ACSAC 2022
December 5-9, 2022 • Austin, Texas, USA

# Alphuzz: Monte Carlo Search on Seed-Mutation Tree for Coverage-Guided Fuzzing

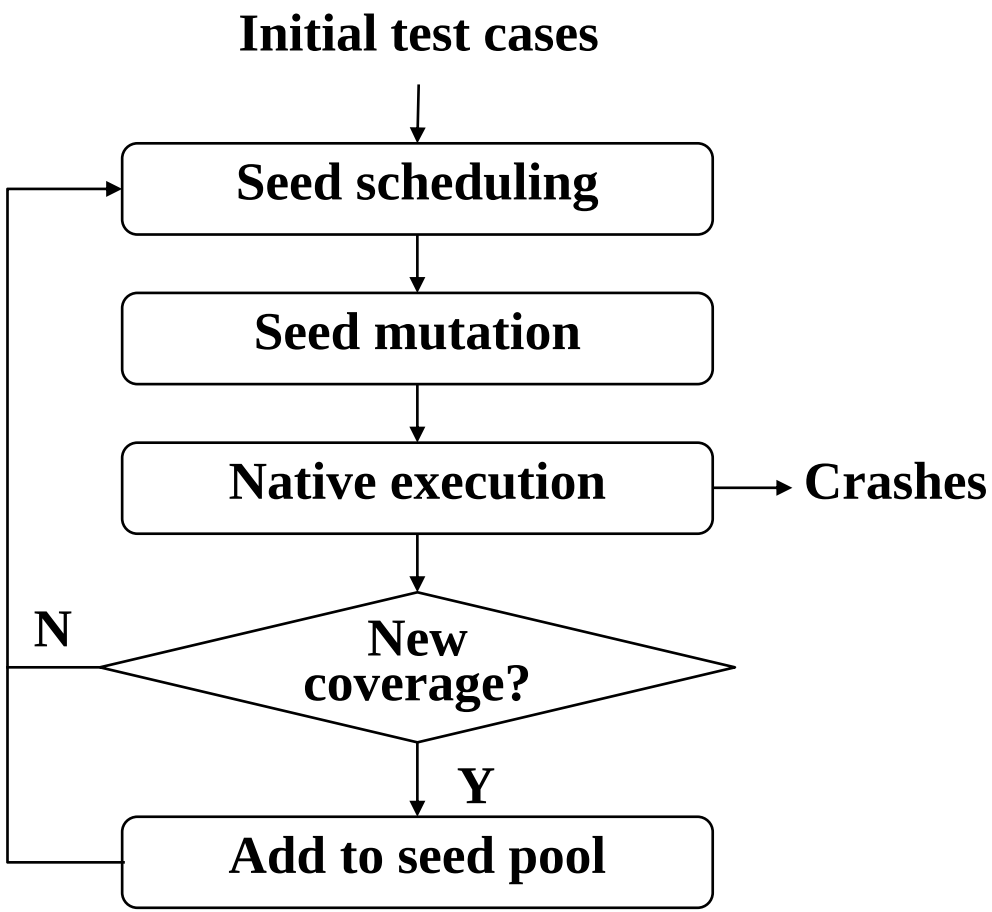
**Yiru Zhao**[1], Xiaoke Wang[1], Lei Zhao[1], Yueqiang Cheng[2], and Heng Yin[3]

[1] Key Laboratory of Aerospace Information Security and Trusted Computing,
Ministry of Education, School of Cyber Science and Engineering, Wuhan University, China
[2] Nio Security, Mountain View, CA, USA
[3] University of California, Riverside, Riverside, CA, USA
Corresponding author: Lei Zhao, email: leizhao@whu.edu.cn



# Background

## Coverage-based greybox fuzzing(CGF)

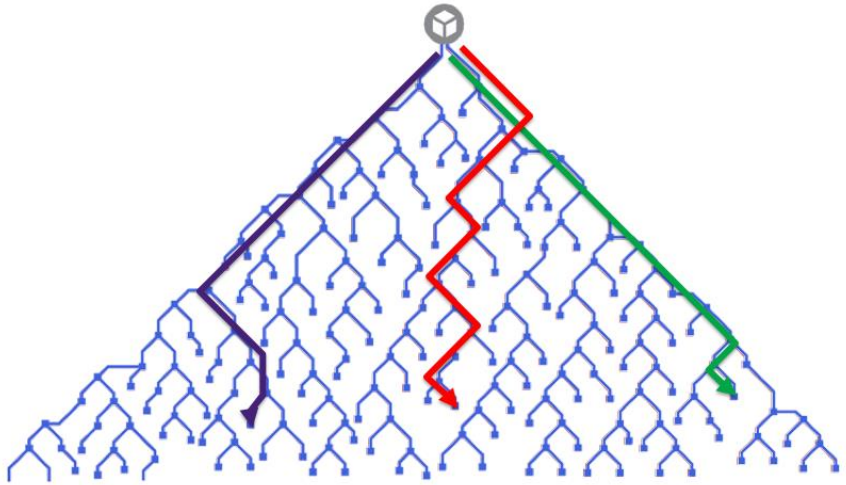



## Research question

**Which seed should be selected ?**

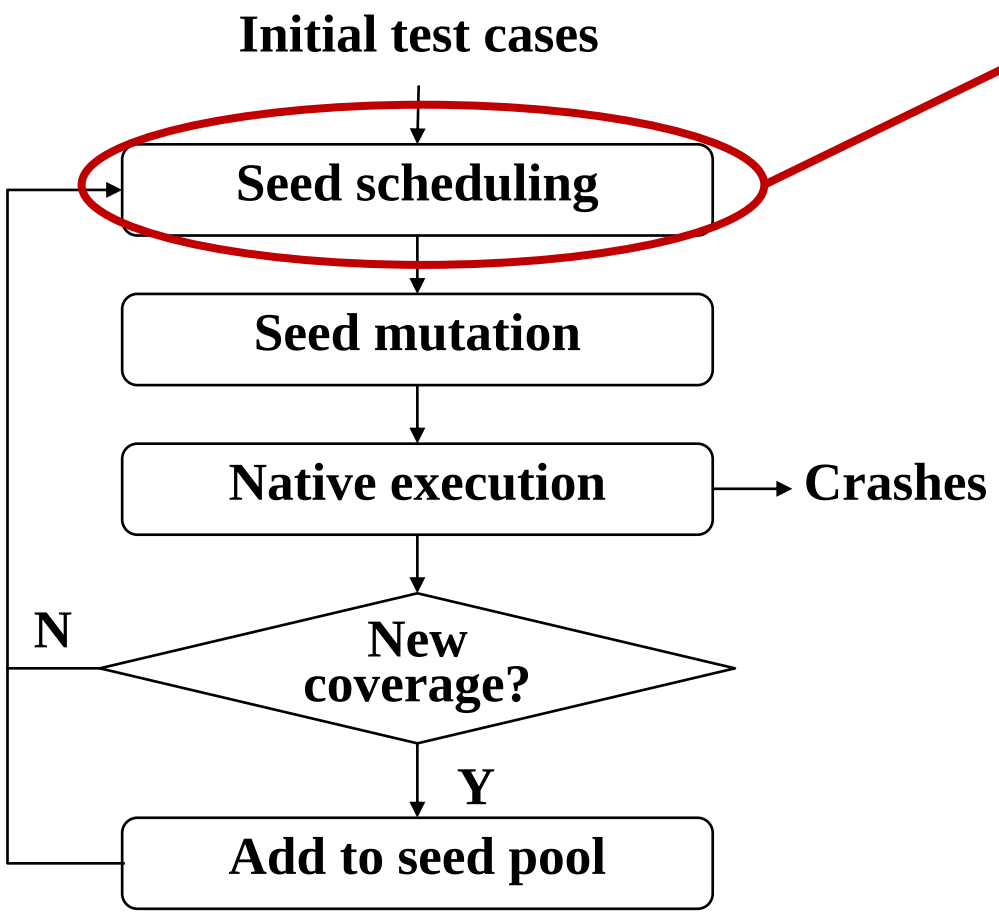
Limited fuzzing effort v.s. Growing number of seeds

Exploration v.s. Exploitation



# Background

## Coverage-based greybox fuzzing(CGF)

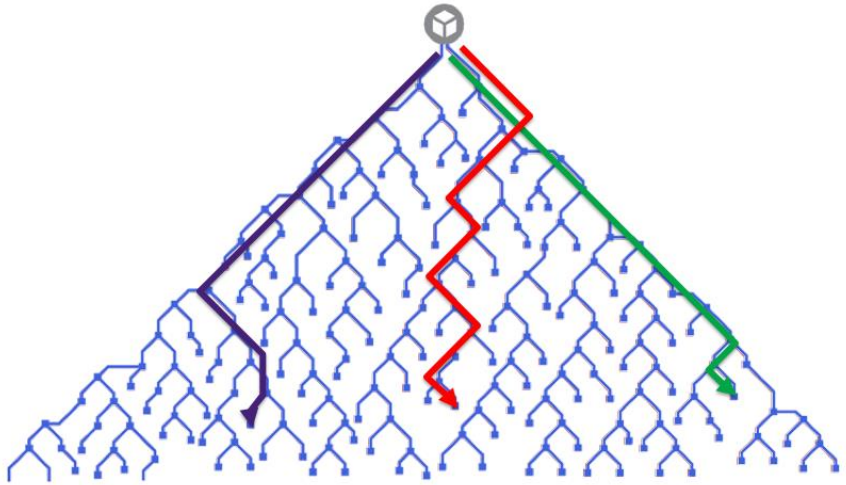



## Research question

**Which seed should be selected ?**

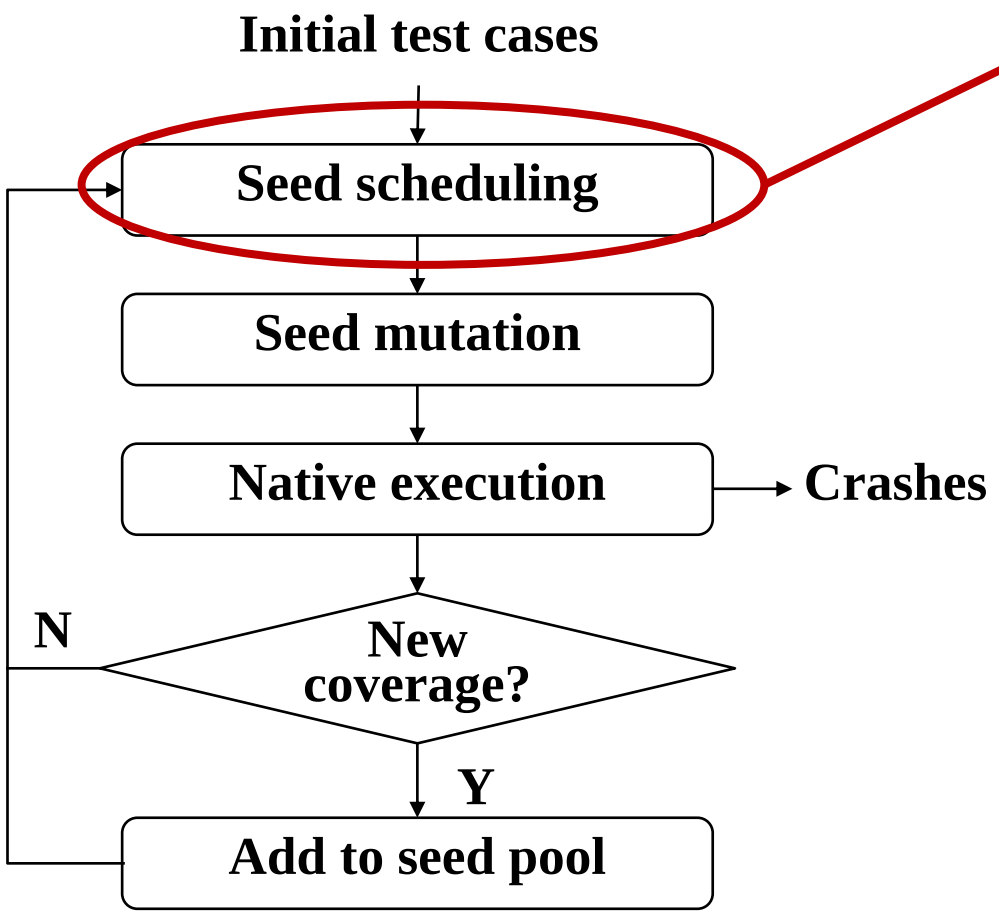
Limited fuzzing effort v.s. Growing number of seeds

Exploration v.s. Exploitation



# Background

## Coverage-based greybox fuzzing(CGF)

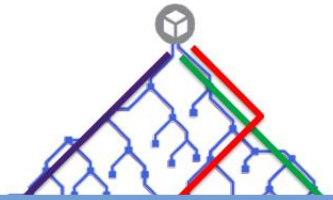



## Research question

**Which seed should be selected ?**

Limited fuzzing effort v.s. Growing number of seeds

Exploration v.s. Exploitation

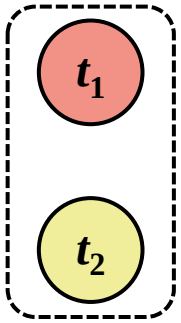


**Seed scheduling refers to selecting an execution path for exploring its neighbor paths**

# Motivation

## Motivating example

| | code | test cases | | | | |
|---|---|---|---|---|---|---|
| | | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ |
| | | 'x' | 'y' | 'xaf' | 'xc' | 'ym' |
| | int func1 (input) { | | | | | |
| $b_0$ | if (input[0] == 'x') | ● | ● | ● | ● | ● |
| $b_1$ | func2(input); | ● | | ● | ● | |
| $b_2$ | else if (input[0] == 'y') | | ● | | | ● |
| $b_3$ | func3(input); | | ● | | | ● |
| | } | | | | | |
| | int func2 (input) { | | | | | |
| $b_4$ | if (input[1] == 'a') | ● | | ● | ● | |
| $b_5$ | if (input[2] == 'f') | | | ● | | |
| $b_6$ | … | | | ● | | |
| $b_7$ | else if (input[1] == 'b') | ● | | | | |
| $b_8$ | … | | | | | |
| $b_9$ | else if (input[1] == 'c') | ● | | | ● | |
| $b_{10}$ | … | | | | ● | |
| | } | | | | | |
| | int func3 (input) { | | | | | |
| $b_{11}$ | if (input[1] == 'm') | | ● | | | ● |
| $b_{12}$ | if (input[2] == 'n') | | | | | ● |
| $b_{13}$ | func4 (input); | | | | | |
| $b_{14}$ | else exit (0); | | ● | | | |
| | } | | | | | |

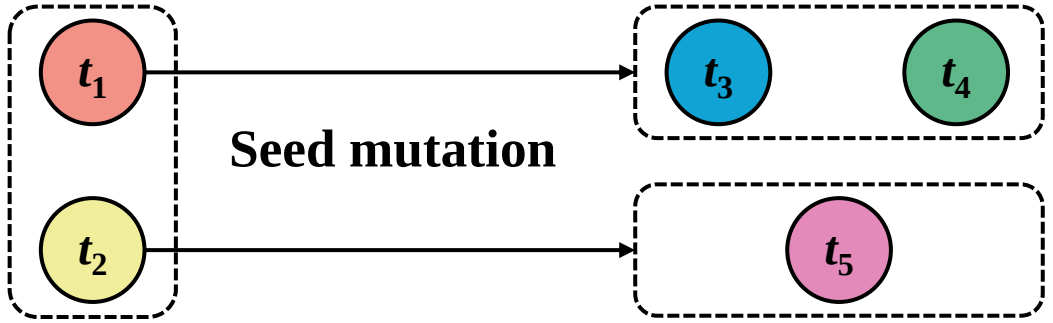



**Initial seeds**

# Motivation

## Motivating example

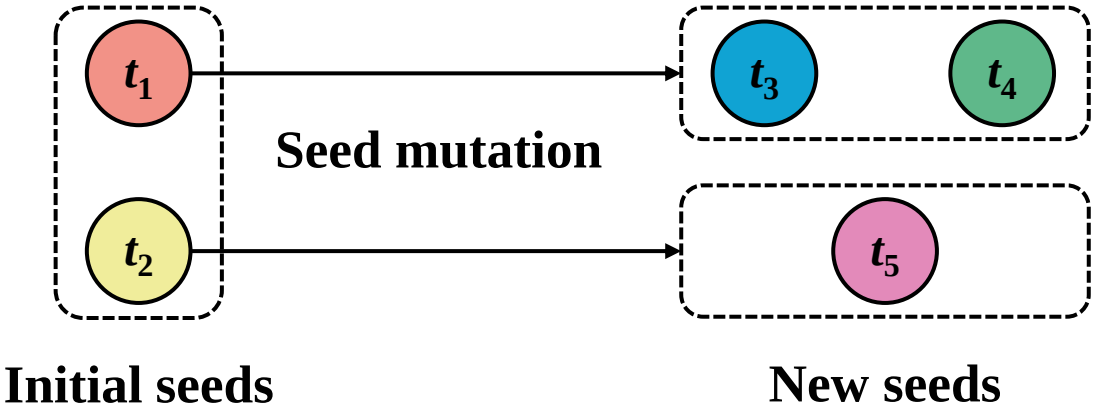
| | code | test cases | | | | |
|---|---|---|---|---|---|---|
| | | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ |
| | | 'x' | 'y' | 'xaf' | 'xc' | 'ym' |
| | int func1 (input) { | | | | | |
| $b_0$ | if (input[0] == 'x') | ● | ● | ● | ● | ● |
| $b_1$ | func2(input); | ● | | ● | ● | |
| $b_2$ | else if (input[0] == 'y') | | ● | | | ● |
| $b_3$ | func3(input); | | ● | | | ● |
| | } | | | | | |
| | int func2 (input) { | | | | | |
| $b_4$ | if (input[1] == 'a') | ● | | ● | ● | |
| $b_5$ | if (input[2] == 'f') | | | ● | | |
| $b_6$ | … | | | ● | | |
| $b_7$ | else if (input[1] == 'b') | ● | | | | |
| $b_8$ | … | | | | | |
| $b_9$ | else if (input[1] == 'c') | ● | | | ● | |
| $b_{10}$ | … | | | | ● | |
| | } | | | | | |
| | int func3 (input) { | | | | | |
| $b_{11}$ | if (input[1] == 'm') | | ● | | | ● |
| $b_{12}$ | if (input[2] == 'n') | | | | | ● |
| $b_{13}$ | func4 (input); | | | | | |
| $b_{14}$ | else exit (0); | | ● | | | |
| | } | | | | | |

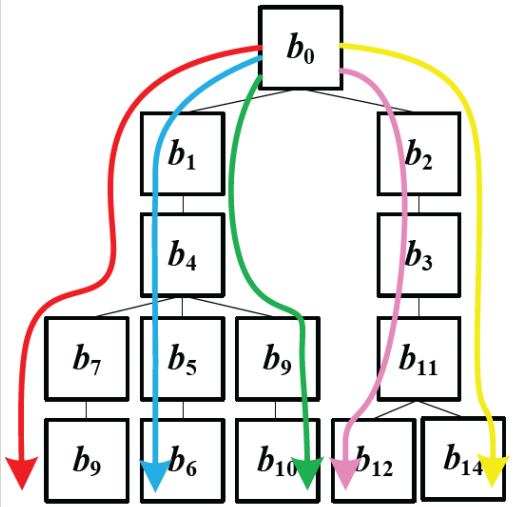


# Motivation

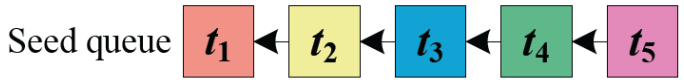
## Motivating example

| | code | test cases | | | | |
|---|---|---|---|---|---|---|
| | | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ |
| | | 'x' | 'y' | 'xaf' | 'xc' | 'ym' |
| | int func1 (input) { | | | | | |
| $b_0$ | if (input[0] == 'x') | ● | ● | ● | ● | ● |
| $b_1$ | func2(input); | ● | | ● | ● | |
| $b_2$ | else if (input[0] == 'y') | | ● | | | ● |
| $b_3$ | func3(input); | | ● | | | ● |
| | } | | | | | |
| | int func2 (input) { | | | | | |



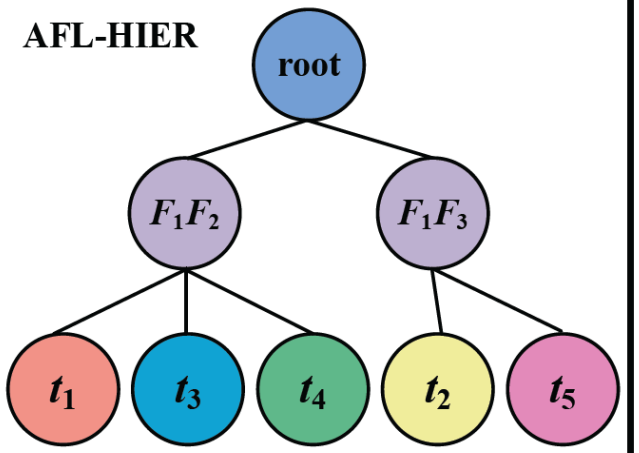




(a) Execution tree

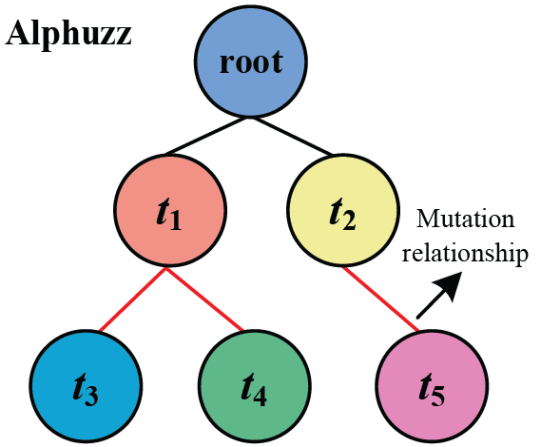



| | |
|---|---|
| **AFL** | Smaller size & Less execution time |
| **AFLFast** | Low-frequency paths |
| **FairFuzz** | Rare branches |
| **EcoFuzz** | Newly generated & Highest reward |

(b) Individual features for seed scheduling



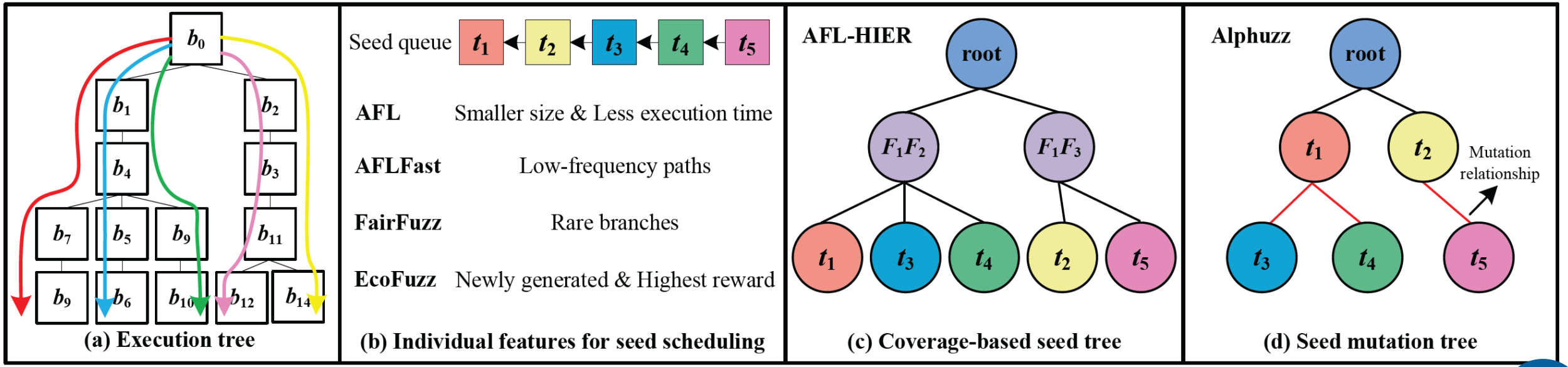

(c) Coverage-based seed tree




(d) Seed mutation tree

# Motivation

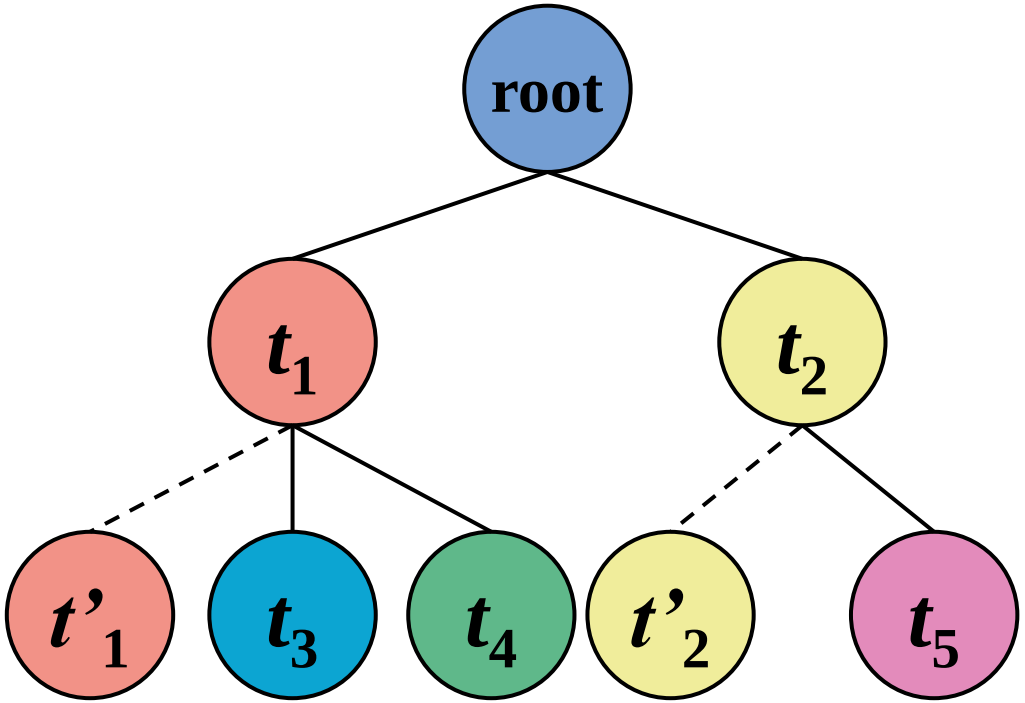
**Seed scheduling strategy should consider both the relationships among seeds and the individual features of seeds.**



# Seed Mutation Tree

- **Insight**

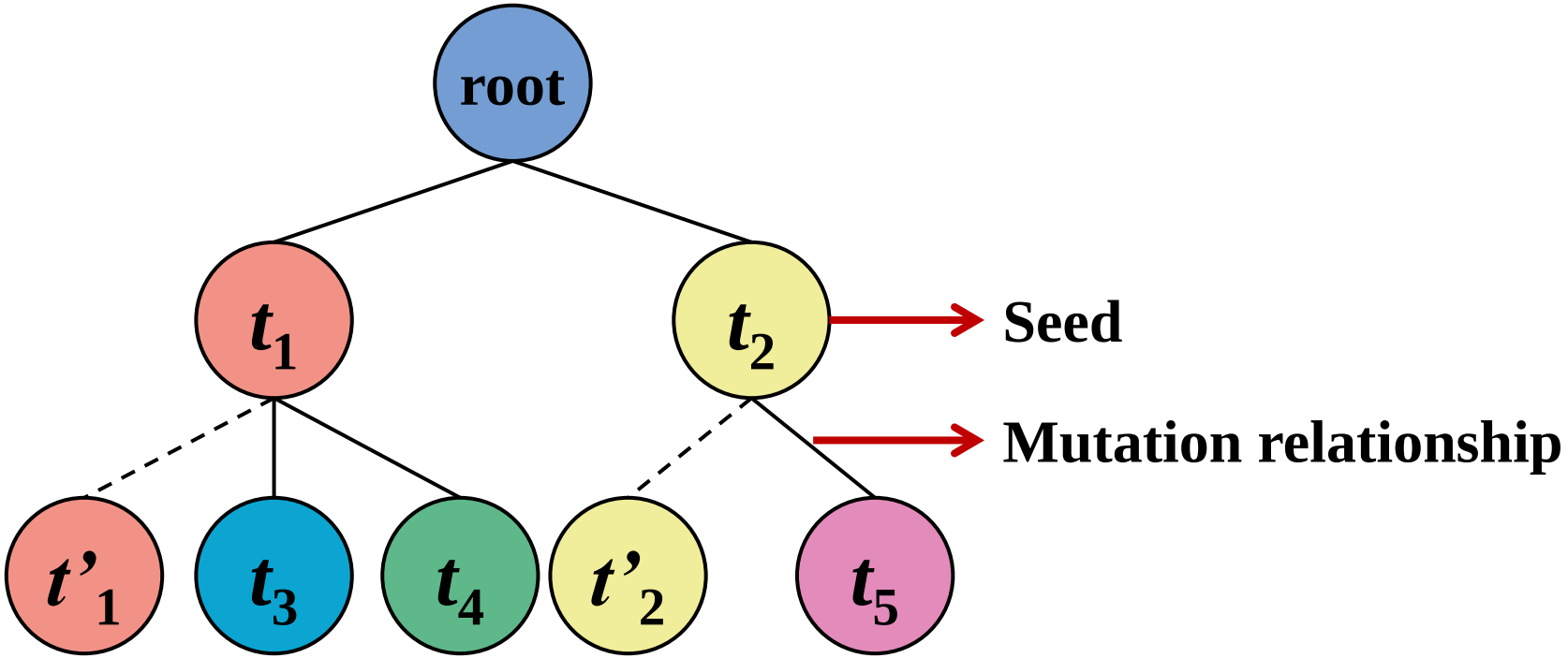
  We leverage the **mutation relationships** among seeds to construct a “seed mutation tree”, which can be an approximation of the execution tree.



# Seed Mutation Tree

- **Insight**

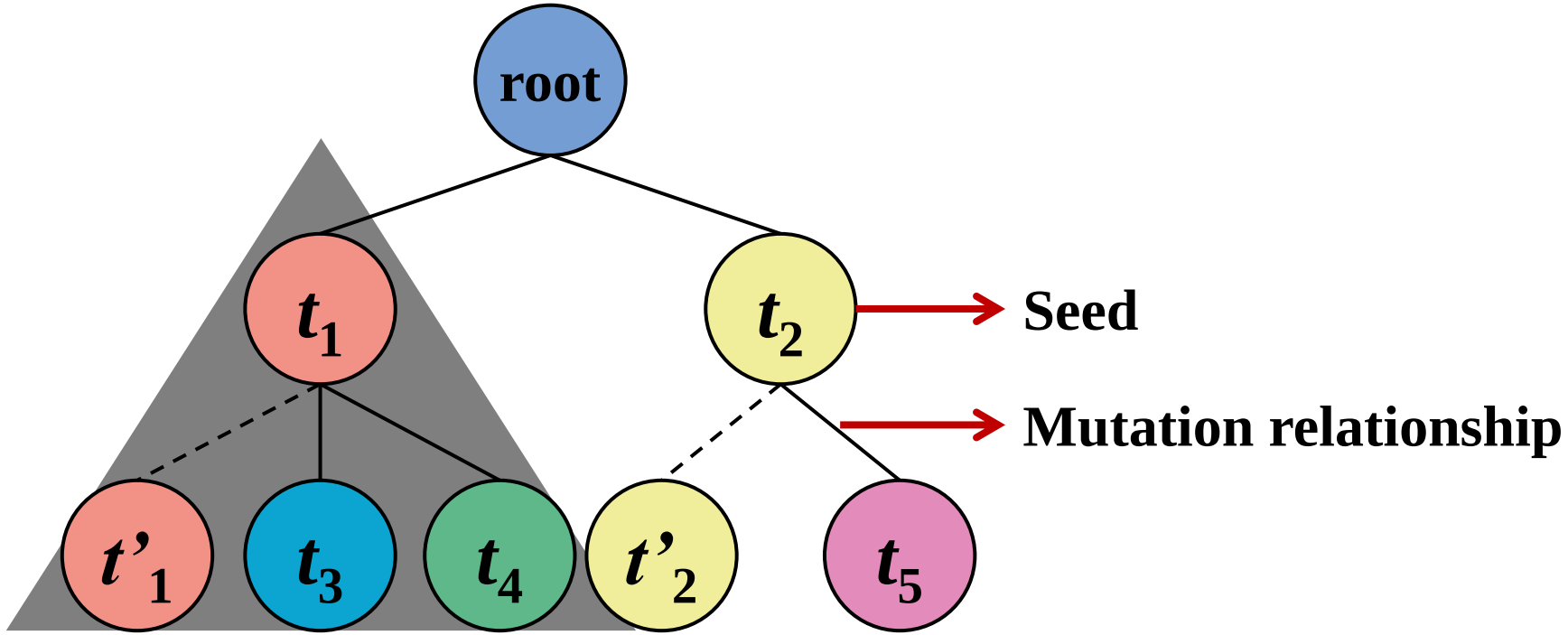
  We leverage the **mutation relationships** among seeds to construct a “seed mutation tree”, which can be an approximation of the execution tree.



# Seed Mutation Tree

- **Insight**

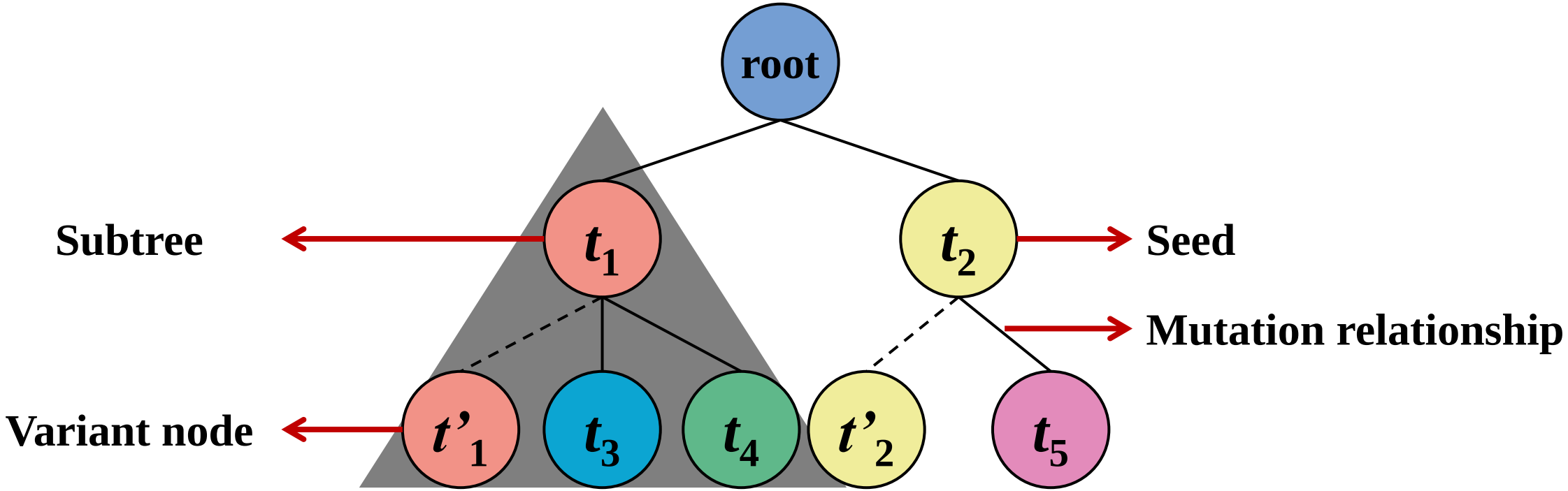
  We leverage the **mutation relationships** among seeds to construct a “seed mutation tree”, which can be an approximation of the execution tree.



# Seed Mutation Tree

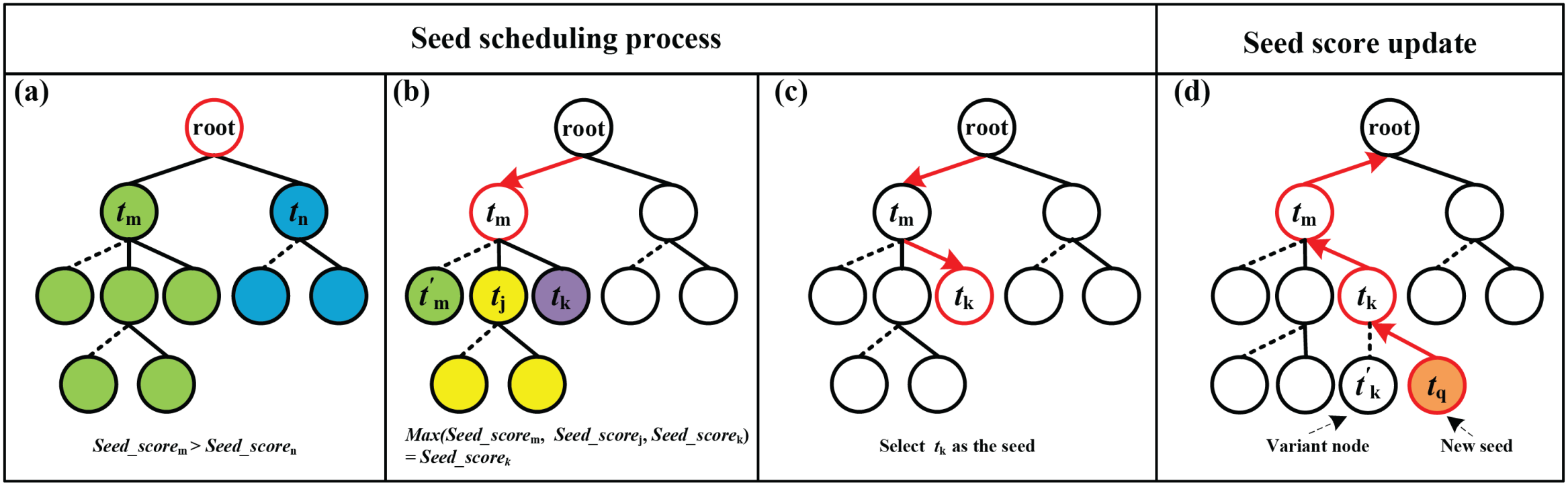
- **Insight**

  We leverage the **mutation relationships** among seeds to construct a “seed mutation tree”, which can be an approximation of the execution tree.



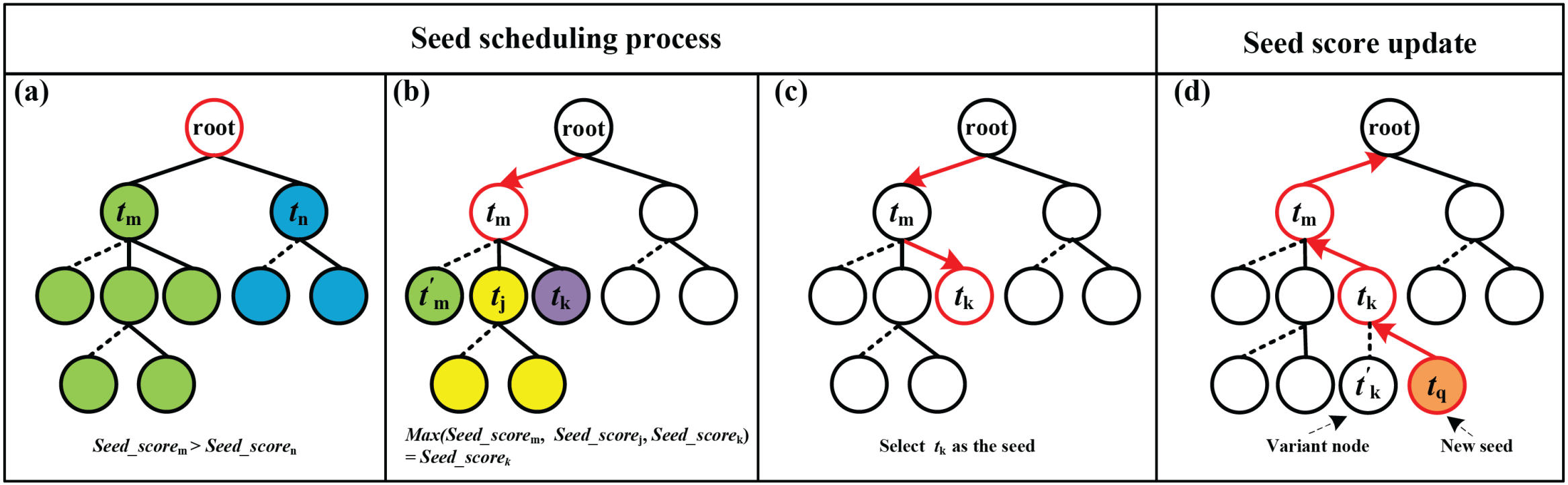
# MCTS-based Seed Scheduling

- **Seed scheduling on seed mutation tree**



# MCTS-based Seed Scheduling

- **Seed scheduling on seed mutation tree**




$$seed\_score = \frac{q_i}{n_i} + k\sqrt{\frac{lnN_i}{n_i}}$$

$\boldsymbol{q_i}$ : the number of branches only covered by node $i$ compared to other sibling nodes.

$\boldsymbol{n_i}$ : the selected time of node i . $\boldsymbol{N_i}$: the selected time of the parent node.

# Evaluation

- Prototypes
  - Alphuzz
  - Alphuzz++
- Datasets (24 hours,10 rounds)
  - 188 CGC binaries
  - 18 UniFuzz binaries
  - 12 real-world binaries
- Baseline technologies
  - AFL-based: AFL, AFLFast, FairFuzz, EcoFuzz
  - AFL++-based: AFL++, AFL-Hier++

# Evaluation

- Code coverage
  - UniFuzz (AFL-based)

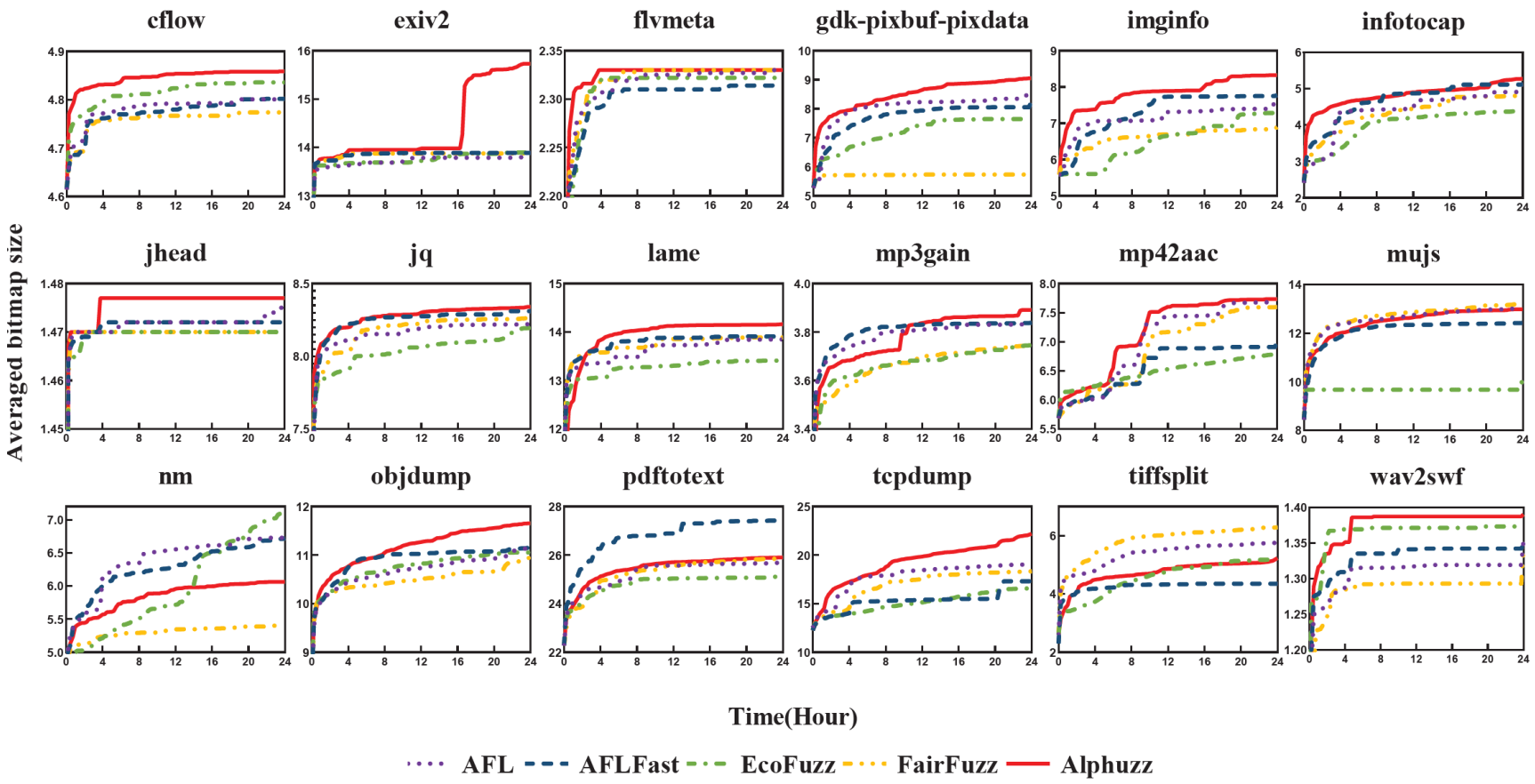



Figure 5: Average bitmap size for every binary of UniFuzz. Each graph represents a binary, with the abscissa representing the time and the ordinate representing the size of the averaged bitmap size.

AFL: 15

AFLFast:15

EcoFuzz: 16

FAirFuzz:14

# Evaluation

- Code coverage
  - UniFuzz (AFL++-based)

AFL++: 14

AFL++-Hier: 15

**Table 6: $p$ values of the code coverage on UniFuzz with ALPHUZZ as the baseline.**

| Binary | ALPHUZZ | AFL | AFLFast | EcoFuzz | FairFuzz |
|---|---|---|---|---|---|
| | AVG cov | $p$ value | $p$ value | $p$ value | $p$ value |
| cflow | 4.859% | ***0.0209*** | ***0.0408*** | 0.4795 | ***0.0078*** |
| exiv2 | 15.728% | ***0.0047*** | ***0.0023*** | 0.0511 | ***0.0054*** |
| flvmeta | 2.33 % | >0.10000 | ***0.0001*** | 0.0867 | >0.9999 |
| gdk | 9.05% | ***<0.0001*** | ***0.0014*** | ***0.0014*** | ***0.0014*** |
| imginfo | 8.324% | ***0.0002*** | ***0.0047*** | ***0.0082*** | ***0.0002*** |
| infotocap | 5.267% | 0.4221 | 0.3046 | ***<0.0001*** | ***0.0027*** |
| jhead | 1.477% | >0.9999 | 0.0698 | ***0.0031*** | ***0.0031*** |
| jq | 8.34% | ***0.0435*** | 0.8951 | ***0.022*** | 0.1898 |
| lame | 14.161% | 0.4648 | 0.2387 | 0.1008 | 0.4692 |
| mp3gain | 3.891% | 0.9887 | 0.4027 | 0.1342 | 0.1013 |
| mp42aac | 7.734% | ***0.0002*** | ***<0.0001*** | ***<0.0001*** | ***<0.0001*** |
| mujs | 12.984% | 0.3047 | ***<0.0001*** | ***<0.0001*** | ***<0.0001*** |
| nm | 6.064% | 0.1172 | ***0.0209*** | ***<0.0001*** | ***0.0217*** |
| objdump | 11.654% | 0.8978 | 0.3047 | 0.4699 | 0.4695 |
| pdftotext | 25.904% | 0.6415 | ***0.0024*** | 0.3035 | 0.9126 |
| tcpdump | 22.204% | ***0.0009*** | ***<0.0001*** | ***<0.0001*** | ***<0.0001*** |
| tiffsplit | 5.231% | 0.072 | 0.3227 | 0.7811 | ***0.0002*** |
| wav2swf | 1.39% | 0.3067 | ***0.0086*** | >0.9999 | 0.3043 |

# Evaluation

- Vulnerability detection
  - CGC

**Table 2: Numbers of discovered bugs on CGC.**

| Fuzzer | = 10 | ≥ 9 | ≥ 8 | ≥ 7 | ≥ 6 | ≥ 5 | ≥ 4 | ≥ 3 | ≥ 2 | ≥ 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| AlphuzZ | 70 | 72 | 72 | 73 | 75 | 75 | 80 | 83 | 87 | 87 |
| AFL | 69 | 69 | 69 | 70 | 71 | 75 | 78 | 83 | 83 | 83 |
| AFLFast | 67 | 67 | 70 | 71 | 73 | 77 | 79 | 79 | 81 | 83 |
| EcoFuzz | 66 | 66 | 66 | 67 | 68 | 68 | 68 | 68 | 73 | 73 |
| FairFuzz | 67 | 67 | 68 | 70 | 71 | 72 | 72 | 73 | 74 | 76 |
| AlphuzZ++ | 72 | 73 | 75 | 75 | 77 | 80 | 80 | 85 | 88 | 88 |
| AFL++ | 70 | 72 | 75 | 75 | 75 | 76 | 79 | 83 | 83 | 84 |
| AFL++-Hier | 72 | 73 | 73 | 75 | 76 | 80 | 83 | 83 | 85 | 85 |

# Evaluation

- Vulnerability detection
  - UniFuzz

**Table 3: Numbers of discovered unique bugs.**

| Binary | AFL | AFLFast | EcoFuzz | FairFuzz | AlphUzz |
|---|---|---|---|---|---|
| cflow | 3 | 4 | 2 | 3 | 4 |
| exiv2 | 0 | 1 | 1 | 1 | 1 |
| flvmeta | 3 | 3 | 3 | 4 | 3 |
| gdk | 6 | 6 | 2 | 0 | 8 |
| imginfo | 0 | 1 | 0 | 0 | 1 |
| infotocap | 2 | 3 | 0 | 2 | 2 |
| jhead | 0 | 0 | 2 | 0 | 0 |
| jq | 0 | 0 | 0 | 0 | 0 |
| lame | 3 | 2 | 2 | 3 | 3 |
| mp3gain | 6 | 7 | 4 | 3 | 7 |
| mp42aac | 2 | 2 | 2 | 2 | 2 |
| mujs | 0 | 0 | 0 | 0 | 1 |
| nm | 0 | 0 | 0 | 0 | 0 |
| objdump | 3 | 3 | 2 | 1 | 4 |
| pdftotext | 1 | 1 | 0 | 1 | 8 |
| tcpdump | 0 | 0 | 0 | 0 | 0 |
| tiffsplit | 4 | 5 | 3 | 8 | 6 |
| wav2swf | 2 | 3 | 2 | 2 | 3 |
| **total** | **35** | **41** | **25** | **30** | **53** |
| | **+18** | **+12** | **+28** | **+23** | |

**Table 4: Number of discovered unique exploitable bugs.**

| Binary | AFL | AFLFast | EcoFuzz | FairFuzz | AlphUzz |
|---|---|---|---|---|---|
| cflow | 0 | 0 | 0 | 0 | 0 |
| exiv2 | 0 | 1 | 1 | 1 | 1 |
| flvmeta | 1 | 1 | 1 | 1 | 1 |
| gdk | 4 | 3 | 2 | 0 | 3 |
| imginfo | 0 | 0 | 0 | 0 | 0 |
| infotocap | 0 | 0 | 0 | 0 | 0 |
| jhead | 0 | 0 | 2 | 0 | 0 |
| jq | 0 | 0 | 0 | 0 | 0 |
| lame | 2 | 1 | 1 | 1 | 2 |
| mp3gain | 1 | 1 | 0 | 0 | 1 |
| mp42aac | 0 | 0 | 0 | 0 | 0 |
| mujs | 0 | 0 | 0 | 0 | 0 |
| nm | 0 | 0 | 0 | 0 | 0 |
| objdump | 2 | 2 | 0 | 1 | 2 |
| pdftotext | 0 | 1 | 0 | 0 | 3 |
| tcpdump | 0 | 0 | 0 | 0 | 0 |
| tiffsplit | 1 | 1 | 1 | 1 | 1 |
| wav2swf | 2 | 3 | 2 | 2 | 3 |
| **total** | **13** | **14** | **10** | **6** | **17** |
| | | **+4** | **+3** | **+7** | **+11** |

+18

# Evaluation

- Vulnerability detection
  - UniFuzz

**Table 5: Number of discovered unique bug and exploitable vulnerabilities found by AFL++-based techniques.**

| Binary | Unique bugs | | | Exploitable bugs | | |
|---|---|---|---|---|---|---|
| | AFL++ | AFL++-HIER | AlphuzZ++ | AFL++ | AFL++-HIER | AlphuzZ++ |
| cflow | 3 | 5 | 5 | 1 | 1 | 1 |
| exiv2 | 2 | 2 | 2 | 1 | 1 | 1 |
| flvmeta | 4 | 3 | 4 | 1 | 1 | 1 |
| gdk | 10 | 9 | 12 | 6 | 3 | 8 |
| imginfo | 0 | 1 | 1 | 0 | 0 | 0 |
| infotocap | 2 | 0 | 3 | 0 | 0 | 0 |
| jhead | 3 | 3 | 3 | 0 | 0 | 0 |
| jq | 0 | 0 | 0 | 0 | 0 | 0 |
| lame | 2 | 3 | 3 | 2 | 2 | 2 |
| mp3gain | 6 | 6 | 6 | 1 | 1 | 1 |
| mp42aac | 2 | 2 | 2 | 0 | 0 | 0 |
| mujs | 0 | 0 | 1 | 0 | 0 | 0 |
| nm | 0 | 0 | 0 | 0 | 0 | 0 |
| objdump | 2 | 3 | 3 | 2 | 1 | 2 |
| pdftotext | 2 | 8 | 4 | 2 | 2 | 3 |
| tcpdump | 0 | 0 | 1 | 0 | 0 | 0 |
| tiffsplit | 3 | 6 | 5 | 1 | 1 | 1 |
| wav2swf | 3 | 2 | 4 | 3 | 2 | 4 |
| **total** | **44** | **55** | **59** | **20** | **15** | **24** |

- 12 real-world binaries

**3 new CVEs**

**Table 9: Vulnerabilities discovered in real-world binaries.**

| Binary | Vulnerabilities | AFL | AFLFast | EcoFuzz | FairFuzz | AlphuzZ |
|---|---|---|---|---|---|---|
| cjpeg | CVE-2018-11214 | ✓ | ✓ | ✓ | ✓ | ✓ |
| | CVE-2018-11212 | ✓ | ✓ | ✓ | ✓ | ✓ |
| | issue#5 | ✗ | ✗ | ✗ | ✗ | ✓ |
| infotocap | CVE-2018-19211 | ✗ | ✓ | ✓ | ✗ | ✓ |
| | **CVE-2021-25794** | ✗ | ✗ | ✗ | ✗ | ✓ |
| mp3gain | CVE-2018-10778 | ✓ | ✓ | ✓ | ✓ | ✓ |
| | CVE-2018-10777 | ✓ | ✓ | ✓ | ✓ | ✓ |
| | CVE-2017-14406 | ✗ | ✗ | ✗ | ✓ | ✗ |
| | CVE-2018-10776 | ✗ | ✗ | ✗ | ✗ | ✓ |
| pdfimages | issue#42073 | ✗ | ✗ | ✗ | ✗ | ✓ |
| | **CVE-2021-25792** | ✗ | ✗ | ✗ | ✗ | ✓ |
| | **CVE-2021-25793** | ✗ | ✓ | ✗ | ✗ | ✓ |
| **Total** | **12** | **4** | **6** | **5** | **5** | **11** |

# Evaluation

- Throughput

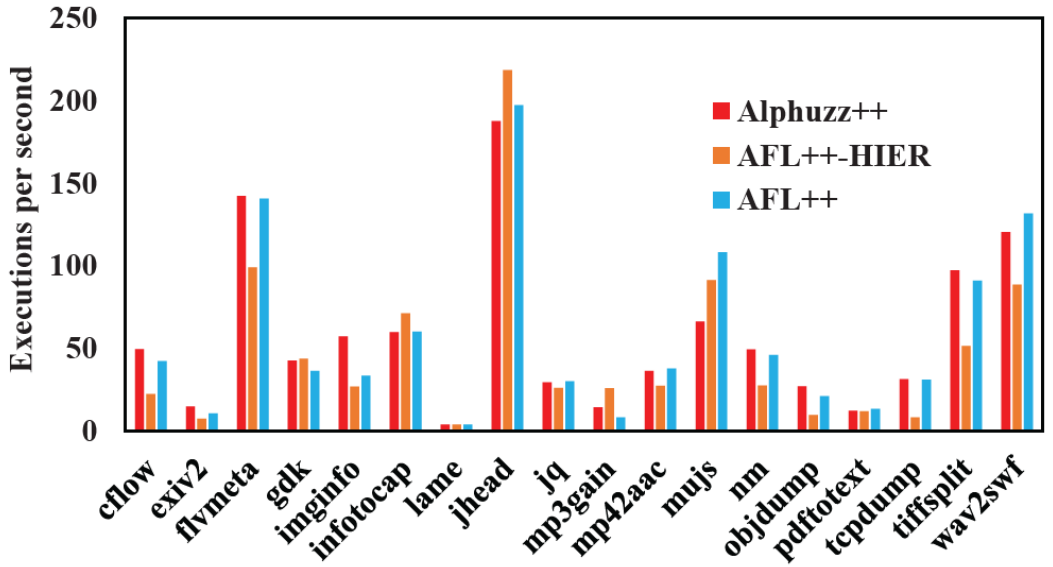



Figure 6: Throughput comparison.

- Parameter k

**Table 8: The Averaged edge coverage on CGC and UniFuzz with different values of the parameter $k$.**

| Dataset | Value of $k$ | | | | |
|---|---|---|---|---|---|
| | 0 | 0.014 | 0.14 | 1.4 | 14 |
| CGC | 2.59% | 2.61% | 2.65% | 2.75% | 2.58% |
| UniFuzz | 8.38% | 8.46% | 8.38% | 8.23% | 8.22% |

# Conclusion

- New insight
  - Key observation: the relationships among seeds are valuable for seed scheduling.
  - Seed mutation tree
- New fuzzing technology
  - MCTS-based seed scheduling strategy.
- Open-source implementation
  - https://github.com/zzyyrr/Alphuzz_overview.git

# Thank you!

## Alphuzz: Monte Carlo Search on Seed-Mutation Tree for Coverage-Guided Fuzzing

Yiru Zhao, Xiaoke Wang, Lei Zhao, Yueqiang Cheng, and Heng Yin

Corresponding author: Lei Zhao, email: leizhao@whu.edu.cn