Regularization in Spider-Style Strategy Discovery and Schedule Construction

Filip Bártek (`filip.bartek@cvut.cz`) and Martin Suda

Czech Technical University in Prague, Czech Republic

January 10, 2024

Main task

Target algorithm: Vampire 4.7
Target problem distribution: TPTP FOF theorems and unknowns
Output: Strategy schedule $[(s_1, t_1), \ldots, (s_n, t_n)]$
Challenges:

- Large strategy space (103 parameters)
- Heterogeneous problem space (TPTP)
- Heavy-tailed solving time distribution

Strategy collection

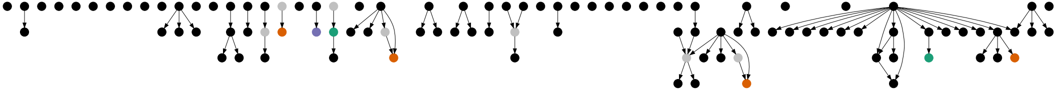
Repeat:

1. Sample 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 solving time
    - Evaluate s on all problems with time limit $2t$
    - Store s and the evaluation results

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, ...

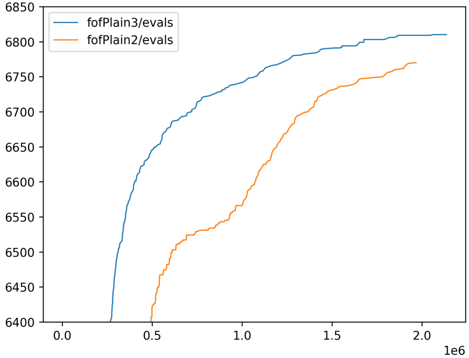
Problem sampling

TPTP FOF theorems and unknown – 7866 problems

- Status
  - Theorem (THM, CAX, UNS): 7711
  - Unknown (UNK, OPN): 154
- With equality: 1276

Distribution: Biased in favor of low TPTP rating

Collected training data



Problems:

- Total: 7866
- Solved by some strategy: 6810
- Solved by the best strategy: 4904

Strategies: 1036

Problem evaluations:

- Total: 8 148 932
- Success: 2 761 157
- Timeout: 4 000 217
- Failure before time limit: 1 387 558

Schedule optimization

Input:

- Strategies S ($|S| = 1036$)
- Problems P ($|P| \approx 7866 \cdot \frac{4}{5} \approx 6293$)
- Solution times $E : S \times P \to \mathbb{N} \cup \{\infty\}$
- Time budget T (in megainstructions; $T \in \{64000, 256000, 2048000\}$)

Output: Schedule $[(s_1, t_1), \ldots, (s_n, t_n)]$

Perfect schedule optimization

NP-hard

Integer programming

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

- $\forall p \in P : \mathrm{Solved}(p) \rightarrow \bigvee_{s \in S} \mathrm{SolvedBy}(p, s)$
- $\forall p \in P, \forall s \in S : \mathrm{SolvedBy}(p, s) \rightarrow \mathrm{StrategyTime}(s) \geq E_p^s$
- $\sum_{s \in S} \mathrm{StrategyTime}(s) \leq T$

Output schedule: $[(s, \mathrm{StrategyTime}(s)) | s \in S]$

Greedy schedule optimization

Input: Problems P, strategies S, solution times E, time budget T
Output: Schedule t_s

$t_s \leftarrow []$
repeat
    $s, t \leftarrow$ strategy s and time t that maximize the number of new problems solved per time:

$$\frac{|\{p \in P | E_p^s < t\}|}{t}$$

    such that $t \leq T - \sum_{(s,t) \in t_s} t$. ▷ Stay within the budget
    $P \leftarrow \{p \in P | E_p^s \geq t\}$ ▷ Remove the solved problems
    $t_s \leftarrow t_s + [(s, t)]$ ▷ Extend the schedule
until No new problems can be solved.
return t_s

Improvement: Slice extension, post-sorting

Covering all problems

Schedule that covers all 6810 "solvable" problems:

- Greedy: 5 305 413 Mi, 594 strategies
- Optimal: 3 049 619 Mi, 240 strategies

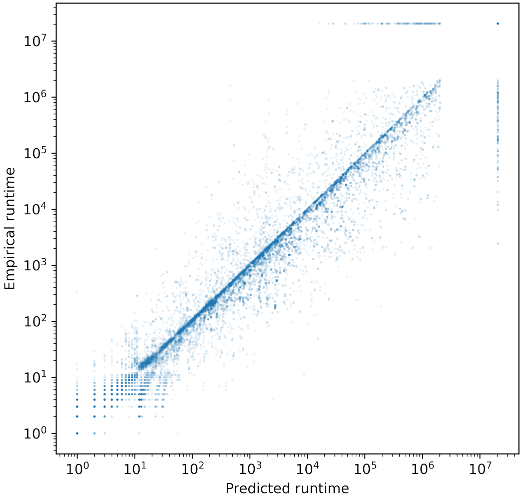
Predicted vs. empirical success

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

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

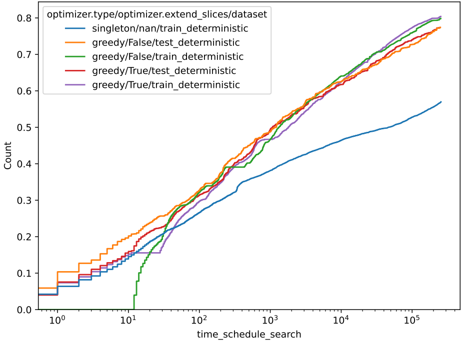
Empirical	Predicted	
	Success	Timeout
Success	18 871	224
Timeout	212	4291

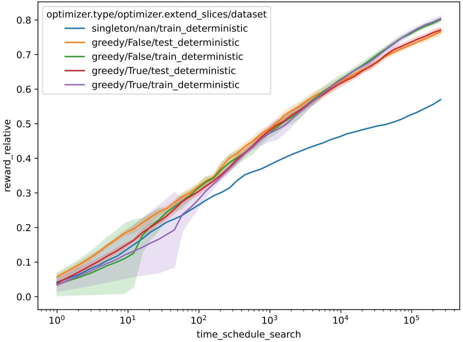
Predicted vs. empirical runtime



Greedy schedule evaluation

Time budget: 256 000 Mi



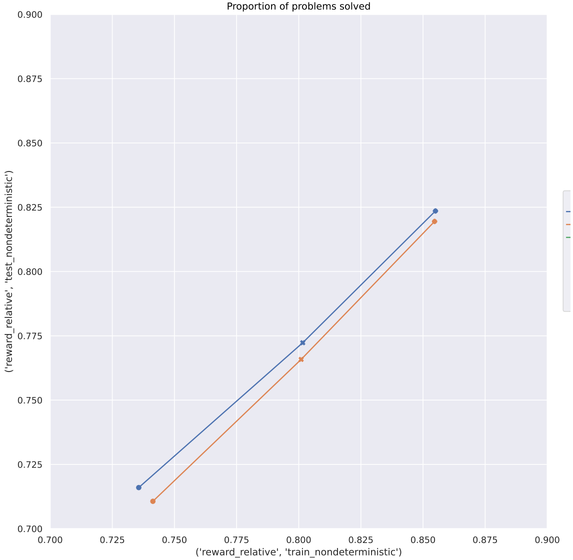


Train vs. test final scores

Table: Normalized problems solved in 256 000 Mi

Case	Dataset	
	Train	Test
Total	7866	7866
Greedy schedule with slice extension	6327	6064
Greedy schedule without slice extension	6301	6027
Best strategy	4481	4481

Train vs. test performance

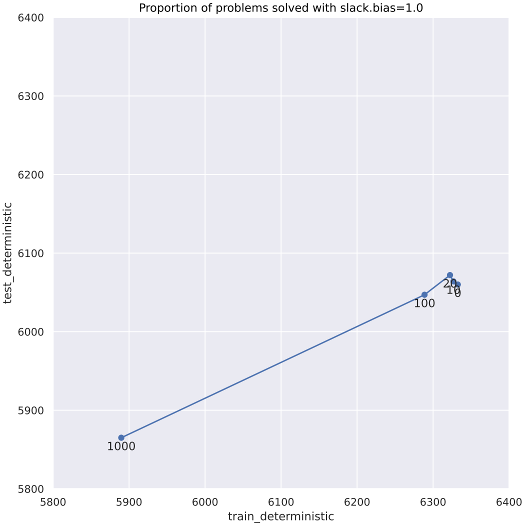


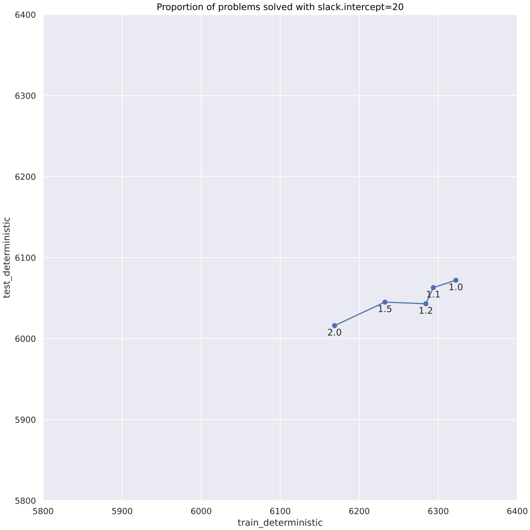
Generalization

How to measure? Evaluate the schedule on a hold-out problem set.
How to improve?

- Inflate the slices by adding slack (additive, multiplicative)
- Score for solving a problem more than once with diminishing returns
- Optimization on a "lucky" problem subset

Generalization by inflation of slices





Strategy distribution extraction

Side task: Improve strategy sampling distribution using the data collected

What makes a good schedule?

- Solves a lot of problems in time limit T
- Anytime performance: Prefer solving problems early (PAR score)

Predicted vs. empirical runtime on the train set