Assessing Reliability of Statistical Maximum Coverage Estimators in Fuzzing

Registered Report - ICSME 2025
Auckland, New Zealand
September 2025

Danushka Liyanage, PhD

Co-authors: Nelum Attanayake, Zijian(Jack) Luo, and Rahul Gopinath, PhD

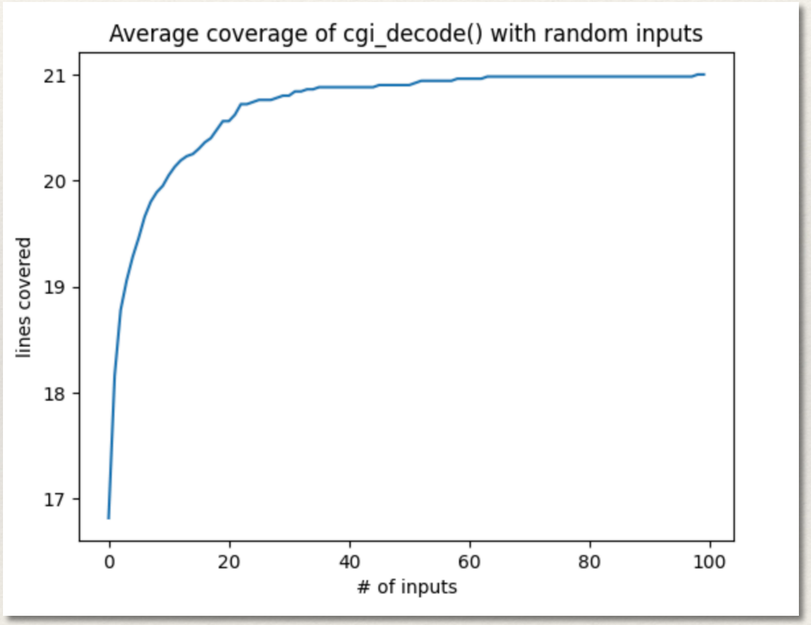
School of Computer Science,
University of Sydney, Australia

Background

> Program testing can be used to show the presence of bugs, but never to show their absence!
>
> Prof.dr. Edsger W. Dijkstra

In testing,

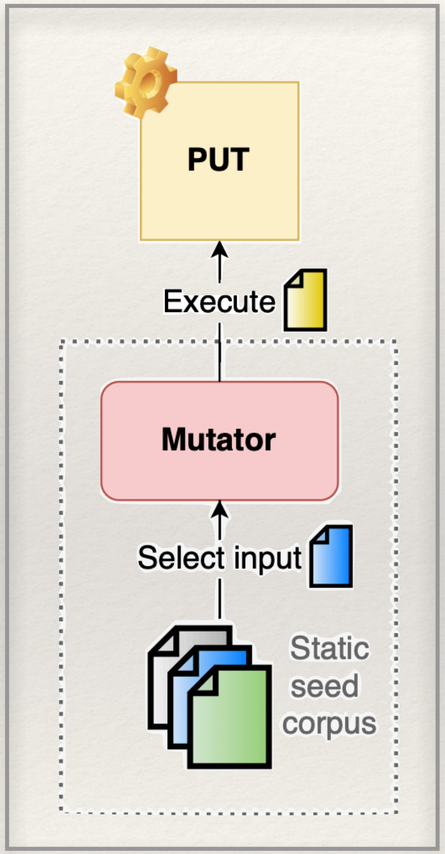
- No matter how long, there could still be bugs in the program
- **Coverage** is a major criteria of testing completeness
- **100% coverage** for most real-world programs is **infeasible**
- **Maximum reachable coverage** is a key concern



Source: *https://www.fuzzingbook.org/html/Coverage.html*

Maximum Reachability in Fuzzing

Fuzzing



Maximum reachability estimation in fuzzing

- Modelled fuzzing as a statistical sampling process (STADS, 2018) [1]
- Proposed a suite of **statistical estimators** for maximum reachable coverage[2,3]
- AKA **species richness estimators** since borrowed from Ecology

Estimator class	Estimator	Notation
Chao-type	Chao2 [15]	$\hat{S}_{\text{Chao2}}$
	Bias corrected Chao2 (Chao2_bc) [16]	$\hat{S}_{\text{Chao2\_bc}}$
	Improved Chao2 (iChao2) [17]	$\hat{S}_{\text{iChao2}}$
Jackknife	First-order Jackknife (JK1) [18]	$\hat{S}_{\text{JK1}}$
	Second-order Jackknife (JK2) [18]	$\hat{S}_{\text{JK2}}$
Incidence-based coverage estimators (ICE)	ICE [19]	$\hat{S}_{\text{ICE}}$
	ICE1 [20]	$\hat{S}_{\text{ICE1}}$
Zelterman estimator	Zelterman [21]	$\hat{S}_{\text{Zelterman}}$
Bootstrap estimator	Bootstrap [22]	$\hat{S}_{\text{Bootstrap}}$
Frequency-based	Chao-Bunge [23]	$\hat{S}_{\text{Chao-Bunge}}$
Unconditional nonparametric maximum likelihood estimator	UNPMLE [24]	$\hat{S}_{\text{UNPMLE}}$
Penalized nonparametric maximum likelihood estimator	PNPMLE [25]	$\hat{S}_{\text{PNPMLE}}$

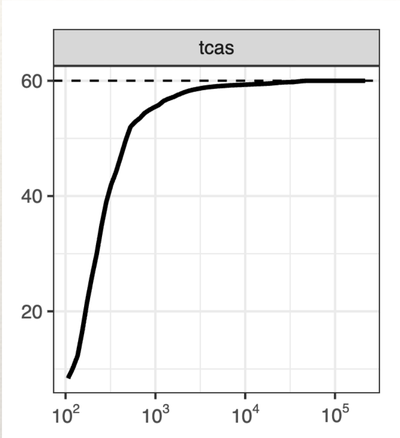
[1] Böhme, M. (2018). STADS: Software testing as species discovery. ACM Transactions on Software Engineering and Methodology (TOSEM), 27(2), 1-52.
[2] Liyanage, D., Böhme, M., Tantithamthavorn, C., & Lipp, S. (2023, May). Reachable coverage: Estimating saturation in fuzzing. In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) (pp. 371-383). IEEE.
[3] Kuznetsov, K., Gambi, A., Dhiddi, S., Hess, J., & Gopinath, R. (2024). Empirical Evaluation of Frequency Based Statistical Models for Estimating Killable Mutants. In 2024 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (pp. 61–71). ACM.

Research Gap

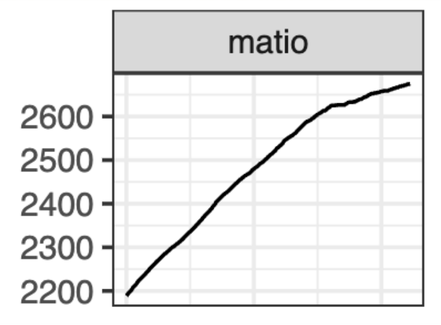
Challenges in Estimation and Evaluation

- **Ground-Truth (*S*)** is **unknown** for real-world programs

	Small-Programs	Real-World Programs
Ground Truth	Attainable in Fuzzing	Unattainable in Fuzzing
Generalizability	No	Yes



Small-Scale



Real-world

- No studies on the **reliability** of effectiveness estimators

[2] Liyanage, D., Böhme, M., Tantithamthavorn, C., & Lipp, S. (2023, May). Reachable coverage: Estimating saturation in fuzzing. In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) (pp. 371-383). IEEE.

Our Approach - Part 01

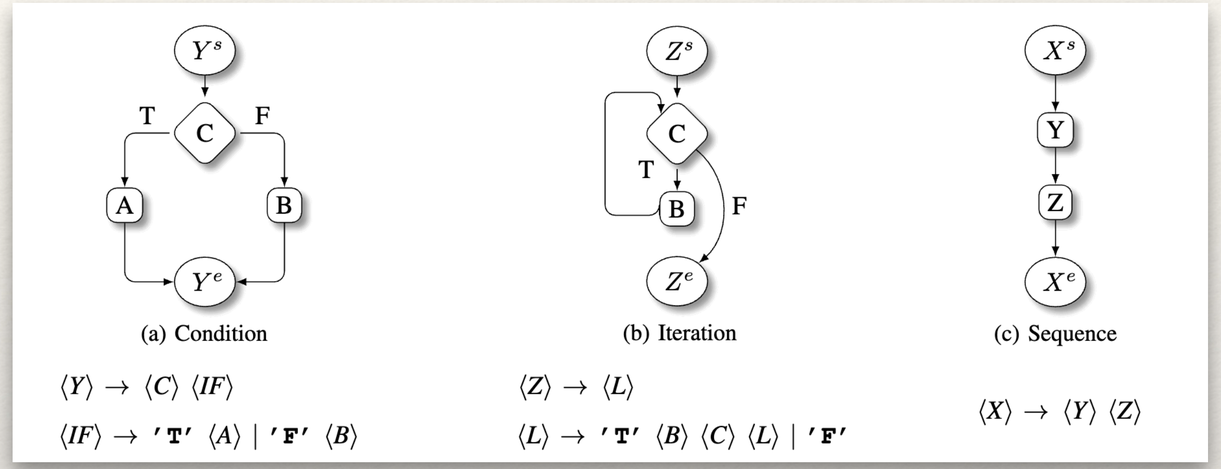
Generate Synthetic Ground-Truth Benchmarks

- Any control-flow can be represented as **context-free LL(1) grammar**

- The **reverse** of this **is also true**

  i.e. Converting LL(1) grammar to recursive descent parser

- These parsers has complexity comparable to real-world programs



Generate Synthetic Benchmarks

$$\langle E \rangle \rightarrow \langle D \rangle \ \langle Es \rangle$$

$$\langle Es \rangle \rightarrow \texttt{'+'} \ \langle D \rangle \ \langle Es \rangle \mid \epsilon$$

$$\langle D \rangle \rightarrow \texttt{'0'} \mid \texttt{'1'} \mid \ldots$$

```
1  def parse_E():
2    parse_D()

4    while lookahead() == '+':
       consume('+')
       parse_D()
     return node

9  def parse_D():
10   token = lookahead()
11   if token and isdigit(token):
12     consume(token)
13   else:
14     raise Error()
```

- Each **nonterminal** becomes a **procedure**
  e.g: ⟨D⟩ ⟶ `def parse_D`
- **Lookahead tokens** (next unprocessed input symbols) guide which rule to apply
- **Linear recursion** can be replaced with a `while` loop
- Each **production rule** is implemented in that **procedure**

Controlling the Program Complexity and Reachability

Variable	Controlled
# of non-terminals	# of procedures
# and length of production rules	Branching complexity
Depth and type of recursion	Direct, indirect, linear recursion
Unreachable nonterminals or dead code	Partially unreachable programs

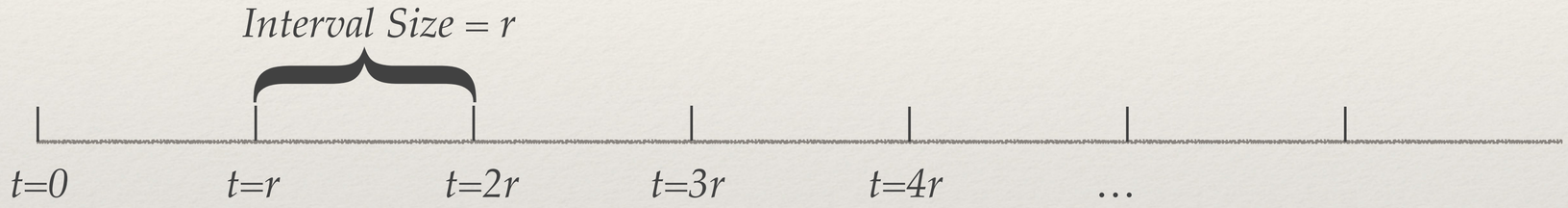
- Grammar has reachability property ⟶ parser procedures are **reachable**
- **Expansion rules** ⟶ **Reachable inputs**

Our Approach - Part 2

Reliability of Effectiveness Estimators

Estimators should not depend on the sampling unit

- **Sampling Unit -** All the inputs generated within a time interval (STADS)



- Previous research only used constant interval size [2] (i.e. 15 mins)

[2] Liyanage, D., Böhme, M., Tantithamthavorn, C., & Lipp, S. (2023, May). Reachable coverage: Estimating saturation in fuzzing. In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) (pp. 371-383). IEEE.

Experimental Setup

	Part 01	Part 02
Fuzzer	**AFL++**	
Subject Programs	**100** Generated Parsers average **100k LoC**	10-20 programs from Fuzzbench
Campaign Length # of Trials	**~100** trials of **24hr** Fuzzing campaigns	**~30** trials of **7-day** Fuzzing campaigns
Initial Seed Corpus	Well-formed valid inputs	Established seeds from Fuzzbench[4]

Evaluation Metrics

- **Mean-bias**

$$mean\ bias(t) = \sum_{i=1}^{K} \frac{\hat{S}_i(t) - S}{KS}$$

- **Variance**
- **Confidence Intervals**
- **Welch's t-test or non-parametric test**

[4] Metzman, J., Szekeres, L., Simon, L., Sprabery, R., & Arya, A. (2021, August). Fuzzbench: an open fuzzer benchmarking platform and service. In Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering (pp. 1393-1403).

Our Paper

Assessing Reliability of Statistical Maximum Coverage Estimators in Fuzzing

Danushka Liyanage*, Nelum Attanayake*, Zijian Luo*, Rahul Gopinath
School of Computer Science, University of Sydney, Australia

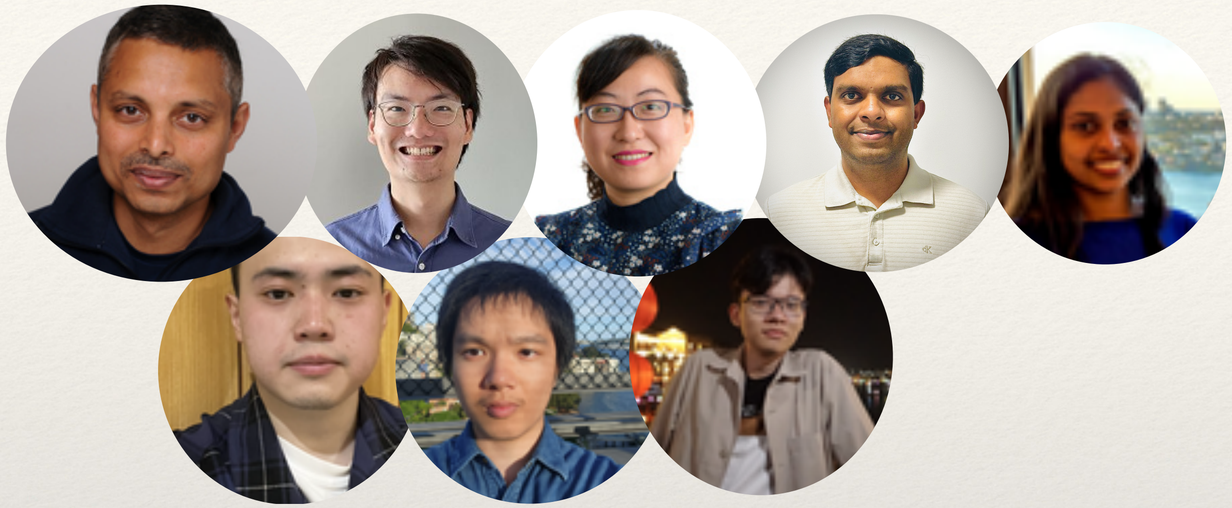
***Abstract*—Background: Fuzzers are often guided by coverage, making the estimation of maximum achievable coverage a key concern in fuzzing. However, achieving 100% coverage is infeasible for most real-world software systems, regardless of effort. While static reachability analysis can provide an upper bound, it is often highly inaccurate. Recently, statistical estimation methods**

for trivial programs. Hence, researchers have resorted to using small programs, fuzzed to saturation, as an alternative [8]. The limitation here is that the performance of estimators in small programs may not generalize to complex real-world software.

This paper suggests an alternative approach. We observe that

Preprint

Software Engineering Team @USyd

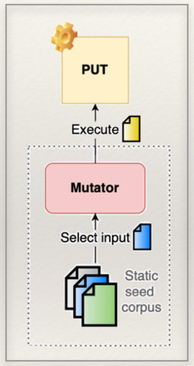


Visit Site 👉

Summary

Maximum Reachability in Fuzzing

Fuzzing



Maximum reachability estimation in fuzzing

- Modelled fuzzing as a statistical sampling process (STADS, 2018) [1]
- Proposed a suite of **statistical estimators** for maximum reachable coverage[2,3]
- AKA **species richness estimators** since borrowed from Ecology
- **Assumptions:** The inputs are ***independently*** generated from ***same distribution (iid)***

Estimator class	Estimator	Notation
Chao-type	Chao2 [15] Bias corrected Chao2 (Chao2_bc) [16] Improved Chao2 (iChao2) [17]	$\hat{S}_{Chao2}$ $\hat{S}_{Chao2\_bc}$ $\hat{S}_{iChao2}$
Jackknife	First-order Jackknife (JK1) [18] Second-order Jackknife (JK2) [18]	$\hat{S}_{JK1}$ $\hat{S}_{JK2}$
Incidence-based coverage estimators (ICE)	ICE [19] ICE1 [20]	$\hat{S}_{ICE}$ $\hat{S}_{ICE1}$
Zelterman estimator	Zelterman [21]	$\hat{S}_{Zelterman}$
Bootstrap estimator	Bootstrap [22]	$\hat{S}_{Bootstrap}$
Frequency-based	Chao-Bunge [23]	$\hat{S}_{Chao-Bunge}$
Unconditional nonparametric maximum likelihood estimator	UNPMLE [24]	$\hat{S}_{UNPMLE}$
Penalized nonparametric maximum likelihood estimator	PNPMLE [25]	$\hat{S}_{PNPMLE}$

Generate Synthetic Benchmarks

$$\langle E \rangle \rightarrow \langle D \rangle \ \langle Es \rangle$$
$$\langle Es \rangle \rightarrow \texttt{'+'} \ \langle D \rangle \ \langle Es \rangle \mid \epsilon$$
$$\langle D \rangle \rightarrow \texttt{'0'} \mid \texttt{'1'} \mid \ldots$$

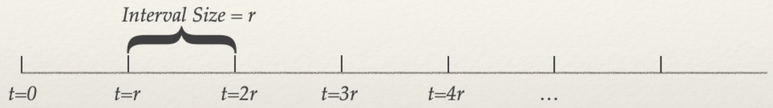
```
def parse_E():
  parse_D()
  while lookahead() == '+':
    consume('+')
    parse_D()
  return node
def parse_D():
  token = lookahead()
  if token and isdigit(token):
    consume(token)
  else:
    raise Error()
```

- Each **nonterminal** becomes a **procedure** e.g: ⟨E⟩ ⟶ `def parse_E`
- **Linear recursion** can be replaced with a `while` loop
- **Lookahead tokens** (next unprocessed input symbols) guide which rule to apply
- Each **production rule** is implemented in that **procedure**

Our Approach - Part 2

Reliability of Effectiveness Estimators

- **Sampling Unit -** All the inputs generated within a time interval (STADS)



Estimators should not depend on the sampling unit (r)

- Previous research only used constant interval size [2] (i.e. 15 mins)

We evaluate estimator reliability by varying sampling unit size r

Experimental Setup

	Part 01	Part 02
Fuzzer	**AFL++**	
Subject Programs	**100** Generated Parsers average **100k LoC**	10-20 programs from Fuzzbench
Campaign Length # of Trials	**~100** trials of **24hr** Fuzzing campaigns	**~30** trials of **7-day** Fuzzing campaigns
Initial Seed Corpus	Well-formed valid inputs	Established seeds from Fuzzbench[4]

Evaluation Metrics

- **Mean-bias**

$$mean\ bias(t) = \sum_{i=1}^{K} \frac{\hat{S}_i(t) - S}{KS}$$

- **Variance**
- **Confidence Intervals**
- **Welch's t-test or non-parametric test**