Improving Local Search for Bit-Vector Logics in SMT with Path Propagation

Aina Niemetz, Mathias Preiner, Andreas Fröhlich and Armin Biere

Institute for Formal Models and Verification (FMV)
Johannes Kepler University, Linz, Austria
http://fmv.jku.at/

DIFTS 2015
September 26 - 27, 2015
Austin, TX, USA
Introduction

**Bit-Vectors in Sat Modulo Theories (SMT)**

- **State-of-the-art: Bit-Blasting**
  - eager reduction to propositional logic (SAT)
    - offline SAT solver integration
  - efficient in practice
  - relies heavily on **rewriting** and other **simplification** techniques
    - does not scale if input size can not be reduced sufficiently

- **DPLL(T)-based layered approaches** [CAV’07, CAV’14]
  - lazy approach with **online** SAT solver integration

- **Recently:** **Stochastic Local Search (SLS)** for SMT [AAAI’15]
  - implemented in the SMT solver **Z3**
  - lifts SLS to the theory level (**word-level** SLS)
  - without the use of a SAT solver
  - mostly simulates bit-level local search (**focus on single bit flips**)
    - does not fully exploit the advantage of working on the theory level
This work

- reimplementation of [AAAI'15] in our SMT solver Boolector → confirms its effectiveness

- additional strategy: propagation moves
  - do not solely rely on bit flips
  - satisfy lines by propagating assignments from outputs to inputs → implemented in Boolector

- extensive experimental evaluation → suggests improved performance for sequential portfolio of
  - Boolector’s bit-blasting engine with
  - propagation-based SLS techniques
SLS for SMT
A Brief Overview

Preprocessing → $\pi$ → Init inputs → $\alpha(\pi)$ → Compute Score → $s(\pi)$

Rewriting
Simplification

$\phi$ → sat?

$\Rightarrow$ no

restart?

$\Rightarrow$ yes

no

Move

sat?
SLS for SMT
A Brief Overview

\[ \phi \rightarrow \text{Preprocessing} \rightarrow \pi \rightarrow \text{Init inputs} \rightarrow \alpha(\pi) \rightarrow \text{Compute Score} \rightarrow s(\pi) \]

- **sat?**
  - yes: choose
    - optional, with prob=0.1
  - no: restart?
    - yes: restart?
    - no: Best Move
      - default
      - optional, with prob=0.1
      - Random Walk

Rewriting Simplification
Example.

\[ \phi \equiv 21_{[7]} \times \nu_{[7]} = 93_{[7]} \]

0010101 1011101

Candidate: \( \nu_{[7]} := 0000000 \) (initial)

Neighborhood [AAAI'15]

- Single Bit Flips:
  - \( \nu_{[7]} := 0000001 \)
  - \( \nu_{[7]} := 0000010 \)
  - \( \nu_{[7]} := 0000100 \)
  - \( \nu_{[7]} := 0001000 \)
  - \( \nu_{[7]} := 0010000 \)
  - \( \nu_{[7]} := 0100000 \)
  - \( \nu_{[7]} := 1000000 \)

- Increment
  - \( \nu_{[7]} := 0000001 \)

- Decrement
  - \( \nu_{[7]} := 1111111 \)

- Bit-Wise Negation
  - \( \nu_{[7]} := 1111111 \)
Example.

\[ \phi \equiv 21_{[7]} \times v_{[7]} = 93_{[7]} \]

\[
\begin{array}{c}
0010101 \\
1011101
\end{array}
\]

Candidate: \( v_{[7]} := 0000000 \) (initial)

Assignment \( \alpha(\phi) = \{ v_{[7]} := 0000000 \} \) \( \rightarrow \) Score \( s(\phi, \alpha(\phi)) = 0.142857 \)

Find Best Move [AAAI'15]

- Single Bit Flips:
  - \( v_{[7]} := 0000001 \) (0.357143)
  - \( v_{[7]} := 0000010 \) (0.071429)
  - \( v_{[7]} := 0000100 \) (0.357143)
  - \( v_{[7]} := 0001000 \) (0.142857)
  - \( v_{[7]} := 0010000 \) (0.285714)
  - \( v_{[7]} := 0100000 \) (0.071429)
  - \( v_{[7]} := 1000000 \) (0.214286)

- Increment
  - \( v_{[7]} := 0000001 \) (0.357143)

- Decrement
  - \( v_{[7]} := 1111111 \) (0.214286)

- Bit-Wise Negation
  - \( v_{[7]} := 1111111 \) (0.214286)
Example.

\[ \phi \equiv 21_{[7]} \times v_{[7]} = 93_{[7]} \]

\[ \begin{array}{c|c}
0010101 & 1011101 \\
\end{array} \]

Candidate: \( v_{[7]} := 0000000 \) (initial)

Assignment \( \alpha(\phi) = \{ v_{[7]} := 0000000 \} \rightarrow \) Score \( s(\phi, \alpha(\phi)) = 0.142857 \)

**Find Best Move** [AAAI'15]

- Single Bit Flips:
  - \( v_{[7]} := 0000001 \) (0.357143)

\( \rightarrow \) (First) **Most** improvement of score \( s(\phi, \alpha(\phi)) \)

\( \rightarrow \) If no score improving neighbor exists: **randomization**
  (e.g. \( v_{[7]} := 0110011 \))
Example.
\[ \phi \equiv 21_{[7]} \ast v_{[7]} = 93_{[7]} \]

0010101 \quad 1011101

Candidate: \[ v_{[7]} := 0000000 \text{ (initial)} \]

Random Walk [AAAI’15]

- Single Bit Flips:
  - \[ v_{[7]} := 0100000 \]

\[ \rightarrow \text{Picked randomly and immediately} \]
\[ \rightarrow \text{without trying all other neighbors} \]
Example.

\[ \phi \equiv 21_{[7]} \ast v_{[7]} = 93_{[7]} \]

\[
\begin{array}{c}
0010101 \\
1011101
\end{array}
\]

Candidate: \( v_{[7]} := 0000000 \) (initial)

\[ \rightarrow \text{solution } v_{[7]} = 0101001 (41_{[7]}) \]

\[ \rightarrow 4 \text{ moves, 0 restarts} \]
Example.

\[ \phi \equiv 274177_{[65]} \times \nu_{[65]} = 18446744073709551617_{[65]} \]

Candidate: \( \nu_{[65]} := 000000...000000 \) (initial)

**Neighborhood [AAAI’15]**

- **Single Bit Flips:**
  - \( \nu_{[65]} := 000000...000001 \)
  - \( \nu_{[65]} := 000000...000010 \)
  - \( \nu_{[65]} := 000000...000100 \)
  - \( \nu_{[65]} := 000000...001000 \)
  - \( \nu_{[65]} := 000000...010000 \)
  - \( \nu_{[65]} := 000000...100000 \)
  - \( \nu_{[65]} := 000000...000001 \)
  - \( \nu_{[65]} := 000000...000010 \)
  - \( \nu_{[65]} := 000000...000100 \)
  - \( \nu_{[65]} := 000000...001000 \)
  - \( \nu_{[65]} := 000000...010000 \)
  - \( \nu_{[65]} := 000000...100000 \)

- **Increment**
  - \( \nu_{[65]} := 000000...000001 \)

- **Decrement**
  - \( \nu_{[65]} := 111111...111111 \)

- **Bit-Wise Negation**
  - \( \nu_{[65]} := 111111...111111 \)
Example.

\[ \phi \equiv 274177_{[65]} \times \nu_{[65]} = 18446744073709551617_{[65]} \]

Candidate: \( \nu_{[65]} := 000000...00000 \) (initial)

**Assume**: no preprocessing (rewriting, simplification)

\[ \rightarrow 355837 \text{ moves, 21 restarts} \]

\[ \rightarrow \text{unable to determine (single) solution } \nu_{[65]} = 67280421310721_{[65]} \]

- within a time limit of 1200 seconds
- on a 3.4GHz Intel Core i7-2600 machine

\[ \rightarrow \text{solved within one single propagation move} \]
Propagation-Based Move Selection

Example.

\[ \phi \equiv c_1 \land c_2 \land c_3 \]

\(c_1\) initial assignment

Propagation-Based Move Selection

Example.

\[ \phi \equiv c_1 \land c_2 \land c_3 \]

1. initial assignment

2. force \( c_1 = 1 \)
Propagation-Based Move Selection

Example.

\[ \phi \equiv c_1 \land c_2 \land c_3 \]

1. initial assignment
   \[ \{v_7 := 0000000, x_1 := 0, y_1 := 0\} \]
2. force \( c_1 = 1 \)
3. choose path and propagate down
   1. Boolean \( \land \)
      \[ \rightarrow \text{justification-based selection} \]
      \[ \circ 0 \rightarrow 1: \text{choose single controlling input} \]
      \[ \circ \text{else choose randomly} \]

\[ \rightarrow \text{Move: } v_7 := 0101001 \]
Propagation-Based Move Selection

Example.

\[ \phi \equiv c_1 \land c_2 \land c_3 \]

1. initial assignment

2. force \( c_1 = 1 \)

3. choose path and propagate down
   1. Boolean \( \land \)
      \[ \rightarrow \] justification-based selection
      - \( 0 \rightarrow 1 \): choose single controlling input
      - else choose randomly

2. if-then-else (ite)
   - flip condition

\[ \rightarrow \text{Move: } v[7] := 0101001 \]
Propagation-Based Move Selection

Example.

\[ \phi \equiv c_1 \land c_2 \land c_3 \]

1. initial assignment

2. force \( c_1 = 1 \)

3. choose path and propagate down
   - Boolean \( \land \)
     - justification-based selection
       - 0 \( \rightarrow \) 1: choose single controlling input
       - else choose randomly
   - if-then-else (ite)
     - a flip condition
     - b follow enabled branch

\[ \rightarrow \text{Move: } y[7] := 1 \]
Propagation-Based Move Selection

**Down Propagation of Assignments**

- via inverse computation

- Restricted set of bit-vector operations
  - **Unary** operations: \( \text{bvnot, extract} \)
  - **Binary** operations: \( =, \text{bvult, bvshl, bvshr, bvadd, bvand, bvmul, bvdiv, bvurem, concat} \)

- for some operations no well-defined inverse operation exists
  - \( \rightarrow \) produce non-unique values
  - \( \rightarrow \) via randomization of bits or bit-vectors


- assume as fixed: \( b := 1011 \)

- down propagated: \( c := 0001 \)

- selected path, inverse \( a := 0X01 \)

- don’t care
Propagation-Based Move Selection

Down Propagation of Assignments (cntd.)

• if no inverse found
  
  ◦ \( c[n] := a[n] \text{ op } b[n] \)
    
    \( \longrightarrow \) disregard \( b \)
    
    \( \longrightarrow \) choose inverse value for \( a \) that matches assignment of \( c \)
    

    disregarding \( b := 1110 \)

    down propagated: \( c := 0001 \)

    selected path, choose \( a := 0001 \)

  
  ◦ \( c[n] := a[n] \text{ op } bvconst[n] \)
    
    \( \longrightarrow \) assignments of \( b \) and \( c \) are conflicting
    
    \( \longrightarrow \) no value for \( a \) found
    
    \( \longrightarrow \) recover with regular SLS move
Propagation-Based Move Selection

Two scenarios

1. Propagation vs. SLS moves with ratio $n : m$
2. Propagation moves only (ratio $\infty : 0$) $\rightarrow$ default
Experimental Evaluation
Configuration

**Boolector Configurations**

- **Bb** Core engine (bit-blasting approach)
  → winner of the QF_BV main track of SMTCOMP'15

- **Bsls** SLS core engine
  → optionally with random walks enabled (+rw)

- **Bprop** SLS engine with propagation-based strategy enabled
  → **default**: propagation moves only
  → **optional**:
    - ratio $n : m$ of propagation to regular SLS moves (+n:m)
    - conflict recovery via random walks (+frw)

**Z3 Configuration**

- **Z3sls** SLS engine of Z3 version 4.4.0
  → random walks enabled by default
Experimental Evaluation

Configuration

Benchmark Set:

→ all QF_BV benchmarks with status sat and unknown in SMT-LIB (18354 benchmarks) except

   o 449 non-SMT-LIB v2 compliant Sage2 benchmarks and
   o 1469 benchmarks Bb proved to be unsatisfiable within 1200 seconds

→ total: 16436 benchmarks

All experiments on a cluster with 30 nodes of 2.83 GHz Intel Core 2 Quad machines with 8GB RAM running Ubuntu 14.04.2 LTS.
Experimental Evaluation

<table>
<thead>
<tr>
<th></th>
<th>Solved [#]</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bb</td>
<td>9673</td>
<td>80197</td>
</tr>
<tr>
<td>Bsls</td>
<td>7665</td>
<td>91491</td>
</tr>
<tr>
<td>Bsls+rw</td>
<td>7630</td>
<td>91635</td>
</tr>
<tr>
<td>Z3sls</td>
<td>8791</td>
<td>83262</td>
</tr>
</tbody>
</table>

Time limit: 10 seconds, Memory limit: 7GB

• confirms effectiveness and overall gap in performance
• enabling random walks does not improve the performance of Bsls
• performance of Bsls+rw and Z3sls differ on some families

<table>
<thead>
<tr>
<th>Family</th>
<th>Bsls+rw</th>
<th>Z3sls</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Solved [#]</td>
<td>Time [s]</td>
</tr>
<tr>
<td>sage</td>
<td>5275</td>
<td>10626</td>
</tr>
<tr>
<td>Sage2</td>
<td>620</td>
<td>64289</td>
</tr>
<tr>
<td>spear</td>
<td>1187</td>
<td>6336</td>
</tr>
<tr>
<td>uclid</td>
<td>23</td>
<td>2551</td>
</tr>
</tbody>
</table>

→ random seed almost no influence on Bsls (+rw)
→ rewriting and simplification huge impact on performance
**Experimental Evaluation**

<table>
<thead>
<tr>
<th></th>
<th>Solved [#]</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bsls</td>
<td>7665</td>
<td>91491</td>
</tr>
<tr>
<td>Bprop+1:100</td>
<td>7651</td>
<td>91505</td>
</tr>
<tr>
<td>Bprop+1:10</td>
<td>7622</td>
<td>91759</td>
</tr>
<tr>
<td>Bprop+1:1</td>
<td>7623</td>
<td>91690</td>
</tr>
<tr>
<td>Bprop+10:1</td>
<td>7711</td>
<td>90015</td>
</tr>
<tr>
<td>Bprop+100:1</td>
<td>7706</td>
<td>89983</td>
</tr>
<tr>
<td>Bprop</td>
<td>7755</td>
<td>89808</td>
</tr>
</tbody>
</table>

**Time limit:** 10 seconds, **Memory limit:** 7GB

- better performance for higher ratio of propagation moves
- configuration Bprop with most improvement in comparison to Bsls
  -> > 56% solved with propagation moves only
  -> for ~ 85% instances less than 8 recovery SLS moves required
Experimental Evaluation

<table>
<thead>
<tr>
<th></th>
<th>Solved [#]</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bsls</td>
<td>6827</td>
<td>10361</td>
</tr>
<tr>
<td>Bprop</td>
<td>7185</td>
<td>9807</td>
</tr>
<tr>
<td>Bprop+frw</td>
<td>7270</td>
<td>9544</td>
</tr>
</tbody>
</table>

Time limit: 1 second, Memory limit: 7GB

- suggests combination of Bb and Bprop+frw in sequential portfolio
  - run Bprop+frw prior to bit-blasting for 1 second
  - virtual configuration Bb+Bprop+frw
Experimental Evaluation

<table>
<thead>
<tr>
<th></th>
<th>Solved [#]</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bsls</td>
<td>6827</td>
<td>10361</td>
</tr>
<tr>
<td>Bprop</td>
<td>7185</td>
<td>9807</td>
</tr>
<tr>
<td>Bprop+frw</td>
<td>7270</td>
<td>9544</td>
</tr>
</tbody>
</table>

Time limit: 1 second, Memory limit: 7GB

- suggests combination of Bb and Bprop+frw in sequential portfolio
  - run Bprop+frw prior to bit-blasting for 1 second
  - virtual configuration Bb+Bprop+frw

<table>
<thead>
<tr>
<th></th>
<th>Solved [#]</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bb</td>
<td>14806</td>
<td>2623801</td>
</tr>
<tr>
<td>Bb+Bprop+frw</td>
<td>14844</td>
<td>2538620</td>
</tr>
</tbody>
</table>

Time limit: 1200 seconds, Memory limit: 7GB

- +38 instances
- in 97% of the total run time
Experimental Evaluation

Time limit: 1200 seconds, Memory limit: 7GB

→ Bb+Bprop+frw orders of magnitude faster than Bb

- $\geq 1000 \times$ faster (117 instances)
- $100 - 999 \times$ faster (582 instances)
- $10 - 99 \times$ faster (1320 instances)
Experimental Evaluation

<table>
<thead>
<tr>
<th></th>
<th>Solved [#]</th>
<th>Time [s]</th>
<th>Change</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bb</td>
<td>14806</td>
<td>2623801</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bb+Bprop+frw (1s)</td>
<td>14844</td>
<td>2538616</td>
<td>+38</td>
<td>97.1%</td>
</tr>
<tr>
<td>Bb+Bprop+frw (2s)</td>
<td>14852</td>
<td>2535600</td>
<td>+46</td>
<td>96.9%</td>
</tr>
<tr>
<td>Bb+Bprop+frw (3s)</td>
<td>14858</td>
<td>2534900</td>
<td>+52</td>
<td>96.9%</td>
</tr>
<tr>
<td>Bb+Bprop+frw (4s)</td>
<td>14861</td>
<td>2538266</td>
<td>+55</td>
<td>97.1%</td>
</tr>
<tr>
<td>Bb+Bprop+frw (5s)</td>
<td>14862</td>
<td>2544488</td>
<td>+56</td>
<td>97.3%</td>
</tr>
<tr>
<td>Bb+Bprop+frw (6s)</td>
<td>14862</td>
<td>2551784</td>
<td>+56</td>
<td>97.6%</td>
</tr>
<tr>
<td>Bb+Bprop+frw (7s)</td>
<td>14862</td>
<td>2558002</td>
<td>+56</td>
<td>97.9%</td>
</tr>
<tr>
<td>Bb+Bprop+frw (8s)</td>
<td>14862</td>
<td>2565357</td>
<td>+56</td>
<td>98.1%</td>
</tr>
<tr>
<td>Bb+Bprop+frw (9s)</td>
<td>14862</td>
<td>2572600</td>
<td>+56</td>
<td>98.0%</td>
</tr>
</tbody>
</table>

*Time limit: 1200 seconds, Memory limit: 7GB*

Increasing the runtime for Bprop+frw up to 5s increases performance.
**Conclusion**

- **reimplementation** of [AAAI’15] in our SMT solver **Boolector**
  -→ confims its effectiveness

- **propagation-based** extension of [AAAI’15] implemented in **Boolector**
  -→ exploit the advantage of working on the theory level
  -→ SLS recovery tactics not strictly necessary, but current inverse computation introduces **short cuts** for some conflict cases
  -→ elimination of these short cuts left to **future work**

- **Promising:** combination of Boolector’s **bit-blasting** engine with propagation-based approach in sequential portfolio manner
  -→ considerably improves performance on satisfiable instances
  -→ implementation in **Boolector** left to **future work**

