Lightweight Online Learning for Sets of Related Problems in Automated Reasoning






Haoze (Andrew) Wu
Stanford Univ.

Christopher Hahn
Stanford Univ.

Florian Lonsing
Unaffiliated

Makai Mann
MIT Lincoln Lab

Raghuram Ramanujan
Davidson College

Clark Barrett
Stanford Univ.

Motivations

- Automated (logical) reasoning
  - Become mature technology over the past decades
  - Wide applications (e.g., formal verification, theorem proving)
  - Often tackle problems NP-hard and beyond
  - **Scalability** is still a concern

- Machine Learning has been successfully applied in diverse domains
  - Perception
  - Natural language processing
  - …
  - **Automated reasoning?**

ML for AR: status quo

Approach 1: Replace solver as a whole (end-to-end approach)

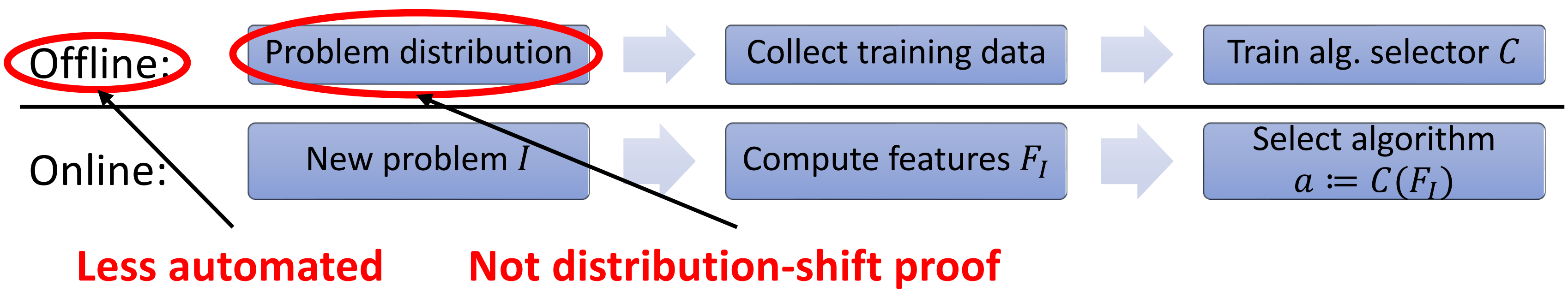
- **Limited scalability**

Approach 2: Replace internal (e.g., branching) heuristics with a ML model

- **Large overhead** compared to hand-crafted heuristics

Approach 3: Meta-algorithmic design

- Currently **most successful**



This work: Self-Driven Strategy Learning (SDSL)

- **Key observation:** many AR tasks involve solving a set $\boldsymbol{S}$ of related problems
  - Bounded Model Checking
  - Iterative abstraction refinement (e.g., CEGAR)
  - Counter-example-guided inductive synthesis (CEGIS)
  - Max-satisfiability
  - …
- Possible to narrow the scope of problem distribution down to this set $\boldsymbol{S}$
- **Key idea: Collect data and learn a model *on-the-fly***

SDSL: a sketch

Given a set of related problems $S \coloneqq \{I_1, I_2, I_3, \dots\}$ and a space of solving strategies $V \coloneqq \{v_0, v_1, v_2, v_3, \dots\}$:

Normal execution:

$\mathbf{Check}(I_1, v_0)$

$\mathbf{Check}(I_2, v_0)$

$\mathbf{Check}(I_3, v_0)$

…

SDSL: a sketch

Given a set of related problems $S \coloneqq \{I_1, I_2, I_3, \dots\}$ and a space of solving strategies $V \coloneqq \{v_0, v_1, v_2, v_3, \dots\}$:

SDSL execution:

$\mathbf{Check}(I_1, v_0)$	$\mathbf{Check}(I_2, v_0)$	$T = \mathbf{fit}(D)$	$\mathbf{Check}(I_3, v_k)$
$\mathbf{Check}(I_1, v_2)$	$\mathbf{Check}(\mathrm{I}_2, v_3)$	$v_k = \arg\min_{v \in V} T(v)$	$\mathbf{Check}(I_4, v_k)$
…	…		…

$I_1, v_0 \to a$	$I_2, v_0 \to e$	$I_3, v_k \to m$
$I_1, v_2 \to b$	$I_2, v_3 \to f$	$I_4, v_k \to n$
…	…	…

Dataset D: $(S, V) \to Cost$

ML model T

Wait, what about the overhead?

Data collection time **amortizes** in many practical applications

SDSL: a calculus view

$$\frac{i < \mathbf{K} \quad \mathbf{check}(f_i, v) = \text{UNSAT}}{i, v \Longrightarrow i+1, v}\ \textbf{(Next)}$$

$$\frac{i = \mathbf{K} \quad \mathbf{check}(f_i, v) = \text{UNSAT}}{i, v \Longrightarrow \text{FAIL}}\ \textbf{(Failure)}$$

$$\frac{\mathbf{check}(f_i, v) = \text{SAT}}{i, v \Longrightarrow \text{SUCCESS}}\ \textbf{(Success)}$$

$$\frac{v_s \in \mathcal{V} \quad j \leq i \quad c = \mathbf{cost}(f_j, v_s)}{i, v, D, T \Longrightarrow i, v, D \cup \{\langle v_s, j, c\rangle\}, T}\ \textbf{(Collect)}$$

$$\frac{T' = \mathbf{fit}(D)}{i, v, D, T \Longrightarrow i, v, D, T'}\ \textbf{(Train)}$$

$$\frac{\mathcal{V}_s \subseteq \mathcal{V} \quad v' = \arg\min_{v_s \in \mathcal{V}_s} T(v_s, i)}{i, v, D, T \Longrightarrow i, v', D, T}\ \textbf{(Strategize)}$$

1) After every **Next**, apply **Collect** *m* times, then apply **Train**
2) Apply **Strategize** whenever i is updated
3) Override 1) when some learning budget is exhausted
4) Terminate whenever **Success** or **Failure** applies

Which strategy to choose? **Which representation?**

SDSL: Collecting informative data

- If $|V|$ is too small, it might not contain good strategy
- If $\mathrm{m} \ll |V|$, how we sample determines the quality of the dataset
  - We need sufficient low-cost strategies in the dataset
  - Explicitly bias towards low-cost strategies with MCMC-sampling (Metropolis-Hastings)

1) Choose a current strategy v
2) Propose to replace v with v', from distribution $\mathrm{q}(v'|v)$
3) If $cost(f, v') \leq cost(f, v)$, accept v' as the current strategy
4) Else, accept v' as the current strategy with probability $\mathrm{a}(v \rightarrow v')$
5) Go to step 2

Case study: Bounded Model Checking

- A widely-used formal verification technique
  - Find bugs
  - Establish formal guarantees

- Check a property P for a transition system over executions bounded by k:

$$bmc(k', k) := I_0 \wedge \bigwedge_{i=0}^{k-1} \rho(i, i+1) \wedge \bigwedge_{i=0}^{k'} P_i \wedge \left(\bigvee_{i=k'+1}^{k} \neg P_i \right)$$

- A Bounded Model Checker solves a set of BMC formulas:

$$\mathcal{F} = \{bmc(k - s, k) \mid k = i \cdot s, 1 \leq i \leq \mathbf{K}\}$$

Step size

BMC Configurations

- **Kissat**: BMC (using Kissat as the underlying SAT solver)

- **Kissat + SDSL**: BMC with self-driven strategy learning
  - Rely on expert knowledge to pick the strategy space

Strategy space I:

Bruno Dutertre, “An Empirical Evaluation of SAT Solvers on Bit-vector Problems”, the SMT workshop, 2020

~~*compacting*:~~	~~compacting internal variables~~
chrono:	support for chronological backtracking
decompose:	elimination of equivalent literals
eagersubsume:	apply subsumption to recently learned clauses
elim:	bounded-variable elimination
elimgates:	recognize clauses that encode and, xor, and if-then-else
~~*lucky*:~~	~~try predefined satisfying assignments~~
probing:	failed-literal probing
rephase:	periodically switch preferred variable polarity
~~*scan-index*:~~	~~optimized watched literal search~~
stabilize:	switch between two heuristic modes
subsumption:	clause subsumption
ternary:	hyper ternary resolution
vivify:	clause vivification
~~*walk*:~~	~~random walks~~

Figure 2: Tested Features in CaDiCaL. Each feature is enabled by default and enables specific CaDiCaL procedures. Except for *scan-index*, all are controlled by command-line options

Strategy space II:

What options might influence kissat's behaviour the most? #25

Closed

arminbiere commented on Jun 16, 2022 Owner ···

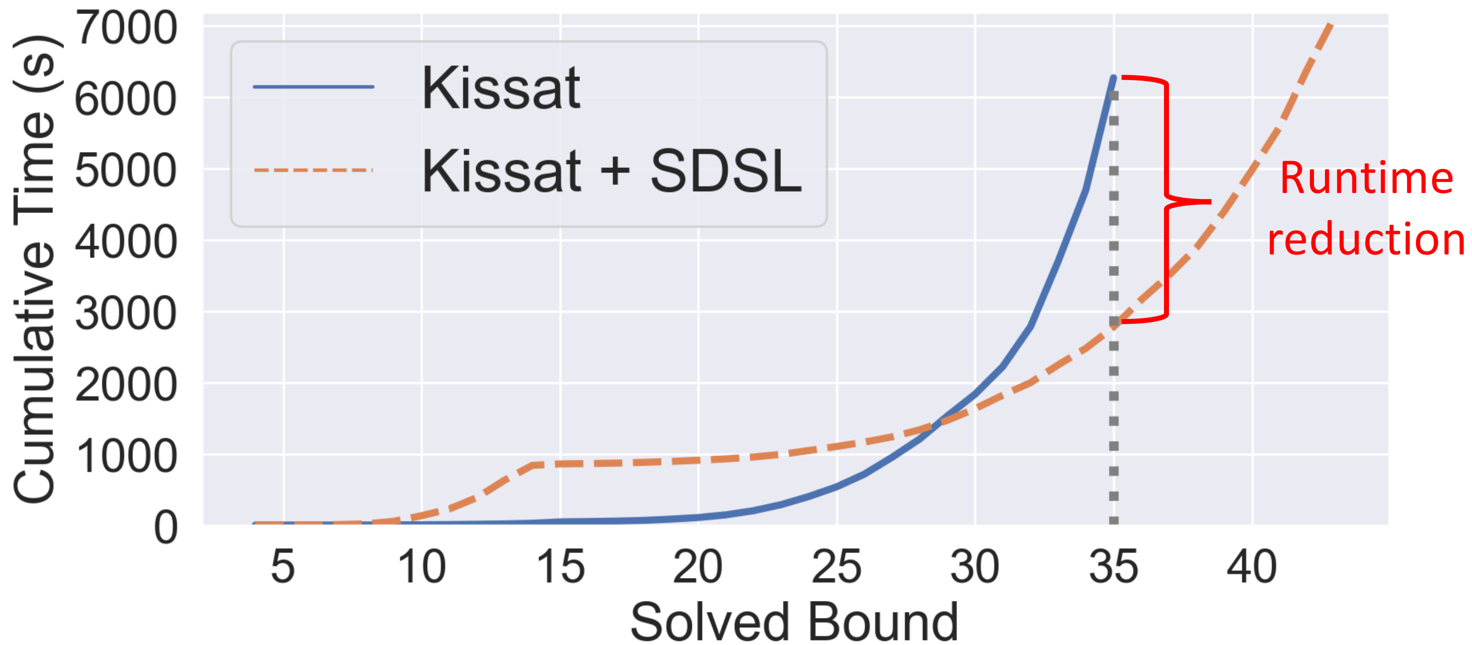
Yes: tier1, chrono, stable, walkinitially, target, phase is a good set. In essence I went over the code and tried to remove actual code and also options. What is left in 'sc2022-light' either had some influence on certain benchmarks or is important for testing and (delta-)debugging (such as all those '--...init=' options). For instance, for hardware miters playing with the '--sweep...' options can give you large benefits (which are problematic for other benchmarks though and not the best you can do for miters anyhow). So this set is pretty close to a robust setting for SAT competition alike benchmarks.

Unrolling the unsolved benchmarks

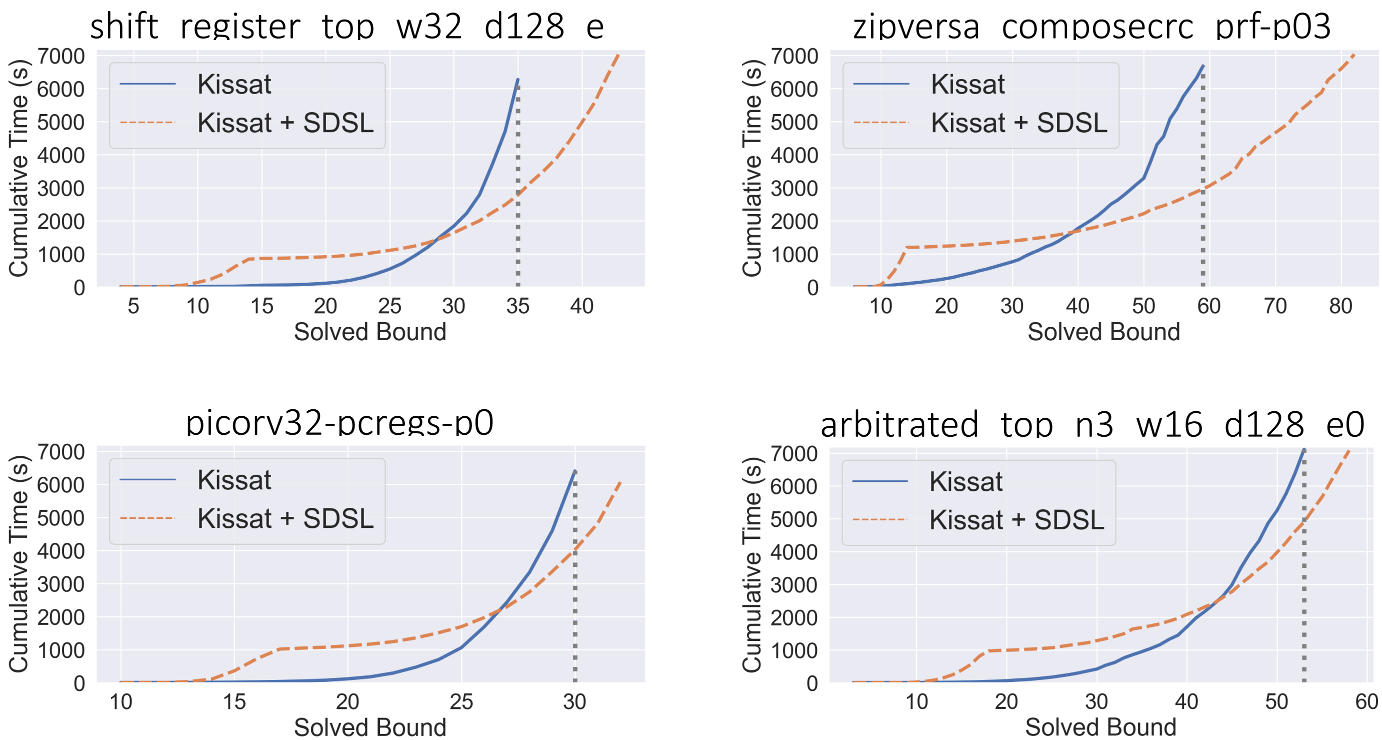
- Unsolved benchmarks from Hardware Model Checking Competition
- Each job is given 2 hours CPU time, one physical core, and 8 GB memory

Unrolling the unsolved benchmarks: example

shift_register_top_w32_d128_e0



Unrolling the unsolved benchmarks: more examples



SDSL certifies larger bounds in > 90% of the cases

Unrolling the unsolved benchmarks

Metrics:

- **Bound**: largest solved bound within 2 hours
- **Time**: time (in seconds) to reach the largest commonly solved bound

	KISSAT + SDSL		KISSAT		PONO
Step size	**Time**	**Bound**	**Time**	**Bound**	**Bound**
1	**4811**	**52.6**	6354	48.7	43.1
10	**2927**	**57.5**	3712	55.4	55.4

Comparison against SoTA Model Checkers

- 89 unknown and satisfiable competition benchmarks
- Additional baselines:
  - **AVR Portfolio**: 16 threads, winner of HWMCC'2020
  - **Pono Portfolio**: 13 threads, winner of HWMCC'2019
- Each thread is given 1 hour CPU time and 8 GB memory

Config.	**Threads**	**Slv.**	**Time**	**Unique**
KISSAT + SDSL	1	**68**	27362	**7**
KISSAT	1	61	6358	0
AVR PORTFOLIO	16	48	12113	2
PONO PORTFOLIO	13	63	10723	0
VIRTUAL BEST	31	72	24700	–

SDSL solves 7 **unsolved** problems during the competition

Summary

- Self-Driven Strategy Learning:
  - A general online Learning methodology potentially applicable to many automated reasoning tasks

- Case study on BMC on hardware designs
  - Consistent improvement over state-of-the-art on satisfiable instances

- **Bigger picture:**
  - **Let ML make high-level decisions, let AR work out the details**

Paper: https://arxiv.org/abs/2305.11087

SCAN ME