Stochastic Local Search for Satisfiability
Modulo Theories
Andreas Fröhlich and Armin Biere
Christoph M. Wintersteiger and Youssef Hamadi
Microsoft Research

Introduction
Satisfiability Modulo Theories (SMT) is essential for many practical applications, e.g., in hard- and software verification, and increasingly also in other scientific areas like computational biology. We present a novel stochastic local search (SLS) algorithm to solve SMT problems, especially those in the theory of bit-vectors, directly on the theory level.

SLS for SAT

SLS for MTS

Theory Information

Industrial Applications

Architecture
- Input formula $F$ as a conjunction of assertions in Negation Normal Form (NNF).
- Given an assignment $a$ to all variables and a constant $c \in [0, 1]$, we define a score function for bit-vector expressions (an extension to bit-vector formulas in NNF is natural):

$$s(t^{(i)} = 0, a) = \begin{cases} 1 & \text{if } t|_{a} = 1 \text{ otherwise} \\ (1 - \text{bin}(-t|_{a})) & \text{if } t|_{a} \leq 1 \text{ otherwise} \\ \end{cases}$$

- Possible moves: Bit-flips, increment, decrement, negation.
- Techniques lifted from SAT: Neighbourhood restriction to pre-selected assertions (similar to WalkSAT), additive weighting scheme for assertions (similar to PMHS), random walks, restarts (similar to lazy).
- Additional techniques: Upper Confidence Bounds (UCB) selection scheme (as used for bands), Variable Neighborhood Search (VNS).

Example
Consider the assertion $a: x + 3 = -x$, where $x$ is a bit-vector of size $n = 6$ (in practice, $n$ is often much larger), $\neg$ represents bitwise negation, and the $+$ operation is as usual, i.e., with overflow semantics. The equation has two solutions: $x = [0, 1, 1, 1, 1, 0]$ and $x = [1, 1, 1, 1, 1, 0]$. If we initialize the search at $x = [0, \ldots, 0]$ and use $c = 1$ for computing the score $s$, the trace of visited states could look as follows:

| $x$ | $s$ | $
eg x$ | $x + 3$ | $\neg(x + 3)$ |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>0%</td>
<td>100000</td>
<td>0%</td>
<td>100000</td>
</tr>
<tr>
<td>111111</td>
<td>100%</td>
<td>000000</td>
<td>0%</td>
<td>100000</td>
</tr>
<tr>
<td>111110</td>
<td>50%</td>
<td>000001</td>
<td>0%</td>
<td>100000</td>
</tr>
<tr>
<td>100000</td>
<td>50%</td>
<td>000001</td>
<td>0%</td>
<td>100000</td>
</tr>
</tbody>
</table>

All states reachable by a single bit-flip from the initial state have a score $s < \frac{1}{2}$.

Experiments
We ran experiments on two different sets of benchmarks. The first benchmark family is the QF_BV benchmark set, which consists of 748 instances that can be found in the SMT-LIB and are also part of the SMT Competition. The second benchmark family is the SAGE2 benchmark set, consisting of 8017 instances. These problems were generated as part of the SAGE project at Microsoft, describing some testcases for automated white-box fuzz testing and are known to be hard for state-of-the-art SMT solvers.

All experiments were run on a Windows HPC cluster of dual-Quad-Xeon (E54xx) machines, 16 GB RAM, and used a time limit of 1200 seconds.

We compared our new solver BV-SLS to the most recent version of Microsoft’s state-of-the-art SMT solver Z3 (which is based on bit-blasting) and also evaluated several SLS solvers for SAT on the propositional encodings of our benchmarks in Conjunctive Normal Form (CNF).

Conclusion
- Novel SLS algorithm directly on the theory level.
- Bridging the gap between SMT and SLS.
- Techniques used for SAT can be successfully lifted to the SMT level.
- Solver BV-SLS outperforms SLS for SAT on the propositional encoding.
- Benefit of using word-level information.
- Insights into the importance of exploiting problem structure also in SAT SLS solvers.
- Still a gap in performance compared to state-of-the-art bit-vector solvers in general, but outperforming Z3 on many industrial instances of practical relevance.
- Interesting possibilities in combining our approach with existing techniques.
- Natural extension to additional theories.

Additional Information
All source code is available at http://z3.codeplex.com as part of the Z3 project. Contact: andreas.froehlich@jku.at, biere@jku.at, c.wintersteiger@microsoft.com, youssef.hamadi@microsoft.com.