Efficient Sampling of SAT solutions for Testing (ICSE ‘18)

Rafael Dutra, Kevin Laeufer, Jonathan Bachrach and Koushik Sen (EECS Department, UC Berkeley)

Background

- In software testing, generating a lot random solutions to the constraints is a important problem.
    - Conventional symbolic execution and dynamic symbolic execution uses SMT solver to generate ONE solution for the path constraint.
        - Not very scalable due to path explosion
    - Generating multiple solutions to constraint can test multiple paths having the same path prefix

Background: SAT problem

- SAT problem: determining if there exists an assignment (of variables) which satisfies a boolean formula. (First problem proven to be NP-complete)
- How to solve?
  - DPLL algorithms (introduced in 60s, still the basis for modern solvers)
    - CNF form ($(a \lor \neg b) \wedge (\neg a \lor b)$) One solution: [1,1]
    - Backtracking: Assign true/false for one variable, and then solve the sub-problems (branching/splitting).
    - Pruning: Unit propagation/Pure literal elimination
    - Heuristics: Which variable to try first? (e.g. the variable that has the most occurrences)
  - Non-DPLL algorithms
    - Stochastic Local Search (WalkSAT)
      - Pick a random assignment, then try to flip one variable.

Background: Translate SMT problem to SAT problem

- A SMT problem asks to decide if a logic-formula (background theories expressed in first-order logic) can be satisfied.
- Eager approach - encoding and translating (bit-blasting)
  - Example (x != 0) ^ (y | 2 = z)
    - Let x=[b1,b2], y=[b3,b4], z=[b5,b6]
    - b1 ∨ b2,   (b3 = 1) ^ (b4 = b6)
    - CNF form: (b1 ∨ b2) ^ (b3) ^ (b4 ∨ ¬b6) ^ (¬b4 ∨ b6)
- Lazy approach
  - First asks SAT for an assignment and then checks for consistency.
  - Example: (x>0 ∨ y = 100) ^ (x<3 ∨ y = 200)
    - SAT solver assigns [False, False] to (x>0), (x<3). Inconsistency!
    - [True, True], [True, False], [False, True] all are satisfiable assignments.

Problem: how to get multiple assignments quickly and uniformly

- Uniformity:
  - Given the set of all satisfiable assignments R, the solutions should be uniformly sampled from R.
- Benefits of uniformity:
  - Ensure the diversity the inputs, exploring more program states.

Related works (baselines):

- Based on universal hashing (e.g. UniGen)
    - Idea: select a set of universal hashing functions to uniformly partition the solution sparse and then plug the hash function (XOR of boolean variables) to the constraints (e.g. original constraints ^ hash function)
    - Strong uniformity guarantee.
    - Bad performance (for each sampling, needs to make a call to SAT)
- SearchTreeSampler
    - Also uses SAT solver as a black-box
    - Maintain a tree a pseudo-solutions. Level i in the tree stores partial-solutions with the first i boolean variables assigned.
    - Recursively build the tree
        - Sample a pseudo-solutions uniformly from level i, and then call SAT to enumerate all satisfiable pseudo-solutions in level i+1 (e.g. original constraints ^ pseudo-solution of i ^ new probing bits in level i+1)
- Others: treat SAT solver as white boxes

Quick Sampler Algorithm

- Overall idea:
  - Sample a random solution
  - Explore the neighbors (delta-mutations)
  - Combining two mutations
- Intuition:
  - δa and δb consist of a minimal set of bits which can be flipped while still preserving the satisfiability of the formula.
  - So the bits in δa are likely to be closely related to each other by some clauses in the formula.
  - It is likely that those clauses would still be satisfied in σ ⊕ (δa ∨ δb), where we flip all the bits from δa in addition to the bits from δb .

σ :	0 1 0 0 0 1 0 1 1 0 1 1
δ_a :	1 0 0 0 1 1 0 0 0 0 0 0
$\sigma_a = \sigma \oplus \delta_a$:	1 1 0 0 1 0 0 1 1 0 1 1
δ_b :	0 1 0 0 0 1 1 0 1 0 0 0
$\sigma_b = \sigma \oplus \delta_b$:	0 0 0 0 0 0 1 1 0 0 1 1
$(\delta_a \vee \delta_b)$:	1 1 0 0 1 1 1 0 1 0 0 0
$\tilde{\sigma} = \sigma \oplus (\delta_a \vee \delta_b)$:	1 0 0 0 1 0 1 1 0 0 1 1

Figure 1: Combining two mutations.

Implementation

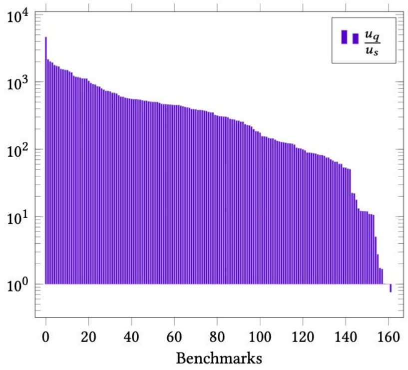
- Use SAT solver as an oracle, to answer MAX-SAT queries
- MAX-SAT query
  - Given a set of hard constraints and a set of soft constraints, can you satisfy all the hard constraints and maximum possible number of soft constraints.
- How to find a random solution?
  - Randomly assign boolean variables.
  - Then call MAX-SAT(hard, soft),
    - where hard is the original constraints,
    - soft is that the **assignment for each variable = randomly assigned one**
- How to find a delta?
  - For one solution, flip one bit of it
  - Then call MAX-SAT(hard, soft)
    - where hard is the original constraints ^ flipped bit must be flipped
    - Soft is that **assignment for each variable = original one**

Evaluation

- Correctness: 75%
- Performance

Benchmark	$\|S\|$	Vars	Clauses	Solutions	QuickSampler n	Calls	Samples	Valid	t_q (μs)	t_q^* (μs)	SearchTreeSampler Samples	t_s/t_q	UniGen2 Samples	t_u/t_q
blasted_case47	28	118	328	262144	244	6616	10010929	0.564	7.5	26	11694350	41.3	3932170	426
blasted_case110	17	287	1263	16384	1387	22208	10001202	0.822	28.3	29	8502350	14.9	245762	34

- Uniqueness



Evaluation - Uniformity

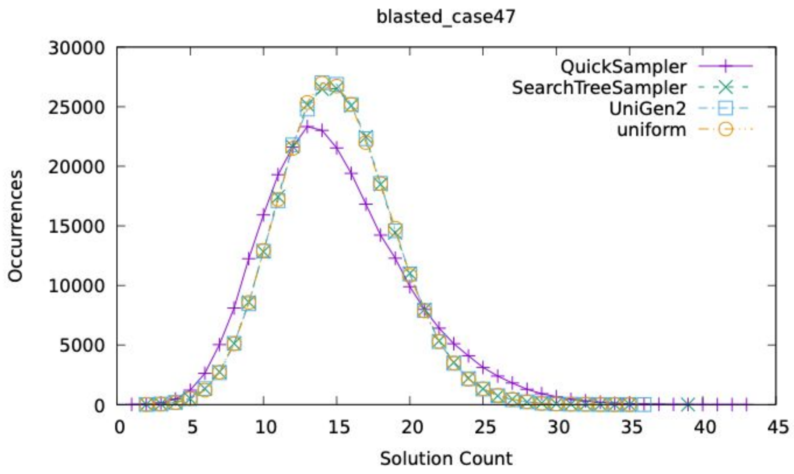


Figure 6: blasted_case47 histogram

More related works: Sampling using SMT solver as an oracle

- **PANGOLIN: Incremental Hybrid Fuzzing with Polyhedral Path Abstraction (S&P 2020)**
- Treat SMT solver as an oracle, determining the path abstraction (range)
- Sampling the range using Dikin walk algorithm

Algorithm 1 Polyhedral path abstraction inference.

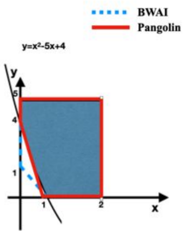
```
procedure INFERENCE(pc ≜ σ_1 ∧ σ_2... ∧ σ_n)
    pc, path constraint. p̂c, polyhedral path abstraction.

    p̂c ← true
    for all input variable v_i in pc do
        min ← SMToptMin(v_i, pc)
        max ← SMToptMax(v_i, pc)
        p̂c ← p̂c ∧ min ≤ v_i ≤ max
    end for
    for all atomic predicate σ_i in pc do
        if σ_i contains linear expression ι_i then
            min ← SMToptMin(ι_i, pc)
            max ← SMToptMax(ι_i, pc)
            p̂c ← p̂c ∧ min ≤ ι_i ≤ max
        end if
    end for

    return φ̂
end procedure
```

$$x \leq 2 \wedge y \leq 5 \wedge x^2 - 5x + 4 \leq y \qquad (1)$$

$$\begin{cases} 0 \leq x \leq 2 \\ 0 \leq y \leq 5 \\ 4 \leq 5x + y \leq 15 \end{cases}$$



How does SMT-opt work?

Symbolic Optimization with SMT Solvers (POPL ‘14)

$$\varphi \equiv 0 \leqslant x \leqslant 3 \wedge 0 \leqslant z \leqslant 2 \wedge (2y \leqslant -x + 4 \vee 4y = 3x + 3),$$

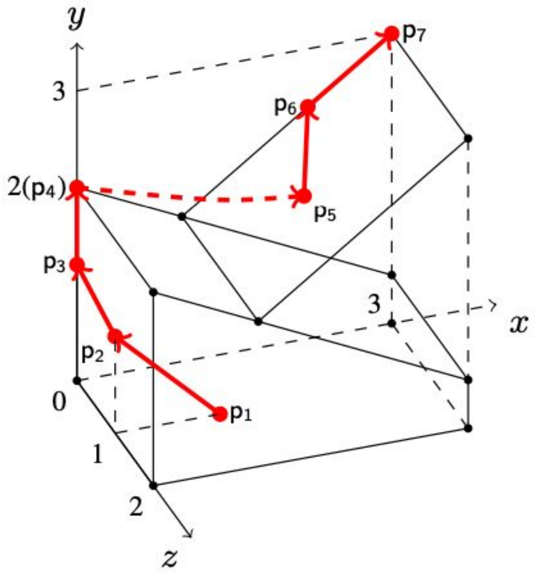


Figure 2. Illustration of SYMBA on a 3-D example.

More related works: leverage extra information to speedup SMT/SAT solving

- Range
  - Pangolin (SP20) - add the range constraints to the formula
  - Trident (ISSTA20)
    - Assigning boolean variables before search
      - E.g if we know, x < C (we reduce 32 bits to logC bits in the vector)
- Variable dependency (add another heuristic)

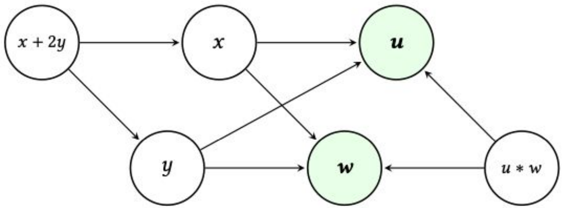


Figure 3: Data-dependence graph for the constraint $\phi \equiv x = u + w \land y = 2 * u - w \land x + 2 * y < 10 \land u * w < 60$.