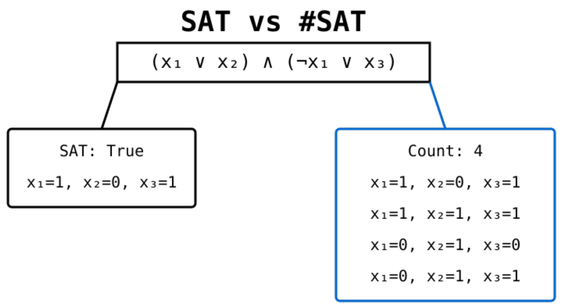
Methods for Evaluating the Similarity of Fuzzers for Model Counters

Cristian Soare (C.Soare@student.tudelft.nl), Dr. Anna L. D. Latour (Supervisor)

Introduction

Model counting (#SAT) is the counting version of the Boolean satisfiability problem (SAT).



Many researchers, working on areas such as quantitative software verification [1], network reliability [2] and reliability of power grids [3], transform their algorithmic problems into #SAT, relying on existing solvers [4]. However, this dependence raises concerns about solver correctness and efficiency due to potential bugs.

- Fuzzing is a method of software validation that works by generating input meant to cause the program to crash.
- This method has been proven effective in detecting such bugs in SAT and #SAT solvers [5].
- Fuzzers for SAT and #SAT solvers are based on random generation of CNF formulas.

$$(P \vee \neg Q \vee R) \wedge (\neg P \vee S) \wedge (Q \vee \neg R \vee \neg S)$$

∧: AND ∨: OR ¬: NOT

This study aims to compare the outputs of different fuzzers to quantify differences in generated problem instance distributions, by looking at the relationship between the features of the CNF input and bug detection patterns.

Research Question

What methods can be employed to evaluate and characterize the similarity between different fuzzers for model counting solvers to ensure comprehensive testing coverage?

Sub-questions:

- **RQ1**: What are the key characteristics and limitations of different methodological approaches for evaluating fuzzer similarity in Model Counting solvers?
- **RQ2**: Which CNF features most effectively characterize fuzzer similarity when using our approach?
- **RQ3**: How effective is our selected method for evaluating fuzzer similarity in Model Counting solvers, and what advantages does it offer compared to alternative approaches?

References

[1] S. Teuber and A. Weigl, "Quantifying Software Reliability via Model-Counting," in *18th International Conference on Quantitative Evaluation of Systems*, A. Abate and A. Marin, Eds., ser. QEST 2021, Springer International Publishing, 2021, pp. 59–79.

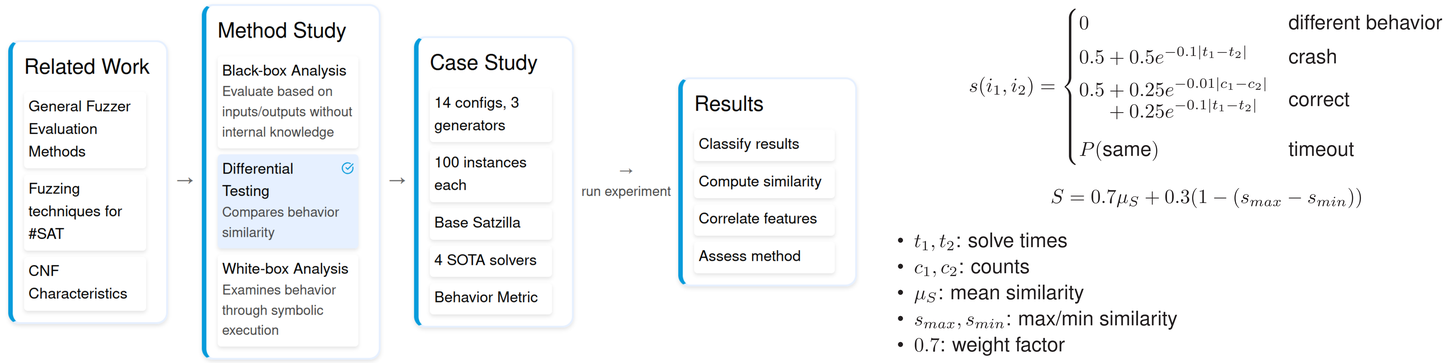
[2] M. Kabir and K. S. Meel, "A Fast and Accurate ASP Counting Based Network Reliability Estimator," in *Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*,, R. Piskac and A. Voronkov, Eds., vol. 94, 2023, pp. 270–287.

[3] A. L. D. Latour, B. Babaki, D. Fokkinga, M. Anastacio, H. H. Hoos, and S. Nijssen, "Exact Stochastic Constraint Optimization with Applications in Network Analysis," *Artificial Intelligence*, vol. 304, p. 103 650, 2022.

[4] C. P. Gomes, A. Sabharwal, and B. Selman, "Model counting," in *Handbook of Satisfiability: Second Edition*, ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. Heule, and H. van Maaren, Eds., IOS Press, 2021, ch. 25.

[5] R. Brummayer, F. Lonsing, and A. Biere, "Automated Testing and Debugging of SAT and QBF Solvers," in *Theory and Applications of Satisfiability Testing – SAT 2010*, O. Strichman and S. Szeider, Eds., Springer Berlin Heidelberg, 2010, pp. 44–57.

Research Design



$$s(i_1, i_2) = \begin{cases} 0 & \text{different behavior} \\ 0.5 + 0.5e^{-0.1|t_1 - t_2|} & \text{crash} \\ 0.5 + 0.25e^{-0.01|c_1 - c_2|} + 0.25e^{-0.1|t_1 - t_2|} & \text{correct} \\ P(\text{same}) & \text{timeout} \end{cases}$$

$$S = 0.7\mu_S + 0.3(1 - (s_{max} - s_{min}))$$

- t_1, t_2: solve times
- c_1, c_2: counts
- μ_S: mean similarity
- s_{max}, s_{min}: max/min similarity
- 0.7: weight factor

Results

Generator	Hardness	Random	Correct	Incorrect	Crash	Timeout
FuzzSAT	easy	0%	397	0	0	3
		50%	400	0	0	0
		100%	400	0	0	0
	hard	0%	157	0	79	164
		50%	400	0	0	0
		100%	400	0	0	0
PairSAT	easy	0%	400	0	0	0
		100%	400	0	0	0
	hard	0%	39	0	55	306
		100%	267	0	71	62
FuzzSATHORN	easy	100%	404	0	0	0
	hard	100%	68	0	3	333
PairSATHORN	easy	100%	404	0	0	0
	hard	100%	98	0	33	273

Table 1: Fuzzing results by generator.

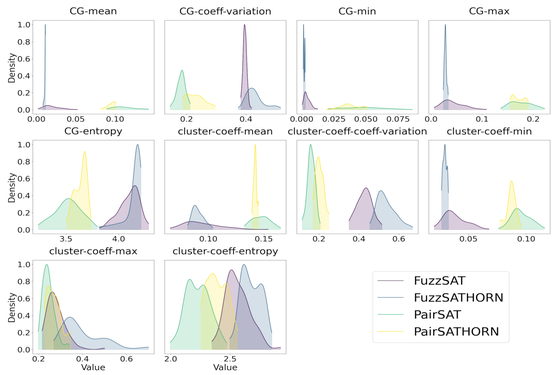


Figure 1: Distribution of Clause Graph Features for hard instances with 100% randomness.

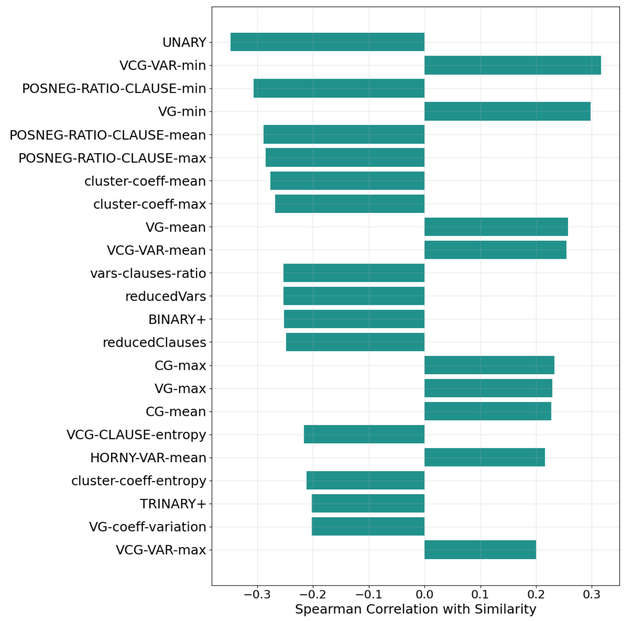


Figure 2: Spearman correlation coefficients between feature distribution similarities and generator behavioral similarity scores. Features are sorted by absolute correlation strength ($|\rho| > 0.3$), showing only statistically significant correlations ($p < 0.05$).

Conclusion

- Fuzzers that generate presumably difficult instances did not consistently cause more timeouts and crashes; increased randomness control often led to easier-to-solve cases.
- Moderate correlations were found between certain CNF features, especially graph structure metrics and balance features, and fuzzer similarity.
- While these correlations offer a good starting point, they don't fully enable the selection of dissimilar fuzzers.
- The lack of instances with incorrect counts and uncertainty in crash analysis indicate a need for better instance generators and methods to distinguish actual solver crashes from external terminations.
- This method offers intriguing insights but requires more extensive experiments and a complementary white-box analysis.