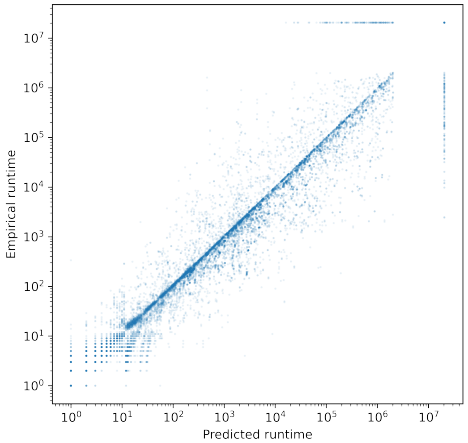
Simulated vs. empirical success

How well does our estimated schedule performance model the actual performance?

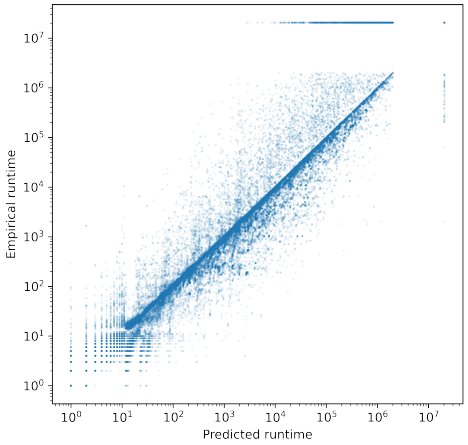
Table: Simulated vs. empirical success (total point evaluations: 23598)

Empirical	Predicted	
	Success	Timeout
Success	18 871	224
Timeout	212	4291

Simulated vs. empirical runtime on the test problems



Simulated vs. empirical runtime on the train problems



Diminishing problem rewards

Input: Problems P, strategies S, runtimes E, budget T, discount factor β $(0 \leq \beta \leq 1)$
Output: Schedule $t_s : S \to \mathbb{N}$

$\forall s \in S : t_s \leftarrow 0$ ▷ Start with an empty schedule
$\forall p \in P : k(p) \leftarrow 0$ ▷ Number of times the problem has been covered
$T' \leftarrow T$ ▷ Remaining budget
repeat
  $s, t \leftarrow \operatorname{argmax}_{s \in S, 0 < t \leq T'} \frac{\sum_{p \in P} [\![t_s < E(p,s) \leq t_s + t]\!] \beta^{k(p)}}{t}$ ▷ Reward per time
  for all $p \in P$ such that $t_s < E(p,s) \leq t_s + t$ **do**
    $k(p) \leftarrow k(p) + 1$
  $t_s \leftarrow t_s + t$
  $T' \leftarrow T' - t$
until no more problems can be covered
return t_s

Slice extension trick

Input: Problems P, strategies S, runtimes E, budget T
Output: Schedule $t_s : S \to \mathbb{N}$

$\forall s \in S : t_s \leftarrow 0$ ▷ Start with an empty schedule
$P' \leftarrow P$ ▷ Remaining problems
$T' \leftarrow T$ ▷ Remaining budget
repeat
  $s, t \leftarrow \operatorname{argmax}_{s \in S, 0 < t \leq T'} \frac{|\{p \in P' | E(p,s) \leq t\}|}{t}$ ▷ Maximize new problems per time
  $t_s \leftarrow t$ ▷ Extend the schedule
  $P' \leftarrow \{p \in P' | E(p,s) \geq t_s\}$ ▷ Remove the solved problems
  $T' \leftarrow T' - t$
until no new problems can be solved
return t_s

Slice extension trick

Input: Problems P, strategies S, runtimes E, budget T
Output: Schedule $t_s : S \to \mathbb{N}$

$\forall s \in S : t_s \leftarrow 0$ ▷ Start with an empty schedule
$P' \leftarrow P$ ▷ Remaining problems
$T' \leftarrow T$ ▷ Remaining budget
repeat
  $s, t \leftarrow \operatorname{argmax}_{s \in S, 0 < t \leq T'} \frac{|\{p \in P' | E(p,s) \leq t_s + t\}|}{t}$ ▷ Maximize new problems per time
  $t_s \leftarrow t_s + t$ ▷ Extend the schedule
  $P' \leftarrow \{p \in P' | E(p,s) \geq t_s\}$ ▷ Remove the solved problems
  $T' \leftarrow T' - t$
until no new problems can be solved
return t_s

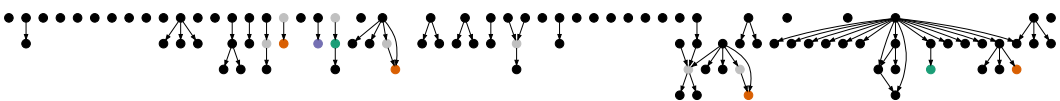
Our dataset: Vampire + TPTP

- Target solver: automatic theorem prover Vampire
  - Strategy space dimension: 103 parameters
- 1096 strategies (configurations of Vampire)
- 7866 first-order logic (FOL) problems from TPTP
  - Solved by some strategy: 6796 in 256 000 Mi, 6405 in 2000 Mi
  - Solved by the best strategy: 4582
  - Solved by the default strategy: 4264
- $8\,621\,136 = 1096 \cdot 7866$ solver runs
  - Time limit: 2000 to 256 000 CPU megainstructions (Mi)
    - Approximately 1 to 128 wallclock seconds

Strategy sampling

103 strategy parameters

- Vampire: 96
  - • Categorical: 89
  - Numeric: 7
    - • Ratio: 4
    - • Floating point uniform: 2
    - • Integer uniform: 1
- • Auxiliary (categorical): 7



Parameter distributions: Biased in favor of strong strategies

Time limit

- Time unit: 10^6 CPU instructions (megainstruction, Mi)
  - 1 second is approximately 2000 Mi on our hardware
- Deterministic sampling: 1000 $\times$ Luby sequence (1, 1, 2, 1, 1, 2, 4, ...)
- Initial time limits: 1000, 1000, 2000, 1000, 1000, 2000, 4000, 1000, 1000, 2000, 1000, 1000, 2000, 4000, 8000, ...

Our dataset: Vampire + TPTP

- Target solver: automatic theorem prover Vampire
- 1096 strategies (configurations of Vampire)
- 7866 FOL problems from TPTP
- $8\,621\,136 = 1096 \cdot 7866$ solver runs
  - Time limit: 2000 to 256 000 CPU megainstructions (Mi)

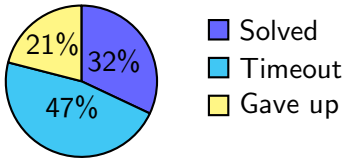
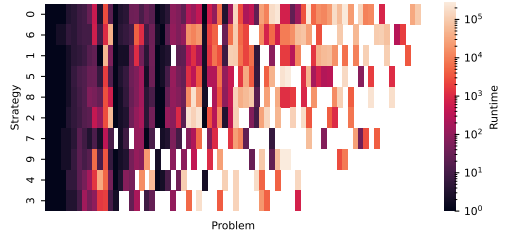
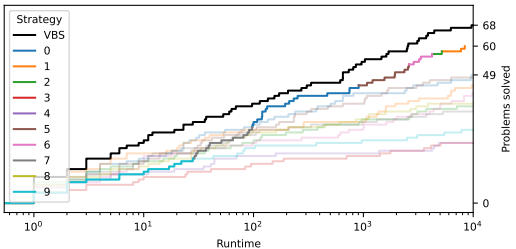


Figure: Distribution of run statuses

Public dataset: `https://zenodo.org/records/10814478`


Strategy
0
6
1
5
8
2
7
9
4
3
Problem
Runtime
10^5
10^4
10^3
10^2
10^1
10^0


Strategy
VBS
0
1
2
3
4
5
6
7
8
9
68
60
49
0
Problems solved
10^0
10^1
10^2
10^3
10^4
Runtime

Perfect schedule optimization

NP-hard

Integer programming

Maximize $\sum_{p \in P} \underbrace{\text{Solved}(p)}$ subject to:

- $\forall p \in P : \underbrace{\text{Solved}(p)} \rightarrow \bigvee_{s \in S} \underbrace{\text{SolvedBy}(p, s)}$
- $\forall p \in P, \forall s \in S : \underbrace{\text{SolvedBy}(p, s)} \rightarrow E(p, s) \leq \underline{t_s}$
- $\sum_{s \in S} \underline{t_s} \leq T$

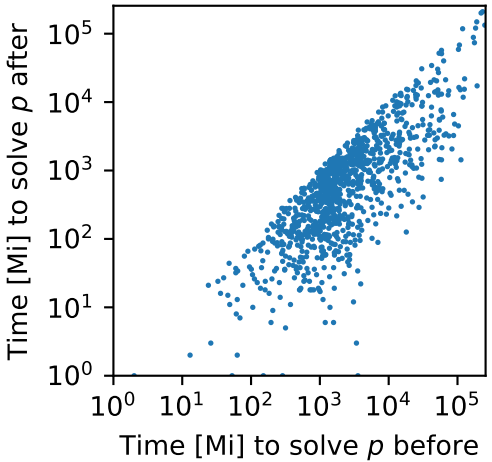
Output schedule: $\underline{t_s} : S \rightarrow \mathbb{N}$

Schedule optimization is NP-hard

Schedule optimization is NP-hard by reduction from maximum coverage problem.
Maximum coverage instance: number k, sets $S_1, \ldots, S_m$
Schedule optimization instance:

- $T \leftarrow k$
- $P \leftarrow \bigcup_{1 \leq i \leq m} S_i$
- $S \leftarrow \{1, \ldots, m\}$
- Strategy i solves problem p in time 1 iff $p \in S_i$.

Strategy optimization by local search



Problem sampling

- Prefers unsolved problems
- Prefers problems with low TPTP rating (easy)