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**

![Diagram](image)

Offline:
- Problem distribution
- Collect training data
- Train alg. selector $C$

Online:
- New problem $I$
- Compute features $F_I$
- Select algorithm $a := C(F_I)$

Less automated  Not distribution-shift proof
This work: Self-Driven Strategy Learning (SDSL)

- **Key observation:** many AR tasks involve solving a set $S$ 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 $S$
- **Key idea:** Collect data and learn a model *on-the-fly*
SDSL: a sketch

Given a set of related problems \( S := \{I_1, I_2, I_3, \ldots \} \) and a space of solving strategies \( V := \{v_0, v_1, v_2, v_3, \ldots \} \):

**Normal execution:**

- \( \text{Check}(I_1, v_0) \)
- \( \text{Check}(I_2, v_0) \)
- \( \text{Check}(I_3, v_0) \)
- \( \ldots \)
SDSL: a sketch

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

**SDSL execution:**

\[
\begin{align*}
&\text{Check}(I_1, v_0) \quad \text{Check}(I_2, v_0) \quad T = \text{fit}(D) \quad \text{Check}(I_3, v_k) \\
&\text{Check}(I_1, v_2) \quad \text{Check}(I_2, v_3) \quad v_k = \arg \min_{v \in V} T(v) \quad \text{Check}(I_4, v_k)
\end{align*}
\]

\[
\begin{array}{cccc}
I_1, v_0 \rightarrow a & I_2, v_0 \rightarrow e & I_3, v_k \rightarrow m \\
I_1, v_2 \rightarrow b & I_2, v_3 \rightarrow f & I_4, v_k \rightarrow n \\
\ldots & \ldots & \ldots
dataset D: (S, V) \rightarrow \text{Cost}
\end{array}
\]

Wait, what about the overhead? Data collection time **amortizes** in many practical applications
SDSL: a calculus view

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 $m \ll |V|$, 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 $q(v'|v)$
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 $a(v \rightarrow v')$
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 $k$:

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

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

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

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:


- 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

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

![Graph showing cumulative time vs solved bound for shift_register_top_w32_d128_e0 benchmark. The graph compares Kissat and Kissat + SDSL, highlighting runtime reduction.]
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

<table>
<thead>
<tr>
<th>Step size</th>
<th>KISSAT + SDSL</th>
<th>KISSAT</th>
<th>PONO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Bound</td>
<td>Time</td>
</tr>
<tr>
<td>1</td>
<td>4811</td>
<td>52.6</td>
<td>6354</td>
</tr>
<tr>
<td>10</td>
<td>2927</td>
<td>57.5</td>
<td>3712</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Config.</th>
<th>Threads</th>
<th>Slv.</th>
<th>Time</th>
<th>Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>KISSAT + SDSL</td>
<td>1</td>
<td>68</td>
<td>27362</td>
<td>7</td>
</tr>
<tr>
<td>KISSAT</td>
<td>1</td>
<td>61</td>
<td>6358</td>
<td>0</td>
</tr>
<tr>
<td>AVR PORTFOLIO</td>
<td>16</td>
<td>48</td>
<td>12113</td>
<td>2</td>
</tr>
<tr>
<td>PONO PORTFOLIO</td>
<td>13</td>
<td>63</td>
<td>10723</td>
<td>0</td>
</tr>
<tr>
<td><strong>VIRTUAL BEST</strong></td>
<td>31</td>
<td>72</td>
<td>24700</td>
<td>–</td>
</tr>
</tbody>
</table>

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