MOPT: Optimize Mutation Scheduling for Fuzzers

Chenyang Lyu Shouling Ji Chao Zhang Yuwei Li
Wei-Han Lee Yu Song Raheem Beyah

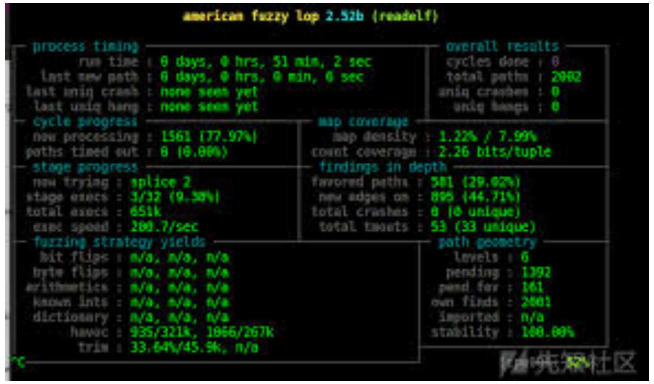
Fuzzing is a popular technique for exploring vulnerabilities

OSS-Fuzz

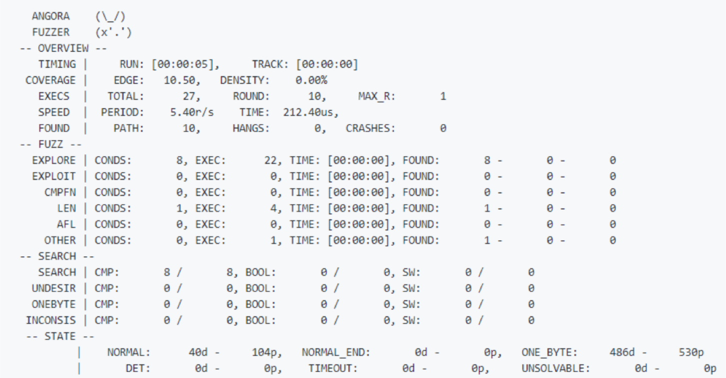
libFuzzer

ClusterFuzz

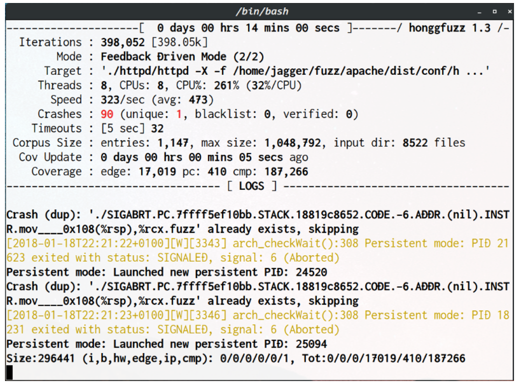
AFL



Angora

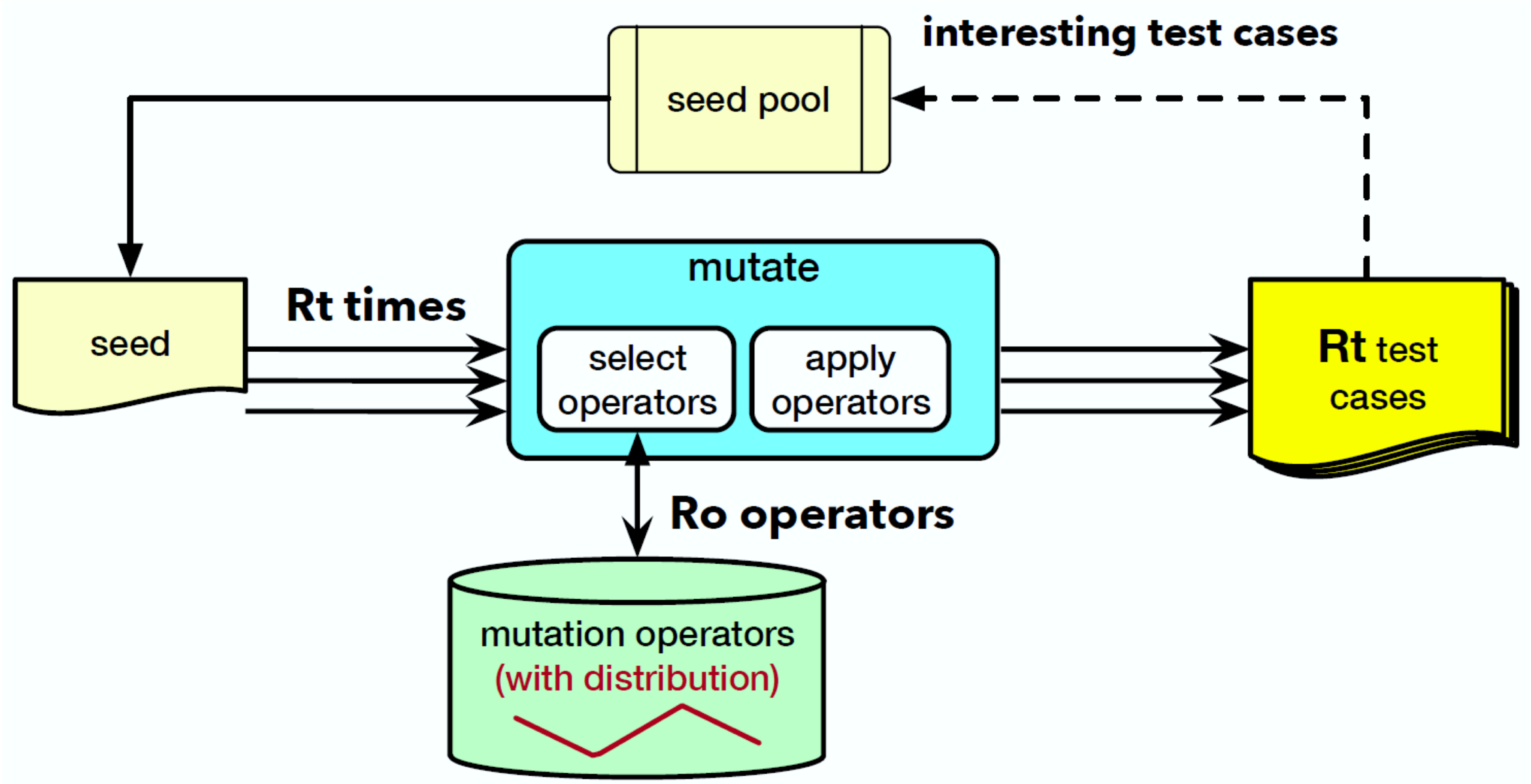


honggfuzz

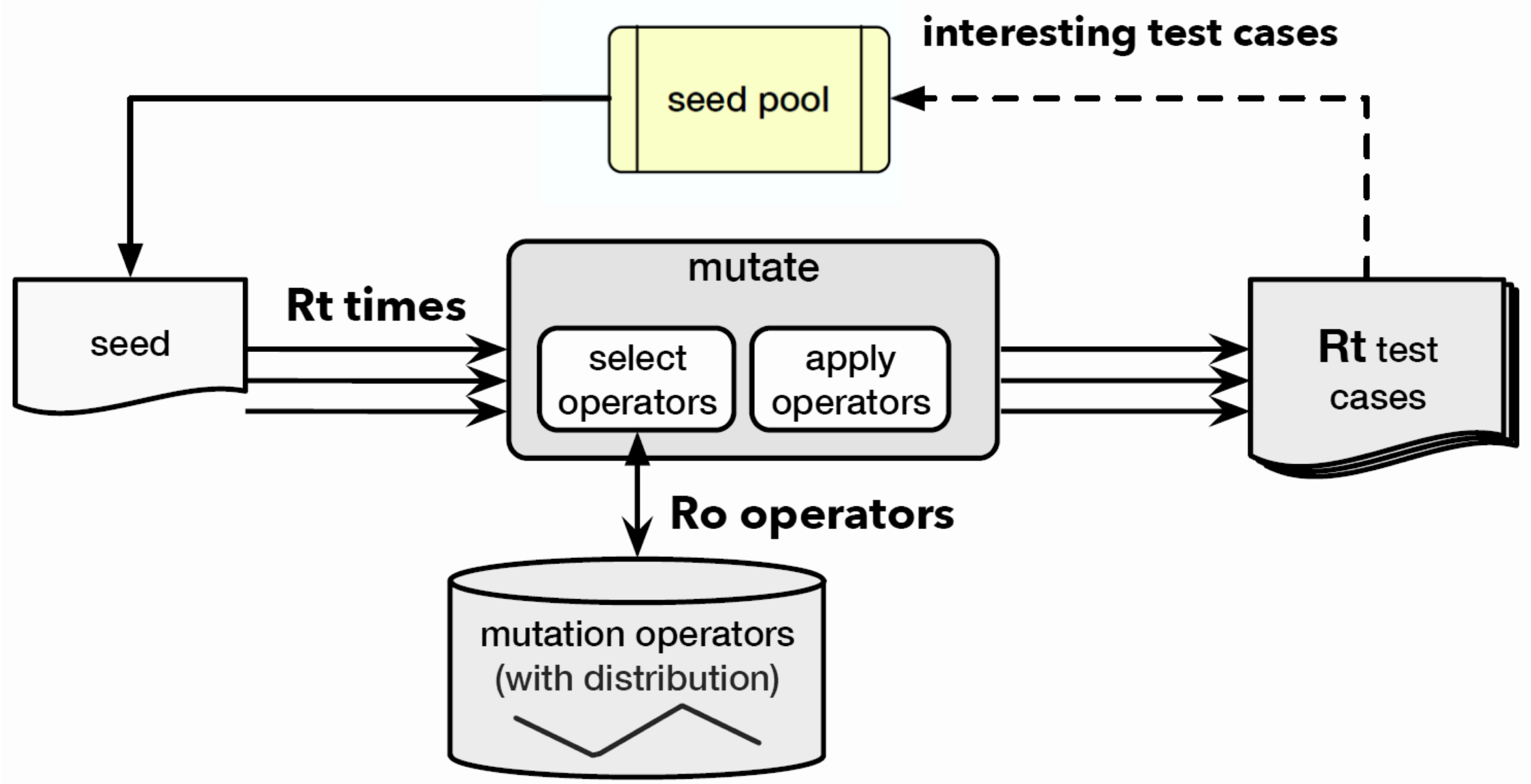


Mutation-based fuzzing

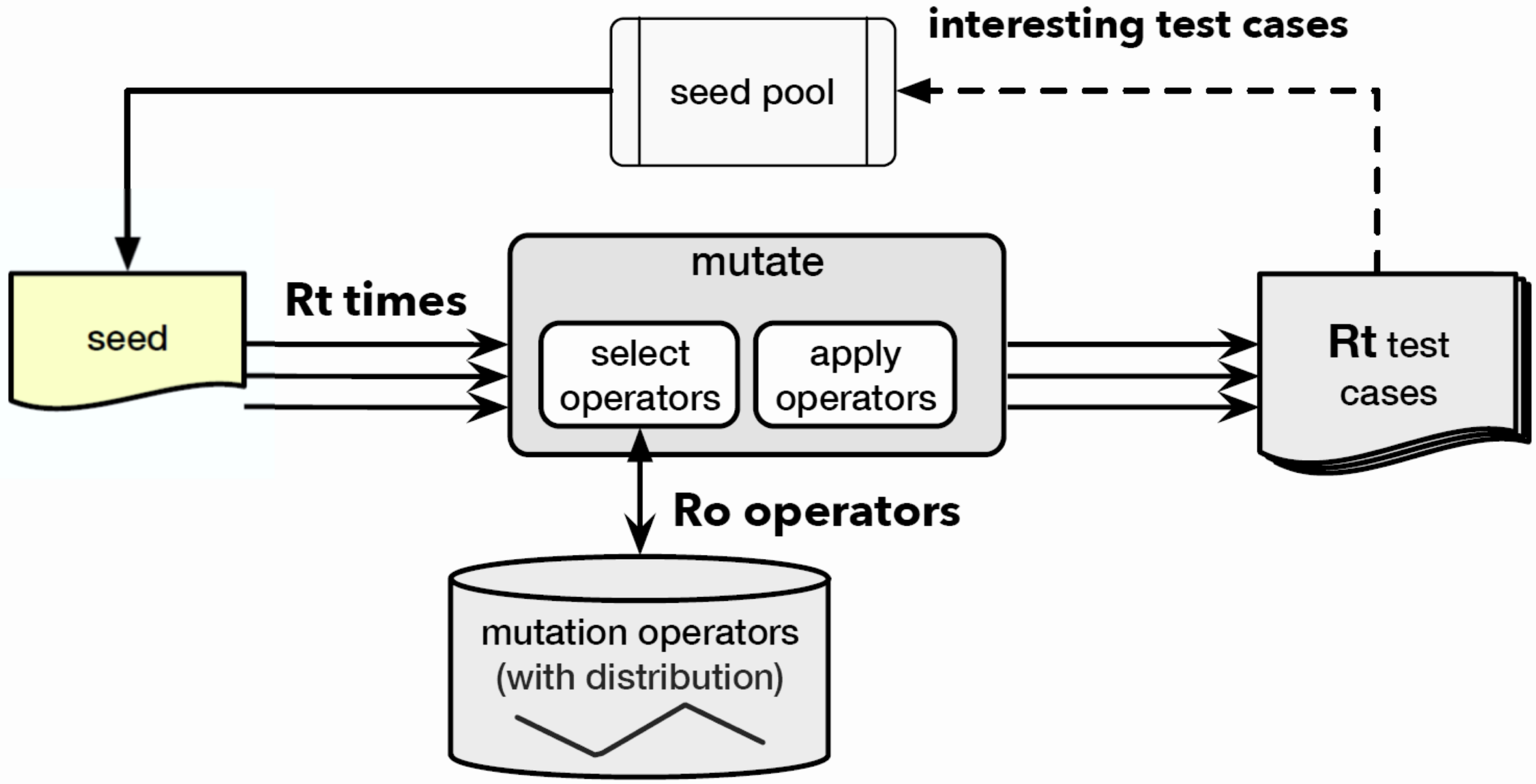
Mutation-based fuzzing



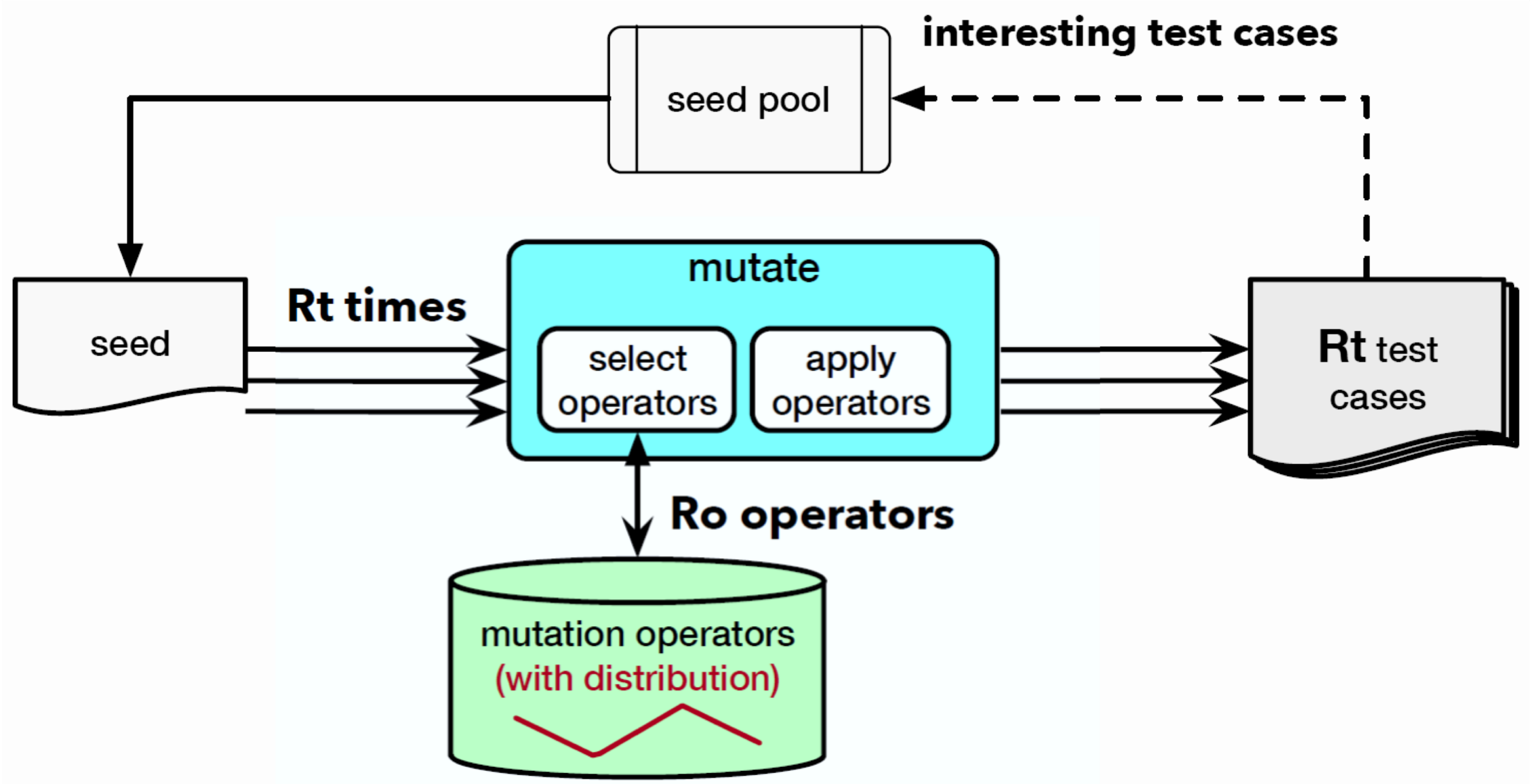
Mutation-based fuzzing



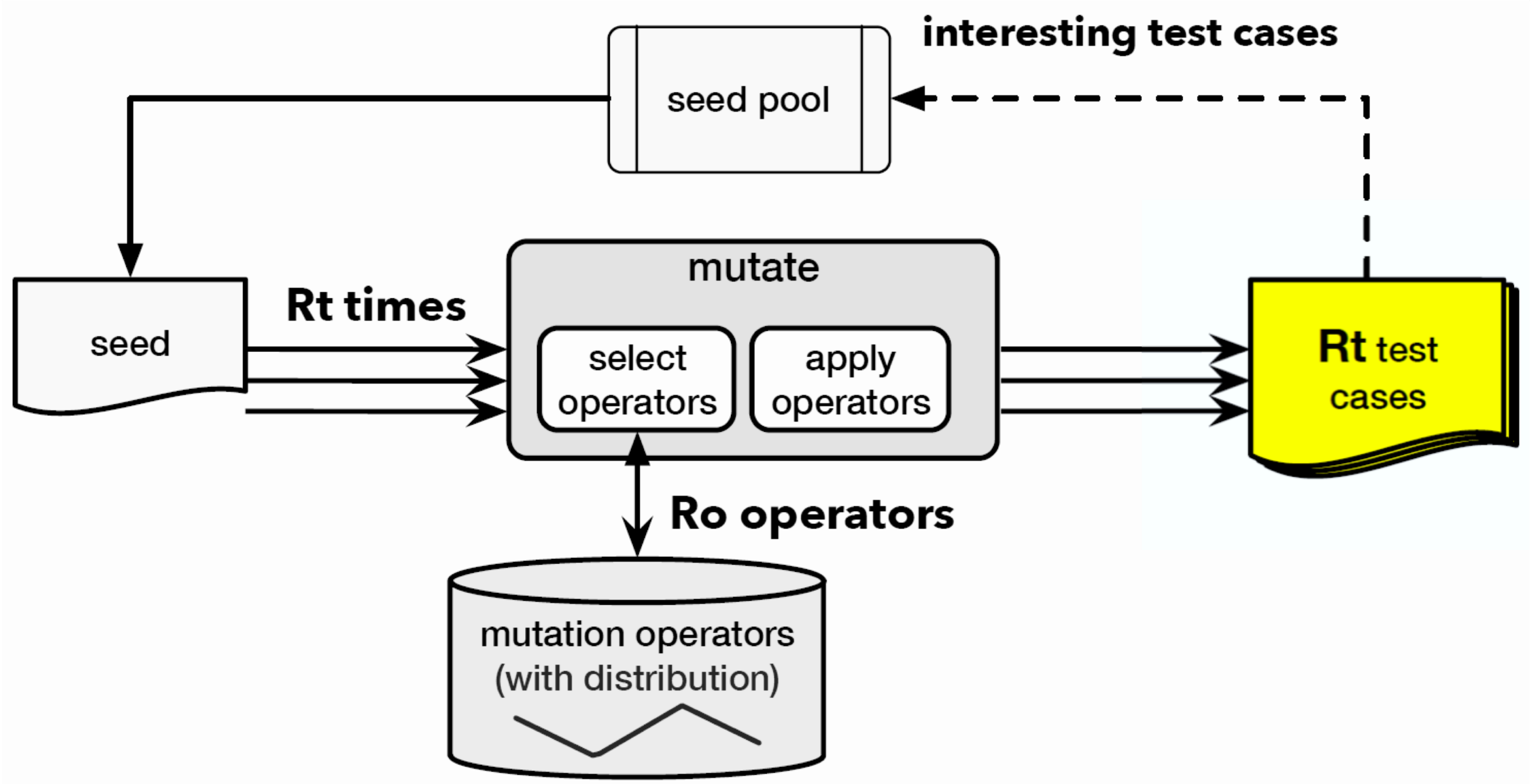
Mutation-based fuzzing



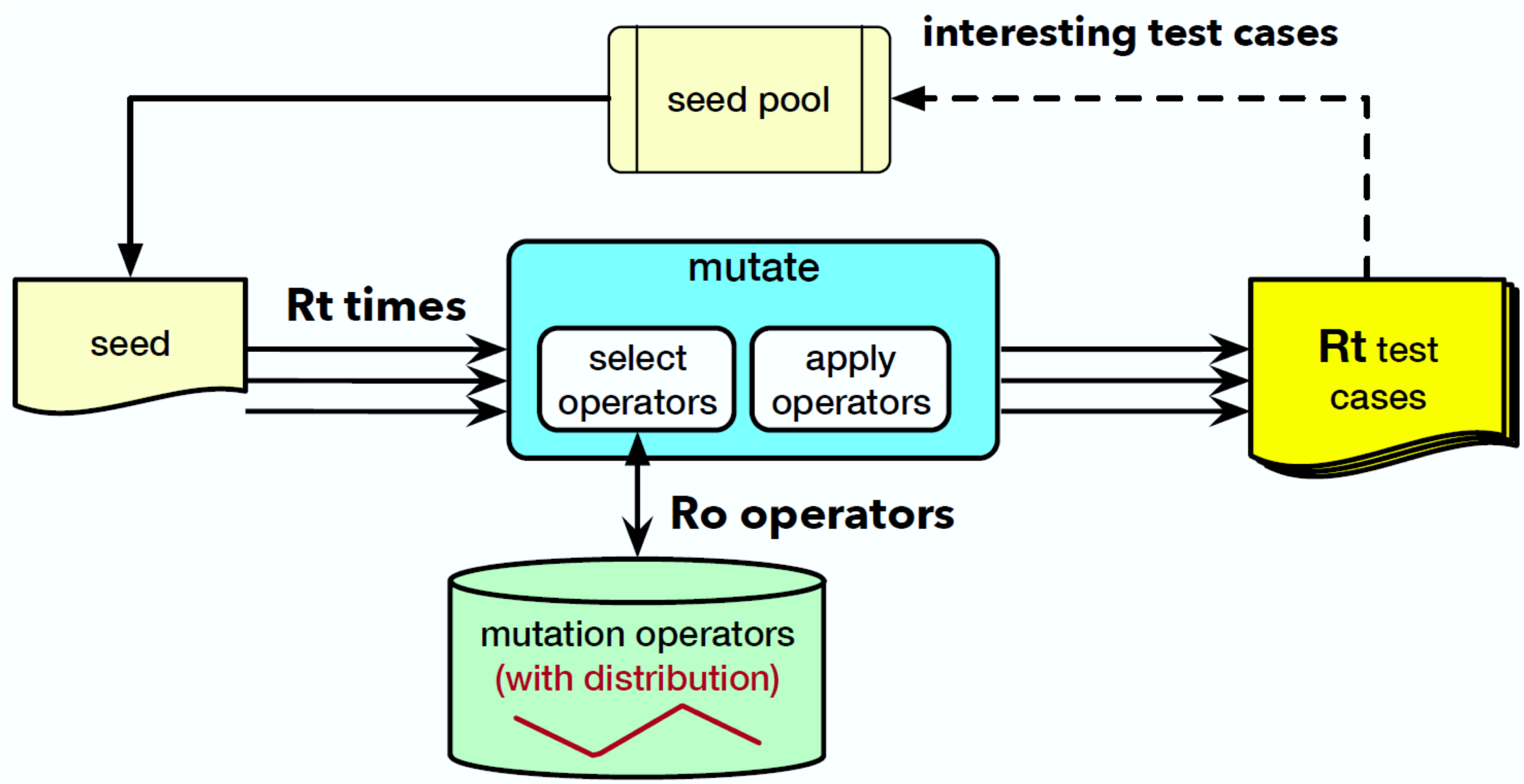
Mutation-based fuzzing



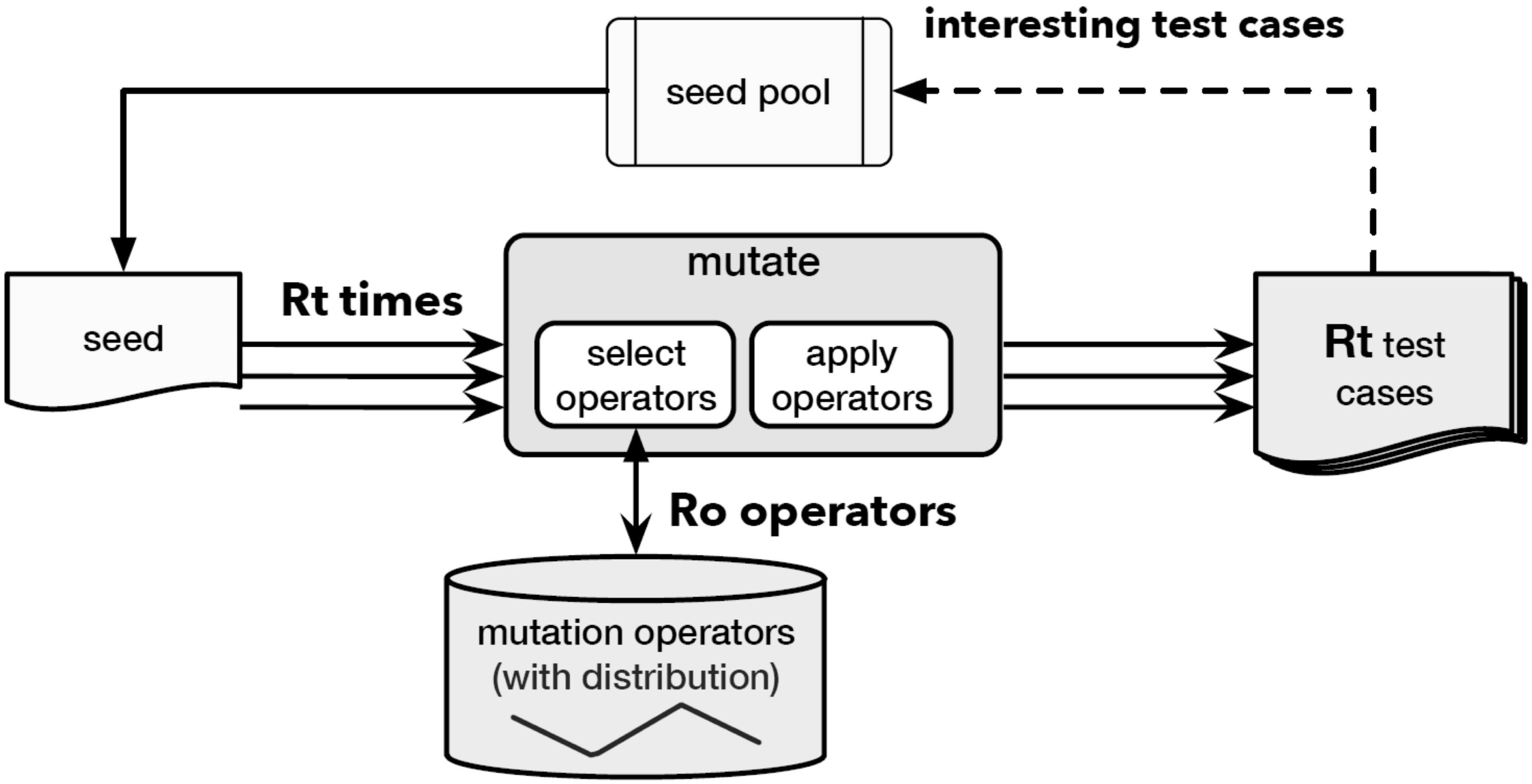
Mutation-based fuzzing



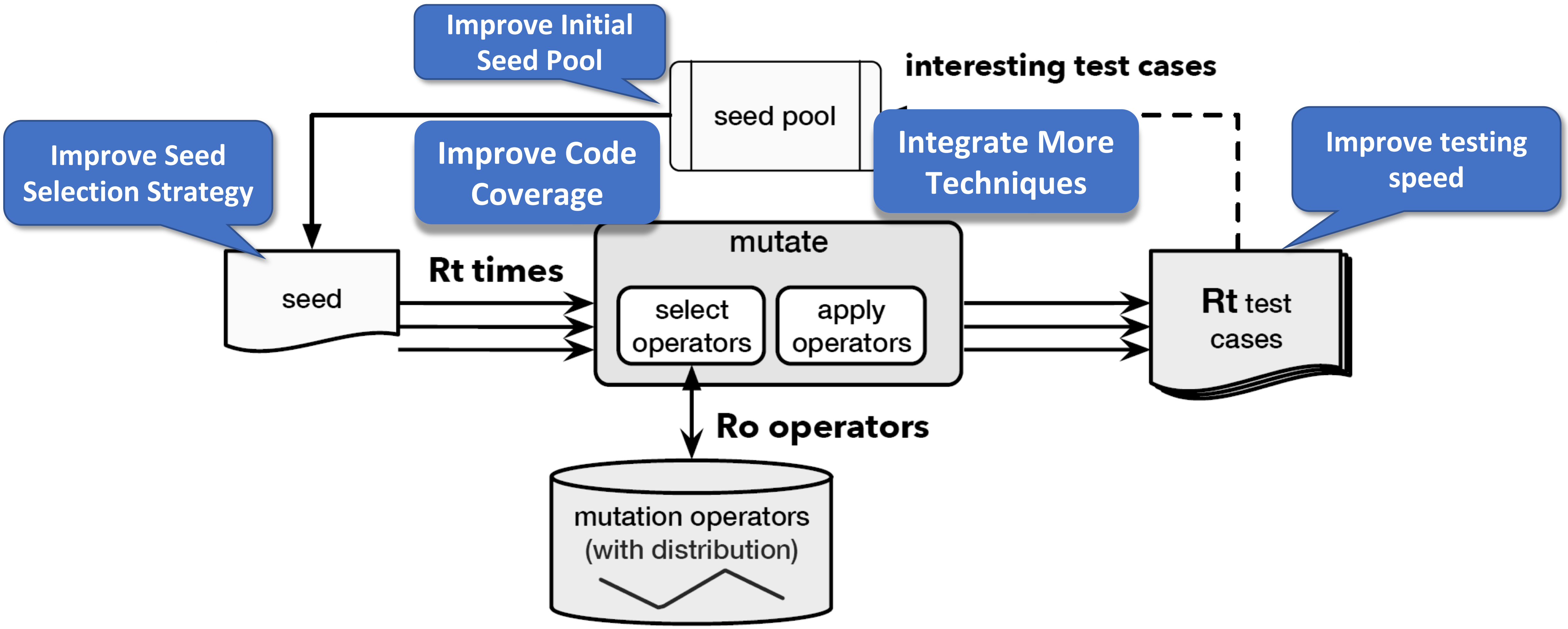
Mutation-based fuzzing



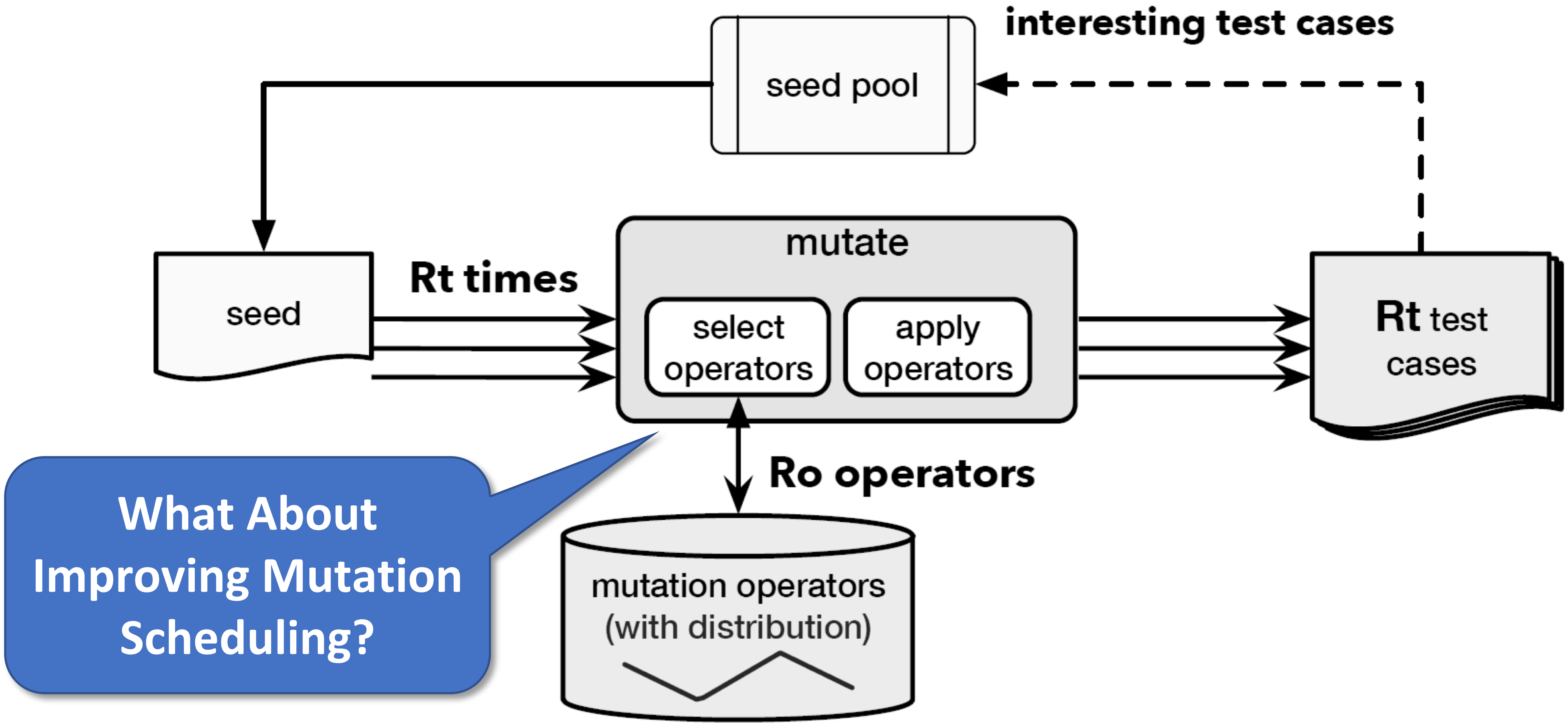
How to improve (mutation-based) fuzzing?



How to improve (mutation-based) fuzzing?



How to improve (mutation-based) fuzzing?



Mutation scheduling scheme

- How to select mutation operators for improving fuzzing?
  - Discover more unique paths
  - Discover more unique crashes
  - Discover more vulnerabilities
  - …

Mutation scheduling scheme

- How to select **mutation operators** for improving fuzzing?
  - Discover more unique paths
  - Discover more unique crashes
  - Discover more vulnerabilities
  - …

- What is mutation operators?

Mutation operators of AFL

- Mutation operators characterize where and how to mutate the seed.

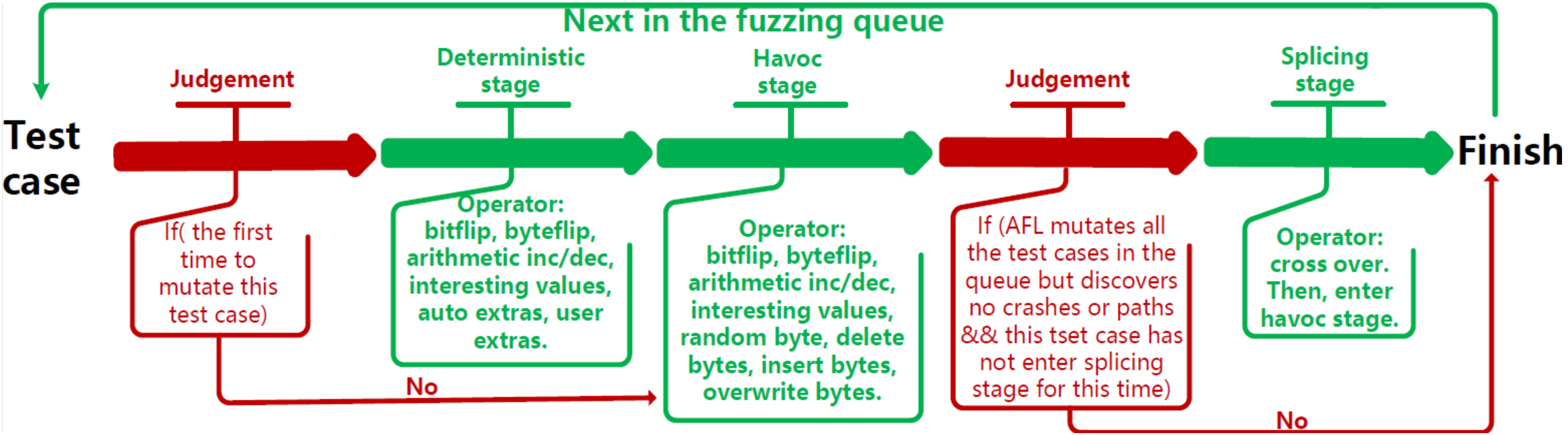
Type	Meaning	Operators
bitflip	Invert one or several consecutive bits in a test case, where the stepover is 1 bit.	`bitflip 1/1`, `bitflip 2/1`, `bitflip 4/1`
byteflip	Invert one or several consecutive bytes in a test case, where the stepover is 8 bits.	`bitflip 8/8`, `bitflip 16/8`, `bitflip 32/8`
arithmetic inc/dec	Perform addition and subtraction operations on one byte or several consecutive bytes.	`arith 8/8`, `arith 16/8`, `arith 32/8`

Some of the mutation operators in AFL.

The mutation operator *bitflip 2/1* represents flipping 2 consecutive bits, where the stepover is 1 bit

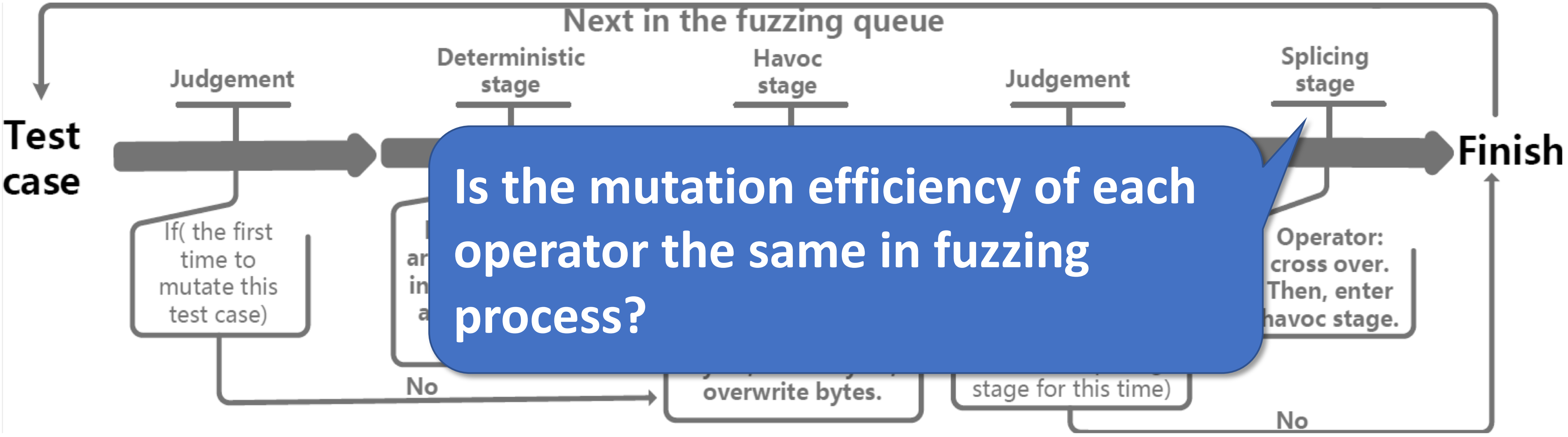
Mutation scheduling of AFL

- Three mutation stages:
  - Deterministic, havoc, and splicing

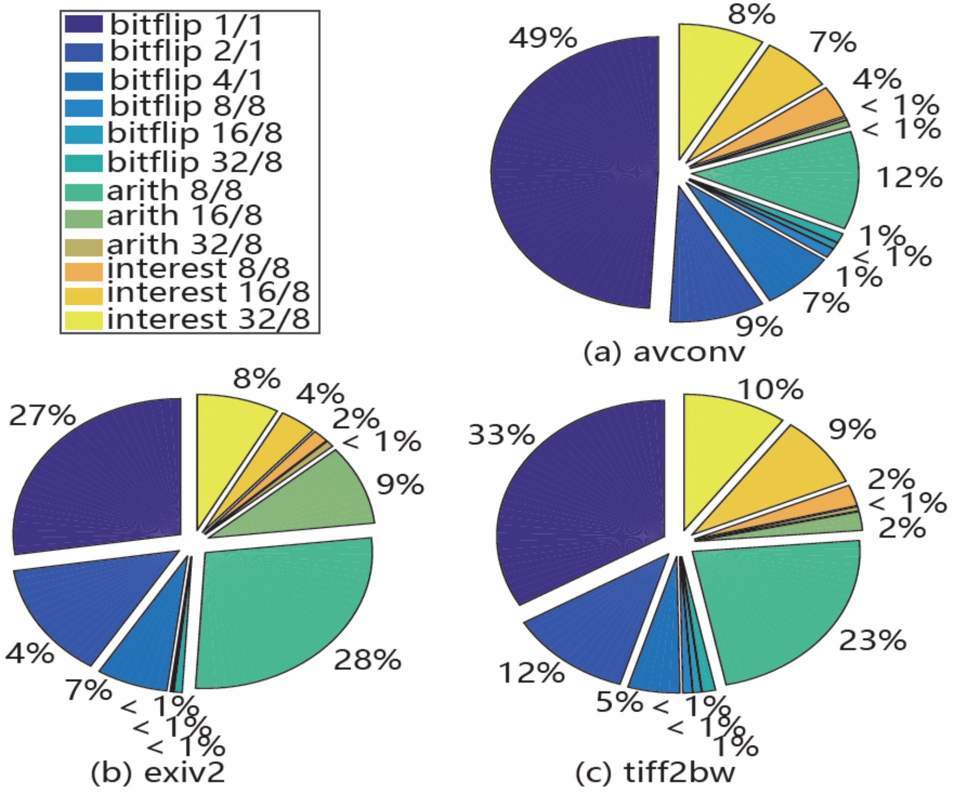


Mutation scheduling scheme of AFL

- Three mutation stages:
  - Deterministic, havoc, and splicing

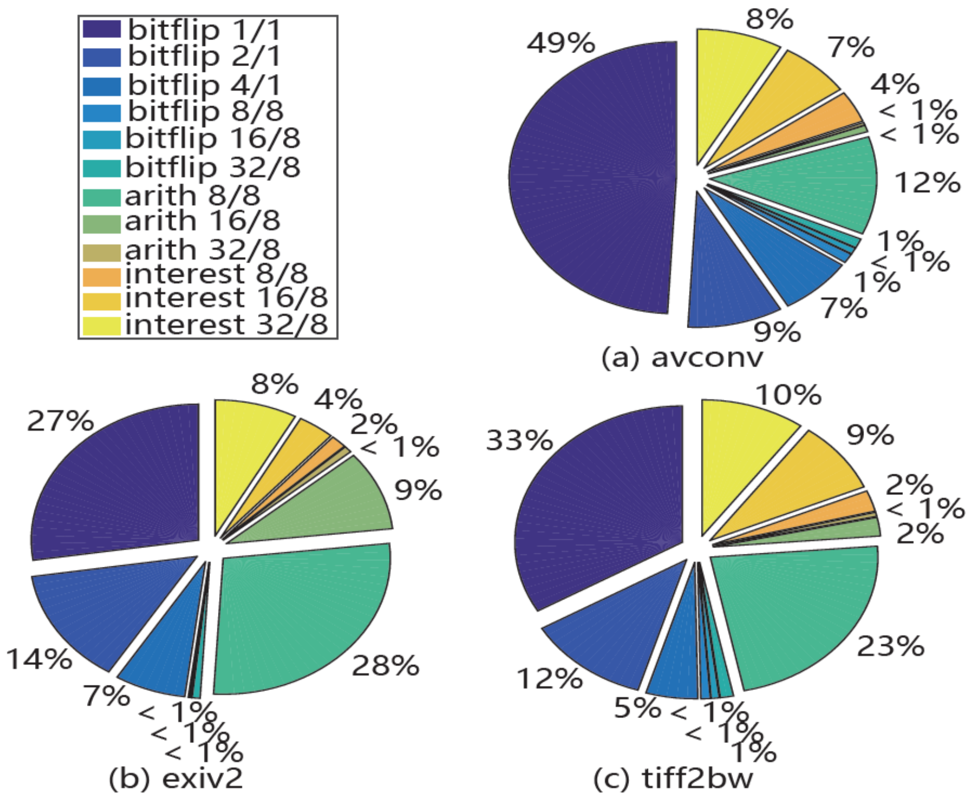


Mutation efficiency study on AFL



Percentages of interesting test cases produced by different operators in the deterministic stage of AFL

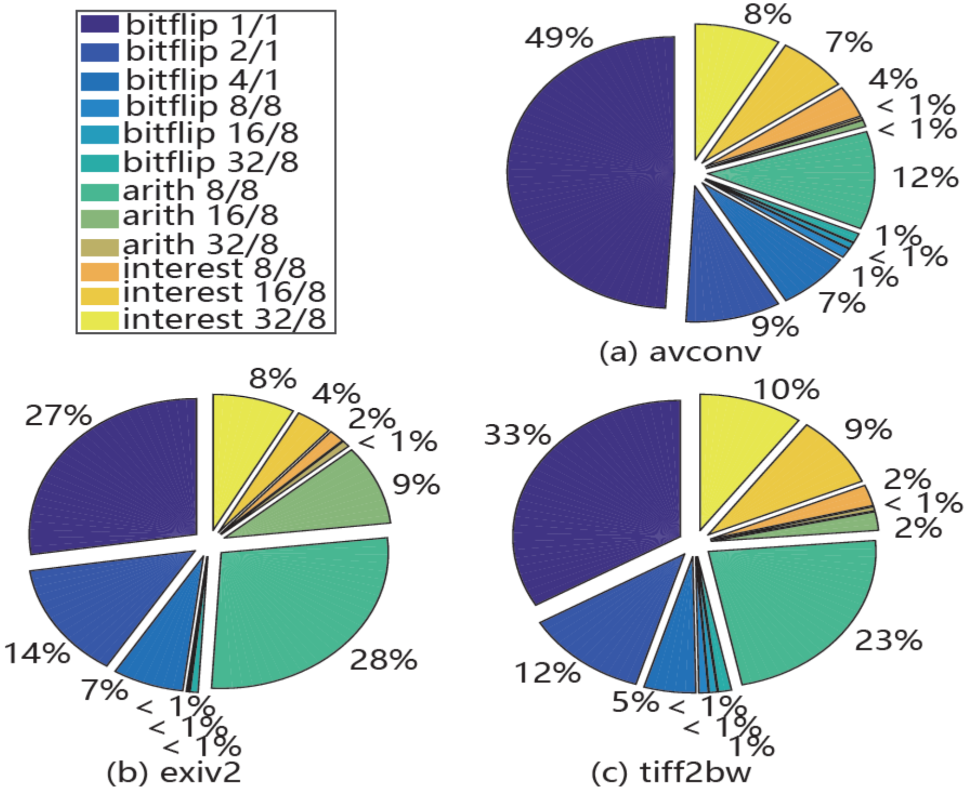
Mutation efficiency study on AFL



Different mutation operators' efficiencies are different.

Percentages of interesting test cases produced by different operators in the deterministic stage of AFL

Mutation efficiency study on AFL

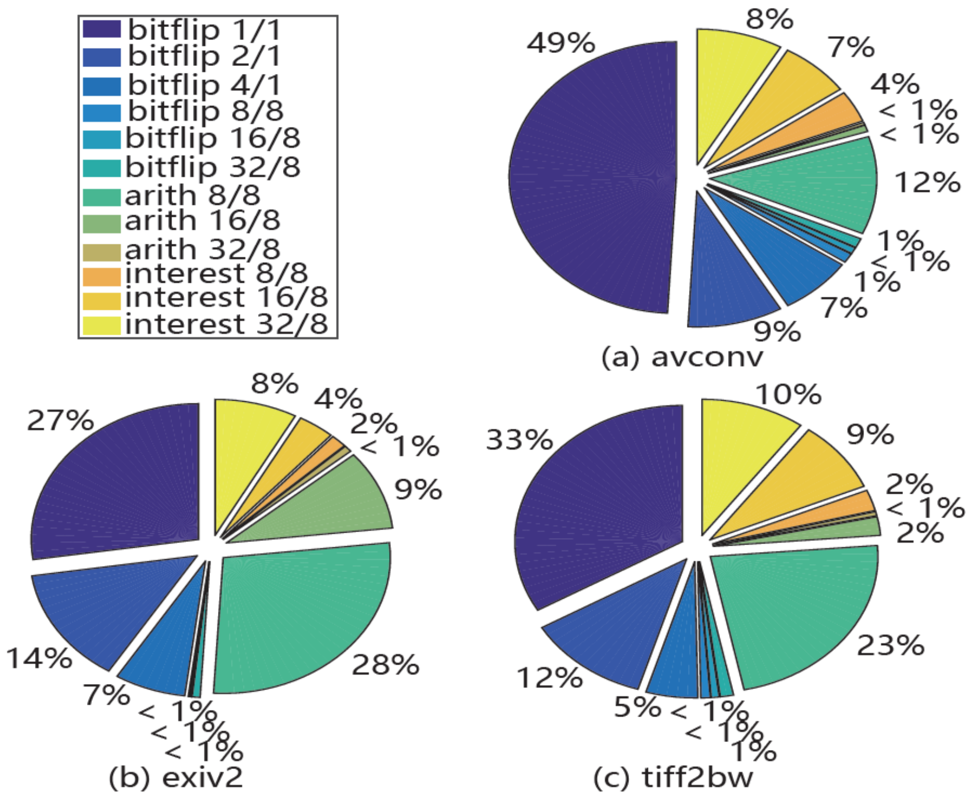


Different mutation operators' efficiencies are different.

For these programs, the mutation operators *bitflip 1/1*, *bitflip 2/1* and *arith 8/8* could yield more interesting test cases than other mutation operators.

Percentages of interesting test cases produced by different operators in the deterministic stage of AFL

Mutation efficiency study on AFL

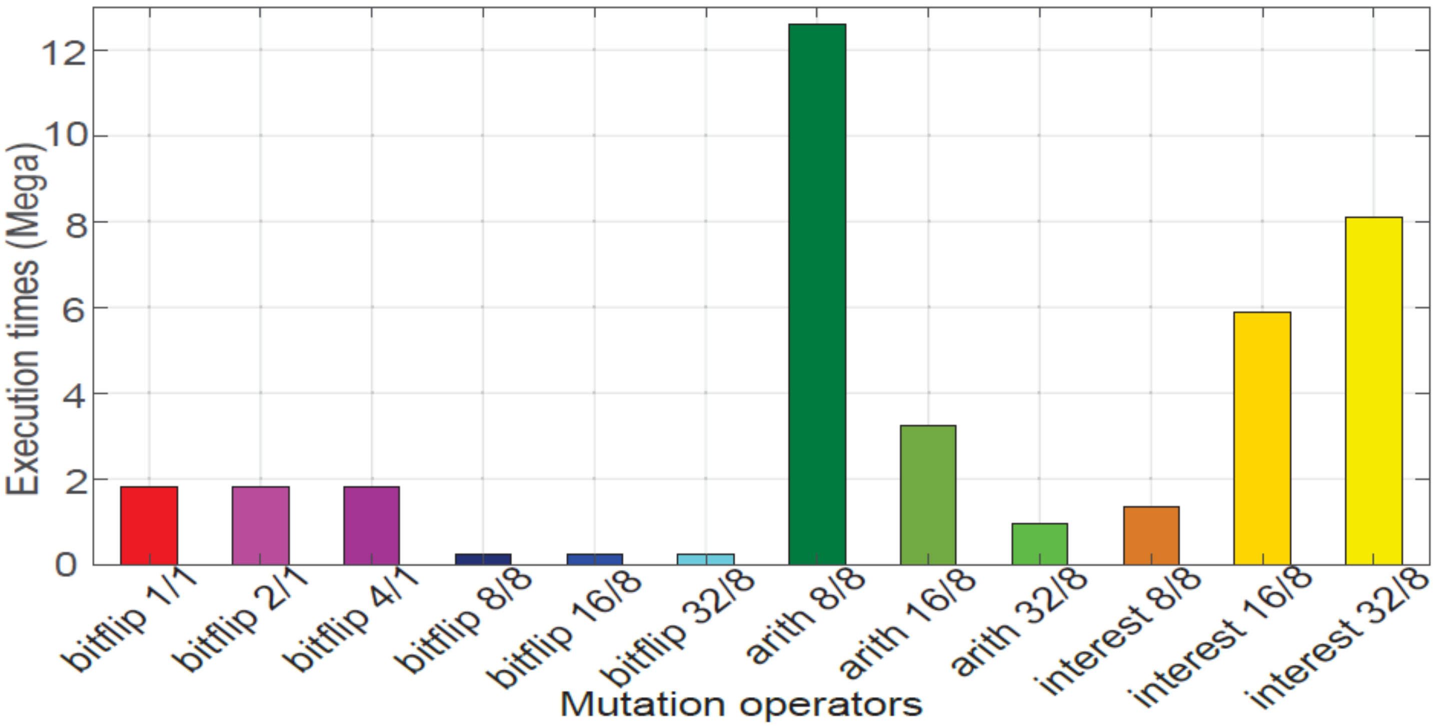


It may be better to select the mutation operators based on this probability distribution (percentages).

Percentages of interesting test cases produced by different operators in the deterministic stage of AFL

How does AFL select these mutation operators?

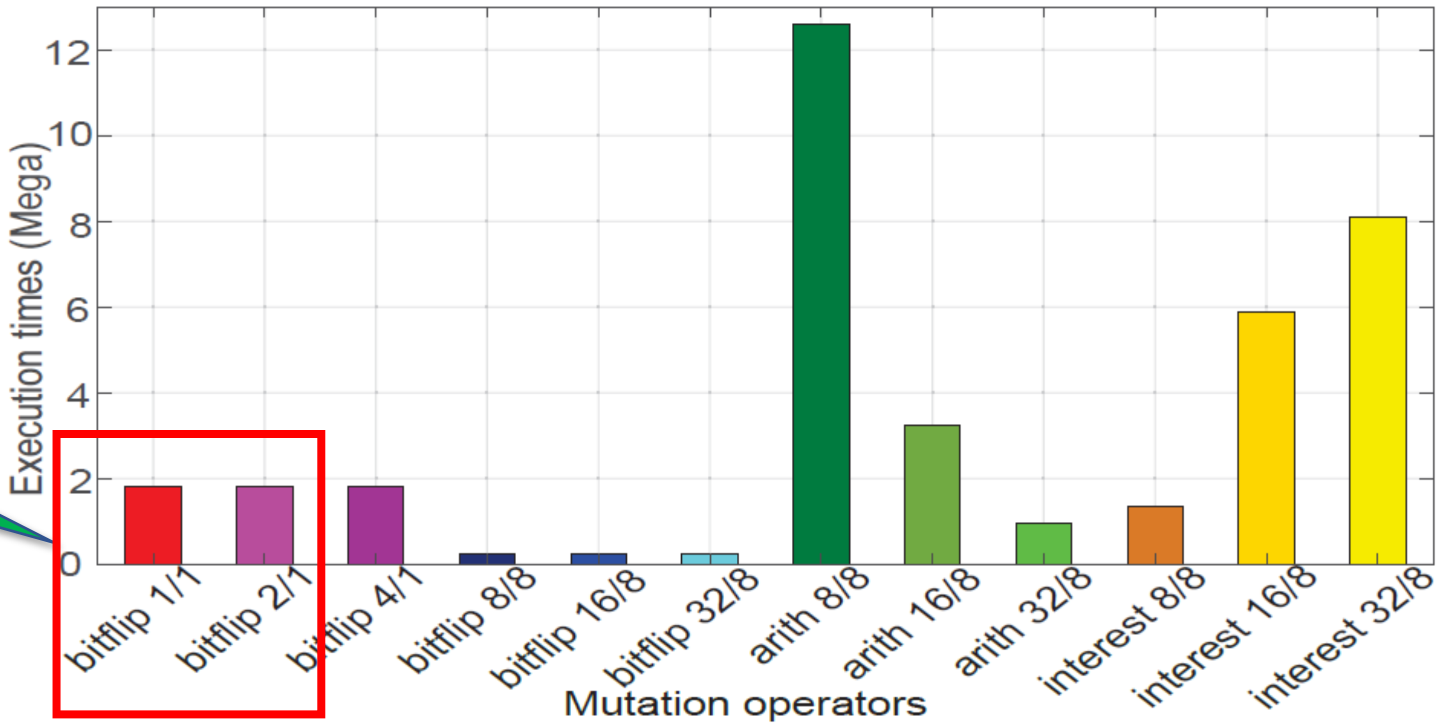
How does AFL select these mutation operators?



The times that mutation operators are selected when AFL fuzzes the target program avconv.

How does AFL select these mutation operators?

The two efficient operators are selected for a small number of times.



The times that mutation operators are selected when AFL fuzzes the target program avconv.

A better mutation scheduling scheme is demanded

A better mutation scheduling scheme is demanded

- In this paper, we simplify the mutation scheduling problem as finding **an optimal probability distribution** of mutation operators, following which the scheduler chooses next operators when testing a target program.

A better mutation scheduling scheme is demanded

- In this paper, we simplify the mutation scheduling problem as finding **an optimal probability distribution** of mutation operators, following which the scheduler chooses next operators when testing a target program.
- Inspired by **Particle Swarm Optimization (PSO)** algorithm, we propose the mutation scheduling scheme **MOPT**.

A better mutation scheduling scheme is demanded

- In this paper, we simplify the mutation scheduling problem as finding **an optimal probability distribution** of mutation operators, following which the scheduler chooses next operators when testing a target program.
- Inspired by **Particle Swarm Optimization (PSO)** algorithm, we propose the mutation scheduling scheme **MOPT**.
- MOPT aims to find the **optimal (selection) probability distribution** of the mutation operators to improve fuzzing.

Particle Swarm Optimization

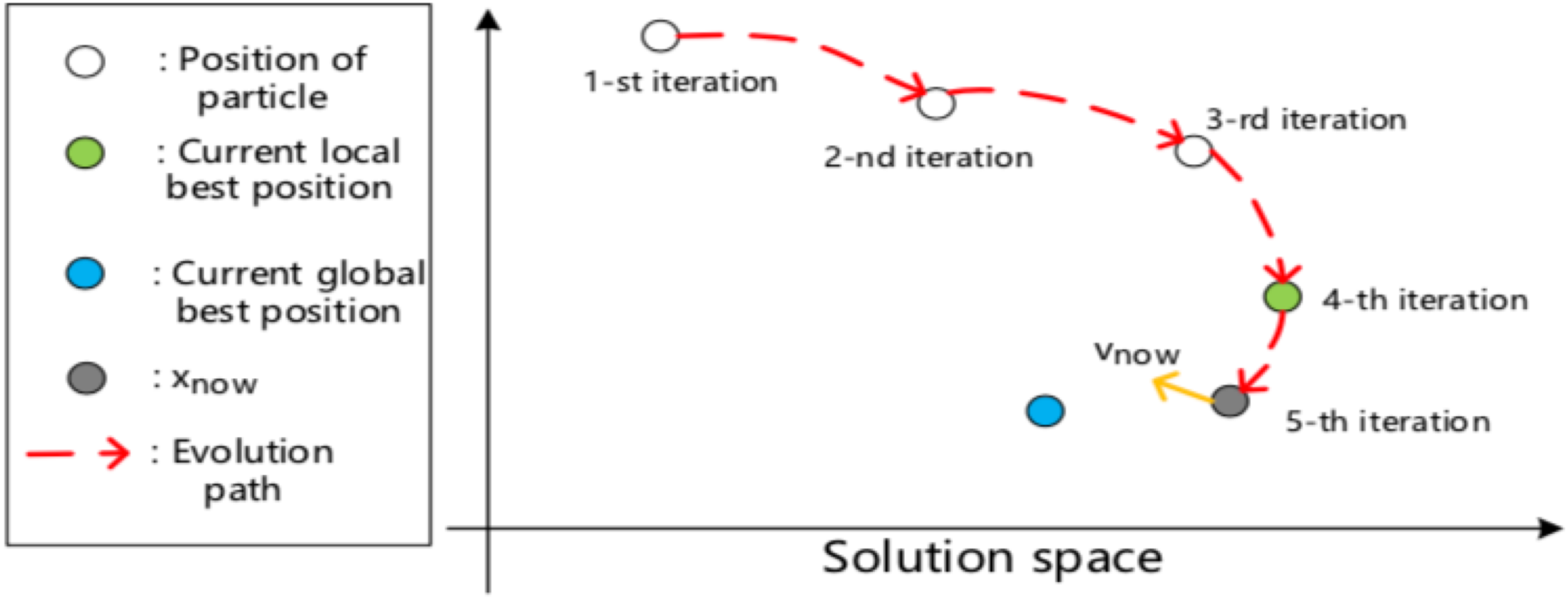
- PSO algorithm employs multiple particles to search the solution space iteratively, in which a position is a candidate solution.

Particle Swarm Optimization

- PSO algorithm employs multiple particles to search the solution space iteratively, in which a position is a candidate solution.
- For each particle p in the swarm, it moves towards its local best position and global best position in each iteration.

Particle Swarm Optimization

- PSO algorithm employs multiple particles to search the solution space iteratively, in which a position is a candidate solution.
- For each particle p in the swarm, it moves towards its local best position and global best position in each iteration.



Particle Swarm Optimization

- For each iteration, the movement of a particle p is updated as follows:

$$V_{now}(\mathrm{p}) \leftarrow w \times V_{now}(\mathrm{p}) + r \times \left(L_{best}(p) \; - \; x_{now}(p)\right) + r \times \left(G_{best} \; - \; x_{now}(p)\right)$$
$$X_{now}(\mathrm{p}) \leftarrow X_{now}(\mathrm{p}) + V_{now}(\mathrm{p})$$

- $V_{now}(\mathrm{p})$ is the velocity of a particle p.
- $X_{now}(\mathrm{p})$ is the position of a particle p.
- $L_{best}(p)$ is the local best position of a particle p.
- G_{best} is the global best position.
- w is the inertia weight.
- $r \in (0,1)$ is a random displacement weight

The customized PSO algorithm of MOPT

- For each iteration, the movement of a particle P_j (mutation operator) in a swarm S_i (a set of mutation operators), its position $X_{now}[S_i]\,[P_j]$ (the probability that it will be selected) is updated by these formula:

$$V_{now}[S_i][P_j] \leftarrow w \times V_{now}[S_i][P_j] + r \times (L_{best}[S_i][P_j] - x_{now}[S_i][P_j]) + r \times (G_{best}[P_j] - x_{now}[S_i]\,[P_j])$$

$$X_{now}[S_i]\,[P_j] \leftarrow X_{now}[S_i]\,[P_j] + [S_i]\,[P_j]$$

- w is the inertia weight.
- $r \in (0,1)$ is a random displacement weigh

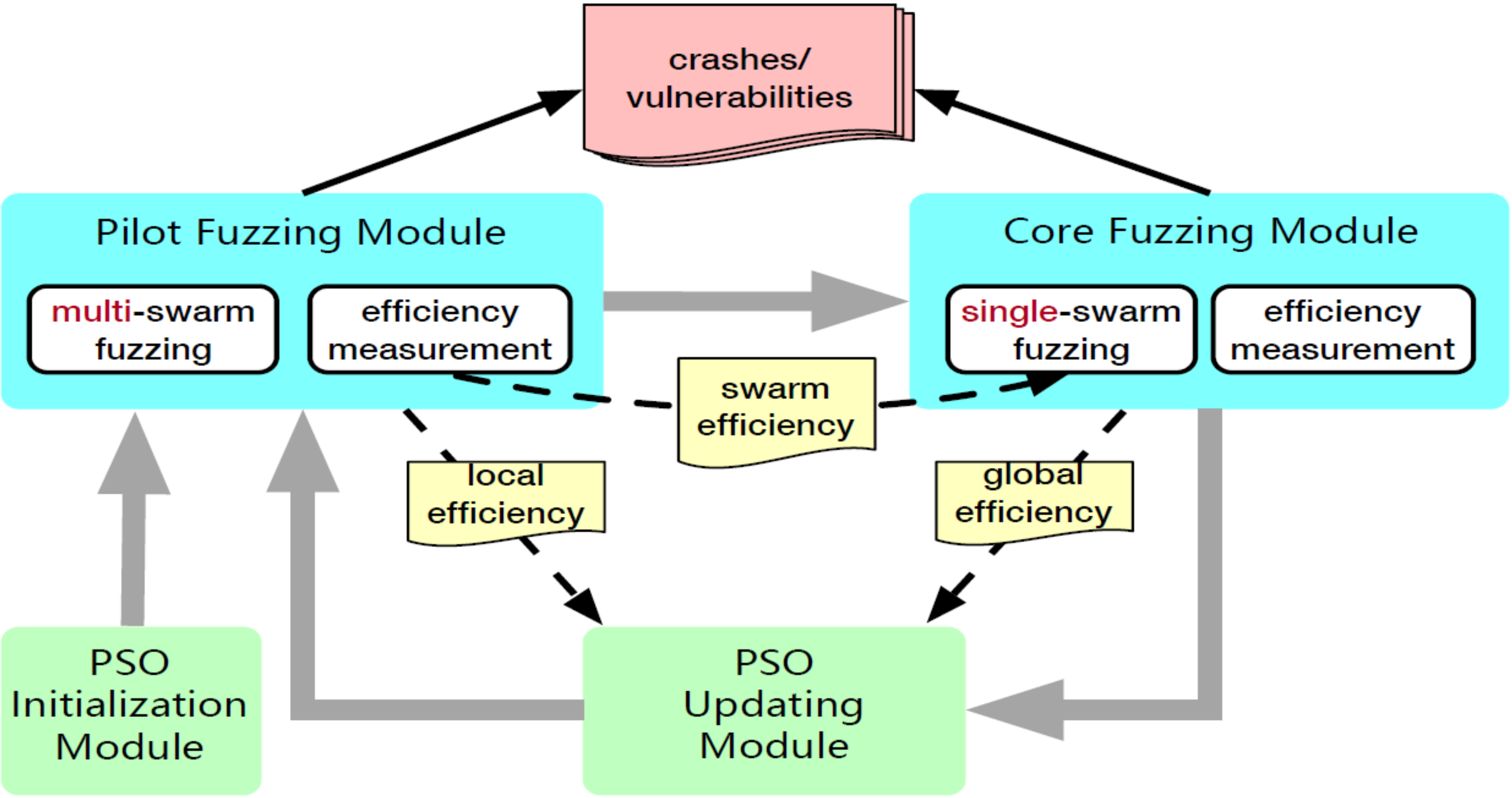
Implementation of MOPT

- MOPT main framework
- Pacemaker fuzzing mode

Implementation of MOPT

- MOPT main framework
- Pacemaker fuzzing mode

MOPT main framework



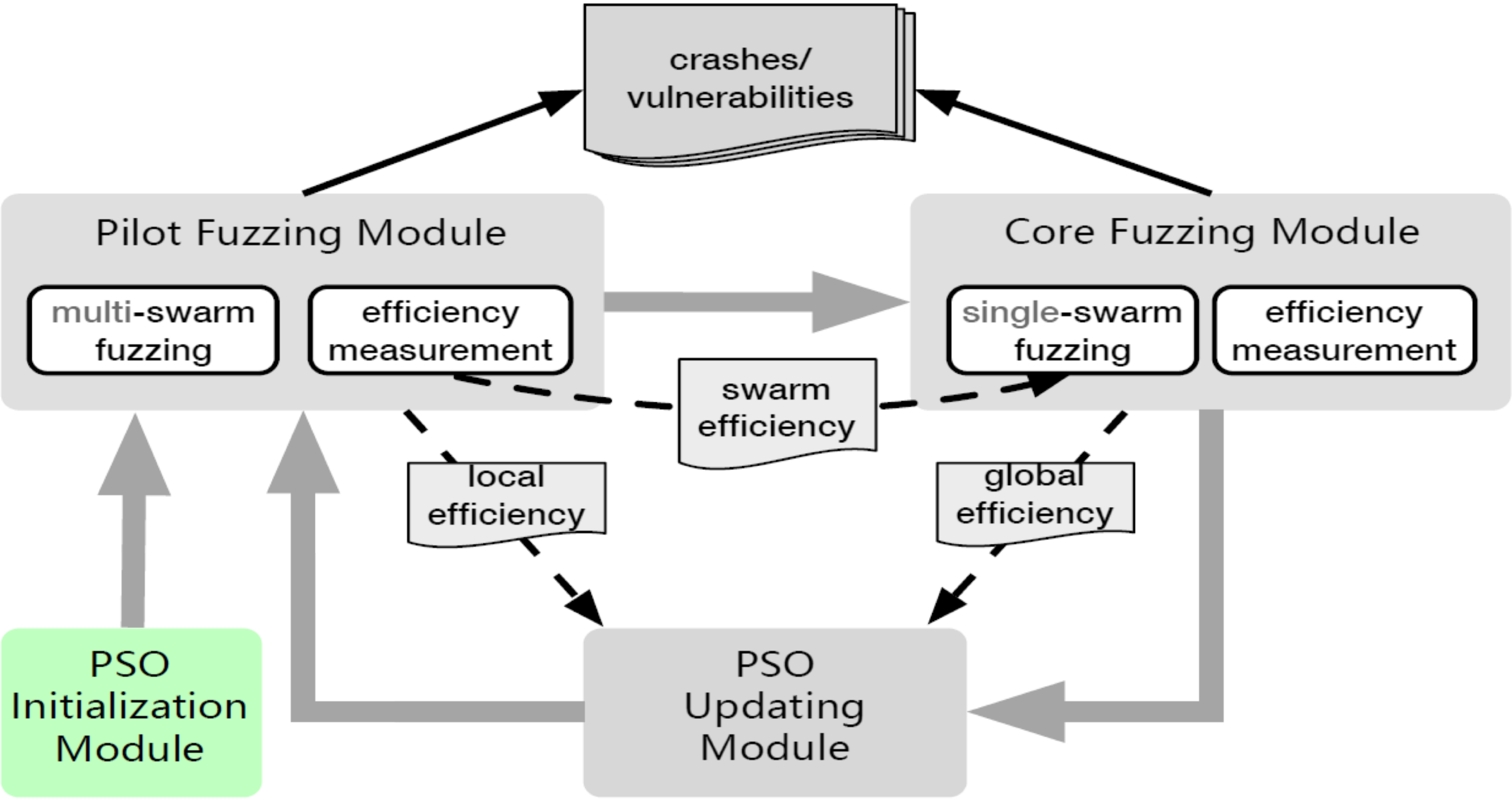
PSO Initialization Module

Pilot Fuzzing Module

Core Fuzzing Module

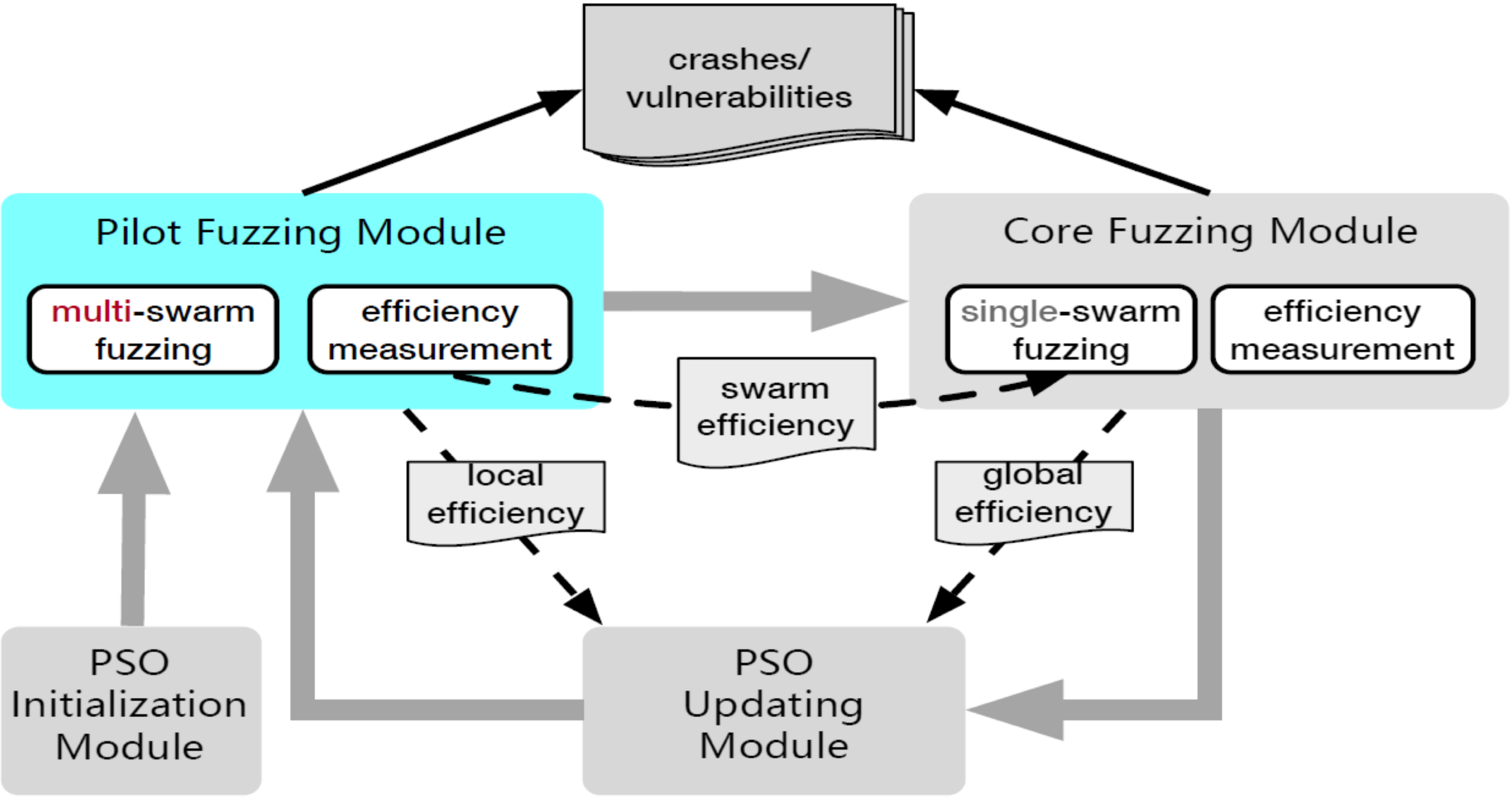
PSO Updating Module

MOPT main framework



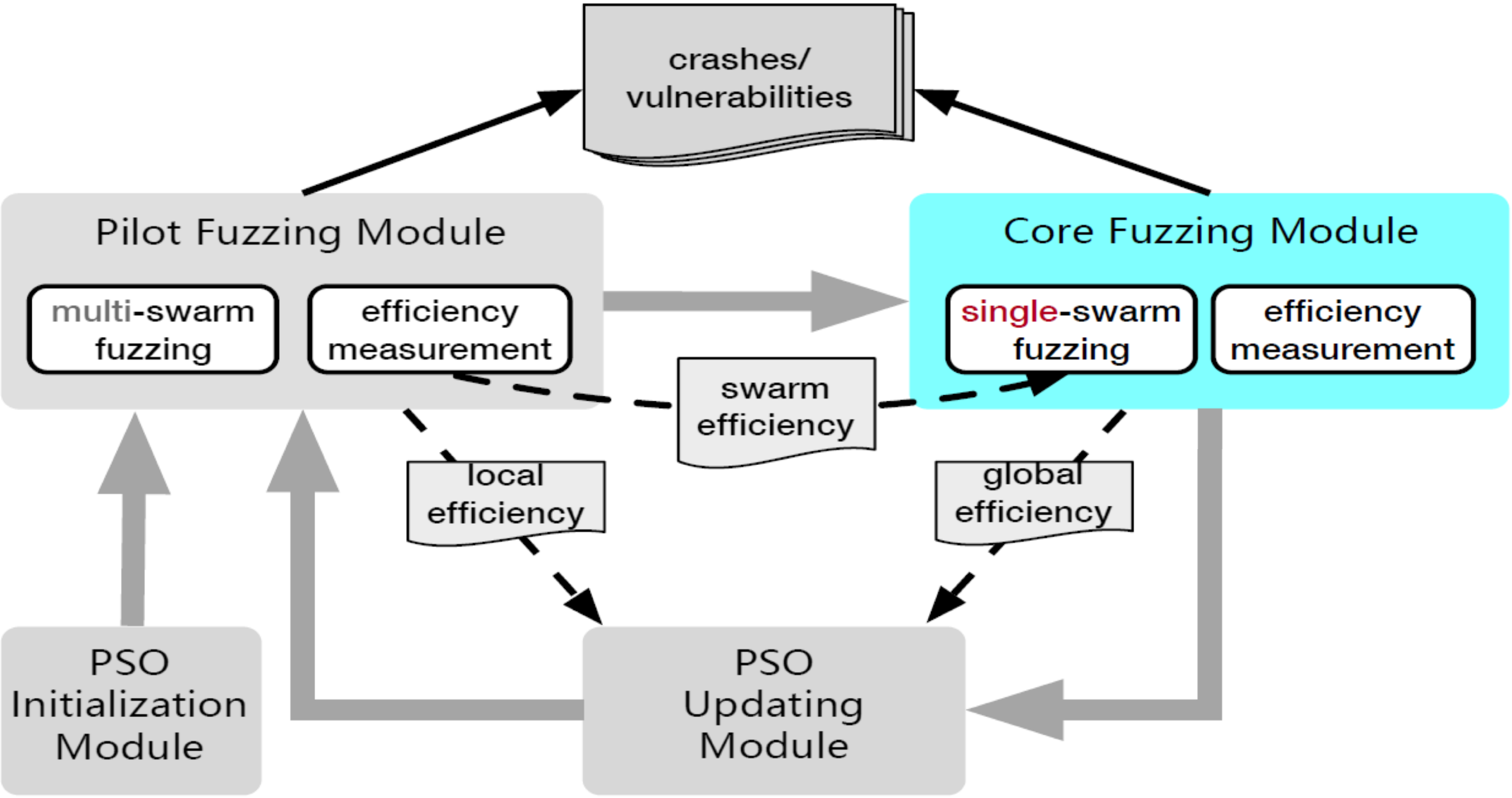
PSO Initialization Module initializes parameters for the customized PSO algorithm.

MOPT main framework



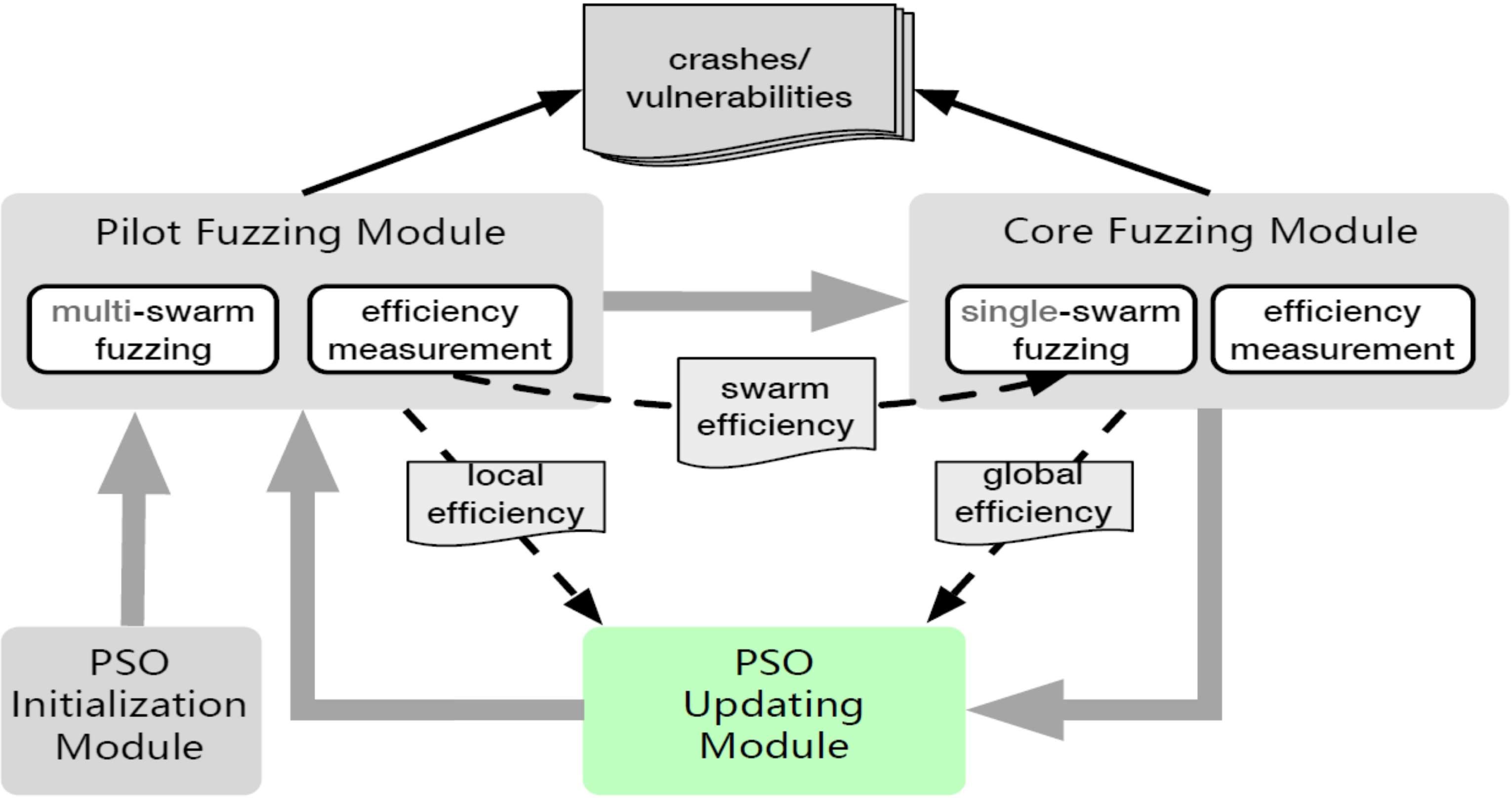
Pilot Fuzzing Module employs the distributions from multiple swarms to perform fuzzing and records the measurements for updating.

MOPT main framework



Core Fuzzing Module employs the best swarm evaluated by *Pilot Fuzzing Module* to perform fuzzing and records the measurements.

MOPT main framework



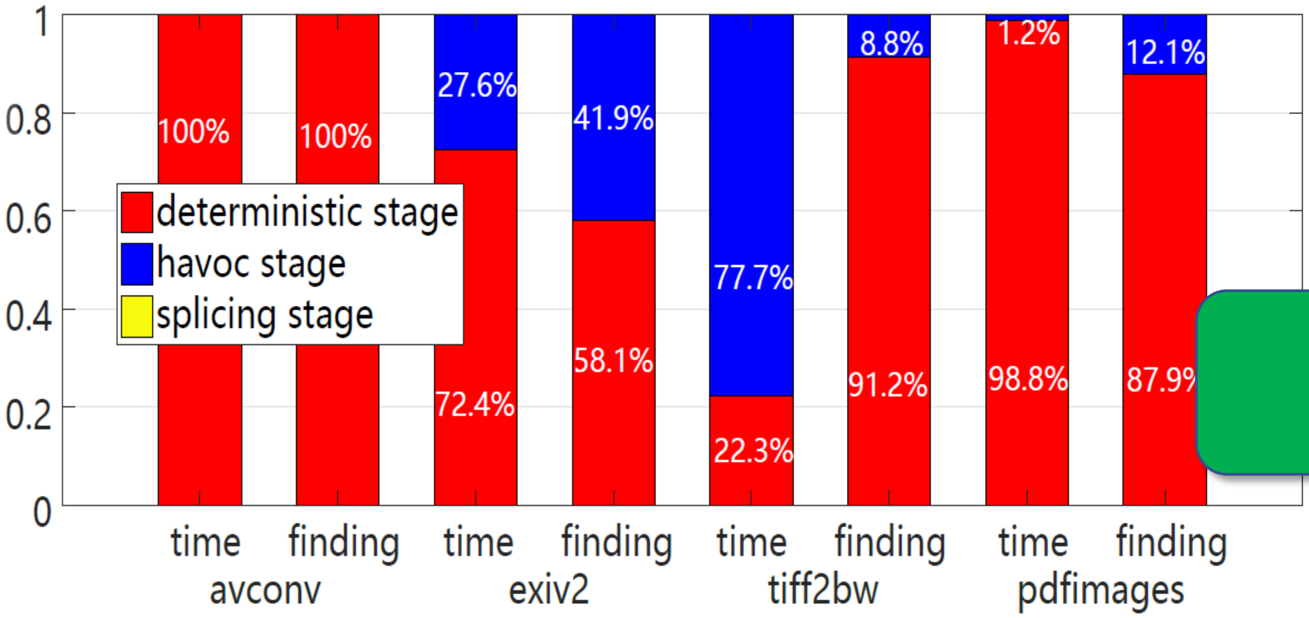
PSO Updating Module updates the distribution of each swarm with the measurements from Pilot Fuzzing and Core Fuzzing Modules.

Implementation of MOPT

- MOPT main framework
- Pacemaker fuzzing mode

Pacemaker fuzzing mode

- In order to selectively avoid the time-consuming **deterministic stage of AFL-based fuzzers,** MOPT provides an optimization, denoted as **pacemaker mode.**



AFL spends most time on the deterministic stage.

Percentages of time and interesting test cases used and found by the three stages in AFL, respectively

Pacemaker fuzzing mode

- In order to selectively avoid the time-consuming deterministic stage, MOPT provides an optimization to AFL-based fuzzers, denoted as pacemaker mode.
- Functionality: if MOPT has not discovered any new unique crash or path for a long time, i.e., ***T*** that is set by users, it will selectively disable the deterministic stage for the following test cases.
- MOPT provides two types of pacemaker fuzzing modes for AFL, based on whether the deterministic stage will be re-enabled (**MOPT-AFL-tmp**) or not (**MOPT-AFL-ever**).

Evaluation

Evaluation

- Evaluate MOPT on 13 real-world programs.
- Evaluation metrics:
  - The number of unique paths
  - The number of unique crashes
  - The number of discovered vulnerabilities
  - The number of discovered CVEs

13 real-world programs

Target	Source file	Input format	Test instruction
mp42aac	Bento4-1-5-1	mp4	mp42aac @@ /dev/null
exiv2	exiv2-0.26-trunk	jpg	exiv2 @@ /dev/null
mp3gain	mp3gain-1_5_2	mp3	mp3gain @@ /dev/null
tiff2bw	libtiff-4.0.9	tiff	tiff2bw @@ /dev/null
pdfimages	xpdf-4.00	PDF	pdfimages @@ /dev/null
sam2p	sam2p-0.49.4	bmp	sam2p @@ EPS: /dev/null
avconv	libav-12.3	mp4	avconv -y -i @@ -f null -
w3m	w3m-0.5.3	text	w3m @@
objdump	binutils-2.30	binary	objdump –dwarf-check -C -g -f -dwarf -x @@
jhead	jhead-3.00	jpg	jhead @@
mpg321	mpg321_0.3.2	mp3	mpg321 -t @@ /dev/null
infotocap	ncurses-6.1	text	infotocap @@
podofopdfinfo	podofo-0.9.6	PDF	podofopdfinfo @@

Evaluation

- Evaluate MOPT on 13 real-world programs.
- Evaluation metrics:
  - The number of unique crashes and paths
  - The number of discovered vulnerabilities
  - The number of discovered CVEs

Evaluation

- Evaluate MOPT on 13 real-world programs.
- Evaluation metrics:
  - The number of unique crashes and paths
  - The number of discovered vulnerabilities
  - The number of discovered CVEs

The number of unique crashes and paths

Program	AFL		MOpt-AFL-tmp				MOpt-AFL-ever			
	Unique crashes	Unique paths	Unique crashes	Increase	Unique paths	Increase	Unique crashes	Increase	Unique paths	Increase
mp42aac	135	815	**209**	+54.8%	1,660	+103.7%	199	+47.4%	**1,730**	+112.3%
exiv2	34	2,195	54	+58.8%	2,980	+35.8%	**66**	+94.1%	**4,642**	+111.5%
mp3gain	178	1,430	**262**	+47.2%	**2,211**	+54.6%	**262**	+47.2%	2,206	+54.3%
tiff2bw	4	4,738	**85**	+2,025.0%	**7,354**	+55.2%	43	+975.0%	7,295	+54.0%
pdfimages	23	12,915	357	+1,452.2%	22,661	+75.5%	**471**	+1,947.8%	**26,669**	+106.5%
sam2p	36	531	105	+191.7%	1,967	+270.4%	**329**	+813.9%	**3,418**	+543.7%
avconv	0	2,478	**4**	+4	**17,359**	+600.5%	1	+1	16,812	+578.5%
w3m	0	3,243	**506**	+506	5,313	+63.8%	182	+182	**5,326**	+64.2%
objdump	0	11,565	**470**	+470	19,309	+67.0%	287	+287	**22,648**	+95.8%
jhead	19	478	55	+189.5%	**489**	+2.3%	**69**	+263.2%	483	+1.0%
mpg321	10	123	**236**	+2,260.0%	1,054	+756.9%	229	+2,190.0%	**1,162**	+844.7%
infotocap	92	3,710	340	+269.6%	6,157	+66.0%	**692**	+652.2%	**7,048**	+90.0%
podofopdfinfo	79	3,397	**122**	+54.4%	**4,704**	+38.5%	114	+44.3%	4,694	+38.2%
total	610	47,618	2,805	+359.8%	93,218	+95.8%	**2,944**	+382.6%	**104,133**	+118.7%

Both MOpt-AFL-tmp and –ever found more unique crashes and paths than AFL.

Evaluation

- Evaluate MOPT on 13 real-world programs.
- Evaluation metrics:
  - The number of unique crashes and paths
  - The number of discovered vulnerabilities
  - The number of discovered CVEs

Vulnerability discovery

Program	AFL				MOPT-AFL-tmp				MOPT-AFL-ever			
	Unknown vulnerabilities		Known vulnerabilities	Sum	Unknown vulnerabilities		Known vulnerabilities	Sum	Unknown vulnerabilities		Known vulnerabilities	Sum
	Not CVE	CVE	CVE		Not CVE	CVE	CVE		Not CVE	CVE	CVE	
mp42aac	/	1	1	2	/	2	1	3	/	5	1	6
exiv2	/	5	3	8	/	5	4	9	/	4	4	8
mp3gain	/	4	2	6	/	9	3	12	/	5	2	7
pdfimages	/	1	0	1	/	12	3	15	/	9	2	11
avconv	/	0	0	0	/	2	0	2	/	1	0	1
w3m	/	0	0	0	/	14	0	14	/	5	0	5
objdump	/	0	0	0	/	1	2	3	/	0	2	2
jhead	/	1	0	1	/	4	0	4	/	5	0	5
mpg321	/	0	1	[illegible]	/	0	1	[illegible]	/	0	1	[illegible]
infotocap	/	3	0	[illegible]	/	3	0	[illegible]	/	3	0	[illegible]
podofopdfinfo	/	5	0	[illegible]	/	6	0	[illegible]	/	6	0	[illegible]
tiff2bw	1	/	/	[illegible]	2	/	/	[illegible]	2	/	/	[illegible]
sam2p	5	/	/	5	14	/	/	14	28	/	/	28
Total	6	20	7	33	16	58	14	88	30	43	12	85

33

88

85

Vulnerabilities discovered by AFL, MOPT-AFL-tmp, MOPT-AFL-ever

Both MOPT-AFL-tmp and –ever found much more vulnerabilities than AFL.

Evaluation

- Evaluate MOPT on 13 real-world programs.
- Evaluation metrics:
  - The number of unique crashes and paths
  - The number of discovered vulnerabilities
  - The number of discovered CVEs

CVE discovery

Target	Types	AFL	MOpt-AFL-tmp	MOpt-AFL-ever	Severity
mp42aac	buffer overflow	CVE-2018-10785	CVE-2018-10785; CVE-2018-18037	CVE-2018-10785; CVE-2018-18037; CVE-2018-17814	4.3
	memory leaks	CVE-2018-17813	CVE-2018-17813	CVE-2018-17813; CVE-2018-18050; CVE-2018-18051	4.3
exiv2	heap overflow	CVE-2017-11339; CVE-2017-17723; CVE-2018-18036	CVE-2017-11339; CVE-2017-17723; CVE-2018-10780	CVE-2017-11339; CVE-2017-17723; CVE-2018-18036	5.8
	stack overflow	CVE-2017-14861	CVE-2017-14861	CVE-2017-14861	4.3
	buffer overflow	CVE-2018-18047	CVE-2018-17808; CVE-2018-18047	CVE-2018-18047	4.3
	segmentation violation	CVE-2018-18046	CVE-2018-18046	CVE-2018-18046	4.3
	memory access violation	CVE-2018-17809; CVE-2018-17807	CVE-2018-17809; CVE-2018-17823	CVE-2017-11337; CVE-2018-17809	4.3
mp3gain	stack buffer overflow	CVE-2017-14407	CVE-2017-14407; CVE-2018-17801; CVE-2018-17799	CVE-2017-14407	4.3
	global buffer overflow	CVE-2018-17800; CVE-2018-17802; CVE-2018-18045; CVE-2018-18043	CVE-2017-14409; CVE-2018-17800; CVE-2018-17803; CVE-2018-17802; CVE-2018-18045; CVE-2018-18043; CVE-2018-18044	CVE-2018-17800; CVE-2018-17803; CVE-2018-17802; CVE-2018-18045; CVE-2018-18043	6.8
	segmentation violation	CVE-2017-14406	CVE-2017-14412	CVE-2017-14412	6.8
	memcpy param overlap		CVE-2018-17824		5.8
pdfimages	heap buffer overflow		CVE-2018-8103; CVE-2018-18054		4.3
	stack overflow	CVE-2018-17114	CVE-2018-16369; CVE-2018-17114; CVE-2018-17115; CVE-2018-17116; CVE-2018-17117; CVE-2018-17119; CVE-2018-17120; CVE-2018-17121; CVE-2018-17122; CVE-2018-18053; CVE-2018-18055	CVE-2018-16369; CVE-2018-17115; CVE-2018-17116; CVE-2018-17119; CVE-2018-17121; CVE-2018-17122; CVE-2018-18053	6.1
	global buffer overflow		CVE-2018-8102	CVE-2018-8102	4.3
	alloc dealloc mismatch		CVE-2018-17118	CVE-2018-17118	4.3
	segmentation violation			CVE-2018-17123; CVE-2018-17124	4.3
avconv	segmentation violation		CVE-2018-17804	CVE-2018-17804	4.3
	memory leaks		CVE-2018-17805		4.3
w3m	segmentation violation		CVE-2018-17815; CVE-2018-17816; CVE-2018-17817; CVE-2018-17818; CVE-2018-17819; CVE-2018-17821; CVE-2018-17822; CVE-2018-18038; CVE-2018-18039; CVE-2018-18040; CVE-2018-18041; CVE-2018-18042; CVE-2018-18052	CVE-2018-17816; CVE-2018-18040; CVE-2018-18041; CVE-2018-18042	5.3
	memory leaks		CVE-2018-17820	CVE-2018-17820	4.3
objdump	stack exhaustion		CVE-2018-12700	CVE-2018-12641	5.0
	stack overflow		CVE-2018-9138; CVE-2018-16617	CVE-2018-9138	4.3
jhead	heap buffer overflow	CVE-2018-17810	CVE-2018-17810; CVE-2018-17811; CVE-2018-18048; CVE-2018-18049	CVE-2018-17810; CVE-2018-17811; CVE-2018-17812; CVE-2018-18048; CVE-2018-18049	4.3
mpg321	heap buffer overflow	CVE-2017-12063	CVE-2017-12063	CVE-2017-12063	4.3
infotocap	memory leaks	CVE-2018-16614	CVE-2018-16614	CVE-2018-16614	4.3
	segmentation violation	CVE-2018-16615; CVE-2018-16616	CVE-2018-16615; CVE-2018-16616	CVE-2018-16615; CVE-2018-16616	4.3
podofopdfinfo	stack overflow	CVE-2018-18216; CVE-2018-18221; CVE-2018-18222	CVE-2018-18216; CVE-2018-18217; CVE-2018-18221; CVE-2018-18222	CVE-2018-18216; CVE-2018-18217; CVE-2018-18218; CVE-2018-18221	4.7
	heap buffer overflow	CVE-2018-18219	CVE-2018-18219	CVE-2018-18219	4.3
	segmentation violation	CVE-2018-18220	CVE-2018-18220	CVE-2018-18220	4.3

Both MOpt-AFL-tmp and –ever found more CVEs with a variety of types than AFL.

Compatibility analysis

Compatibility analysis

- MOPT is not limited to AFL!

Compatibility analysis

- MOPT is not limited to AFL!
- The workflow of MOPT can be implemented on many mutation-based fuzzers.
- We combine MOPT scheme with AFLFast and VUzzer to implement **MOPT-AFLFast-tmp**, **MOPT-AFLFast-ever** and **MOPT-VUzzer**

Compatibility analysis

The compatibility of the MOPT scheme.

		mp42aac	exiv2	mp3gain	tiff2bw	pdfimages	sam2p	mpg321
AFL	Unique crashes	135	34	178	4	23	36	10
	Unique paths	815	2,195	1,430	4,738	12,915	531	123
MOPT-AFL-tmp	Unique crashes	**209**	54	**262**	**85**	357	105	**236**
	Unique paths	1,660	2,980	**2,211**	**7,354**	22,661	1,967	1,054
MOPT-AFL-ever	Unique crashes	199	**66**	**262**	43	**471**	**329**	229
	Unique paths	**1,730**	**4,642**	2,206	7,295	**26,669**	**3,418**	**1,162**
AFLFast	Unique crashes	210	0	171	0	18	37	8
	Unique paths	1,233	159	1,383	5,114	12,022	603	122
MOPT-AFLFast-tmp	Unique crashes	**393**	51	**264**	5	292	**196**	**230**
	Unique paths	**3,389**	2,675	2,017	7,012	24,164	2,587	**1,208**
MOPT-AFLFast-ever	Unique crashes	384	**58**	259	**18**	**345**	114	30
	Unique paths	2,951	**2,887**	**2,102**	**7,642**	**26,799**	**2,623**	160
VUzzer	Unique crashes	12	0	54,500	0	0	13	3,598
	Unique paths	12%	9%	50%	13%	25%	18%	18%
MOPT-VUzzer	Unique crashes	**16**	0	**56,109**	0	0	**16**	**3,615**
	Unique paths	12%	9%	**51%**	13%	25%	18%	18%

MOPT is easily compatible with state-of-the-art mutation-based fuzzers to improve their fuzzing performance.

Further analysis

- Statistical experiments with different seed sets
- Iteration analysis of selection probability in MOPT
- Long term parallel analysis

Further analysis

- Statistical experiments with different seed sets
- Iteration analysis of selection probability in MOPT
- Long term parallel analysis

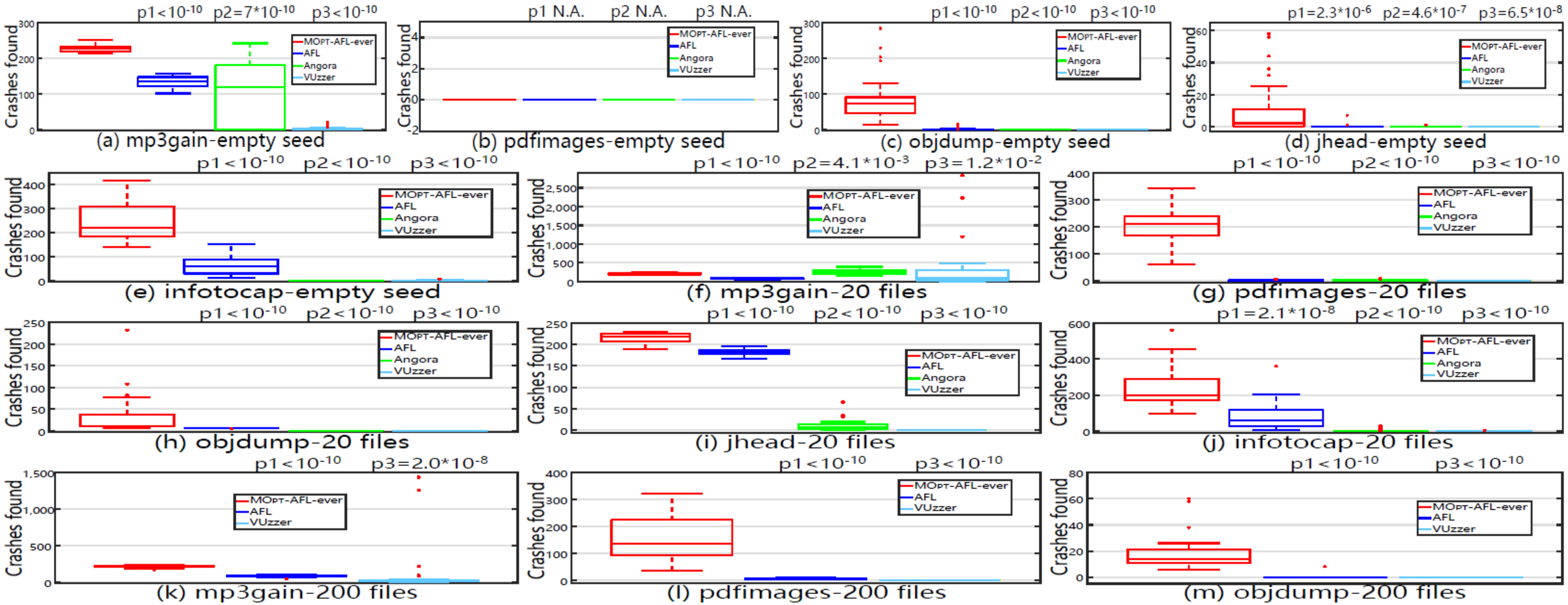
Statistical experiments with different seed sets

- Experiments: evaluate **MOPT-AFL-ever**, **AFL**, **Angora** and **VUzzer** on 5 programs.

- To eliminate the effect of randomness, we run each experiment for 30 trials, 10 days.

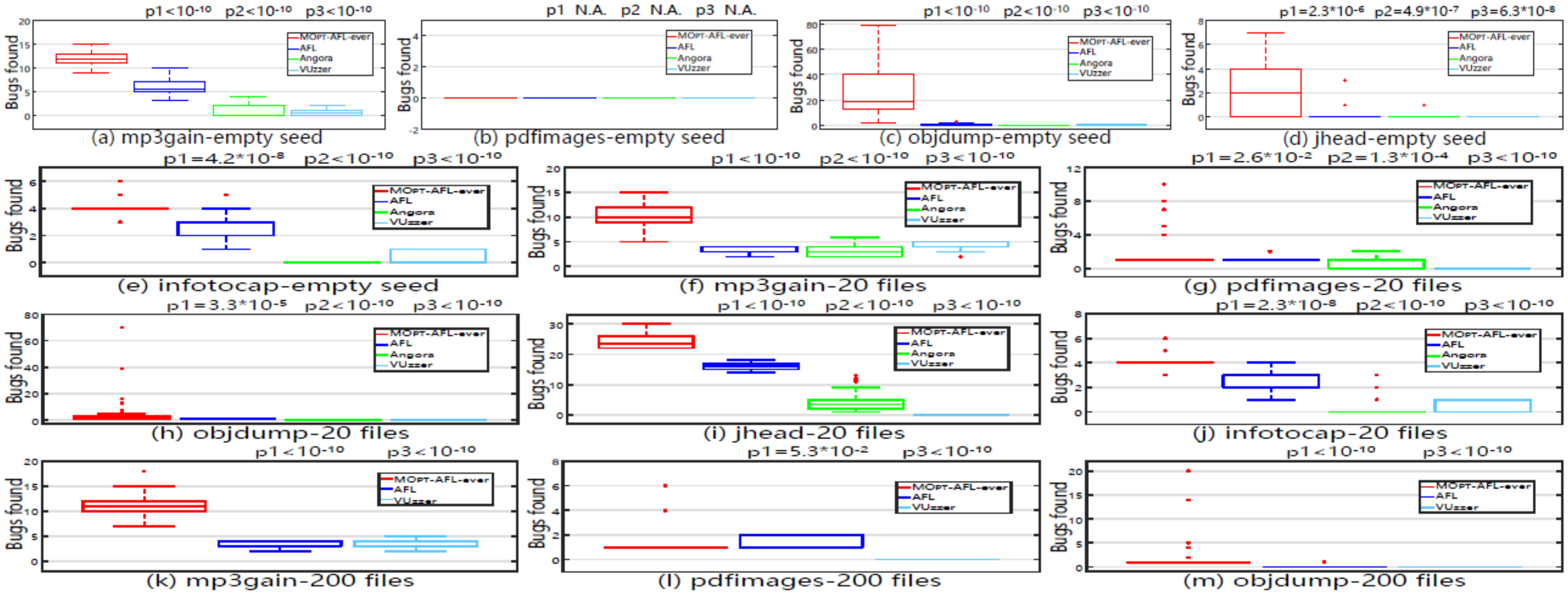
- To eliminate the effect of different seeds, we run each experiment with different seed sets:
    - an empty seed
    - 20 seeds
    - 200 seeds

The number of unique crashes in 30 trials



MOpt–AFL-ever found more unique crashes with the statistical evidence than other fuzzers in most evaluations.

The number of unique bugs in 30 trials

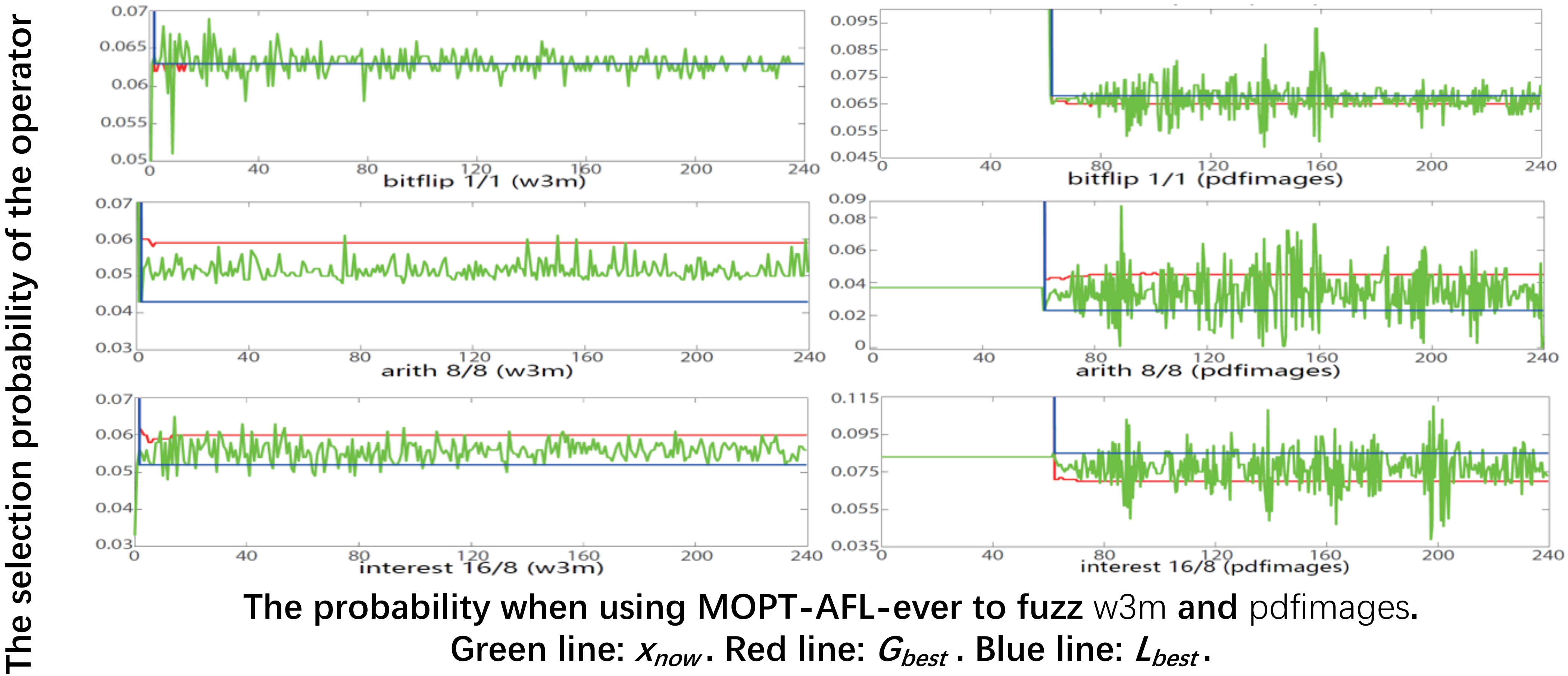


MOPT–AFL-ever found more unique bugs with the statistical evidence than other fuzzers in most evaluations.

Further analysis

- Statistical experiments with different seed sets
- Iteration analysis of selection probability in MOPT
- Long term parallel analysis

Iteration analysis of selection probability in MOPT



The probability when using MOPT-AFL-ever to fuzz w3m **and** pdfimages**.**
Green line: x_{now}. Red line: G_{best}. Blue line: L_{best}.

The MOPT scheme generally converges fast to the proper selection probability.

Further analysis

- Statistical experiments with different seed sets
- Iteration analysis of selection probability in MOPT
- Long term parallel analysis

Long term parallel analysis

- Run three instances of **AFL**, **MOPT-AFL-tmp** and **MOPT-AFL-ever** in parallel, respectively.
- The total CPU time of each experiment is more than 70 days.

		Fuzzer1	Fuzzer2	Fuzzer3	Total
AFL	Unique crashes	11	871	896	1,778
	Unique paths	24,763	29,329	29,329	83,421
MOPT-AFL-tmp	Unique crashes	834	1,031	1,042	2,907
	Unique paths	30,098	31,600	31,520	93,218
MOPT-AFL-ever	Unique crashes	723	974	1,005	2,702
	Unique paths	28,047	30,910	30,966	89,923

Both MOPT-AFL-tmp and –ever found more unique crashes and paths than AFL in the long term parallel experiments.

Limitation and future work

- Extension to more fuzzers
  - MOPT can be easily adapted for most mutation-based fuzzers
- Large-scale evaluation
  - Use more real-world programs and benchmarks to evaluate MOPT
- Better mutation operators
  - Investigate better mutation operators to further enhance the effectiveness of MOPT
- Investigate more mutation scheduling scheme

Conclusion

- We investigated the drawbacks of existing mutation schedulers.
- We proposed a novel mutation scheduling scheme named MOPT.
- We applied MOPT to several state-of-the-art fuzzers and evaluated them with the extensive experiments to demonstrate the high efficiency, compatibility and steadiness of MOPT.

https://github.com/puppet-meteor/MOpt-AFL

MOPT is open sourced!

Thank you!

Zhejiang University, NESA Lab

https://nesa.zju.edu.cn

MOPT: https://github.com/puppet-meteor/MOpt-AFL

Local and global best positions

$$V_{now}[S_i][P_j] \leftarrow w \times V_{now}[S_i][P_j] + r \times (L_{best}[S_i][P_j] - x_{now}[S_i][P_j]) + r \times (G_{best}[P_j] - x_{now}[S_i][P_j])$$

$$X_{now}[S_i][P_j] \leftarrow X_{now}[S_i][P_j] + [S_i][P_j]$$

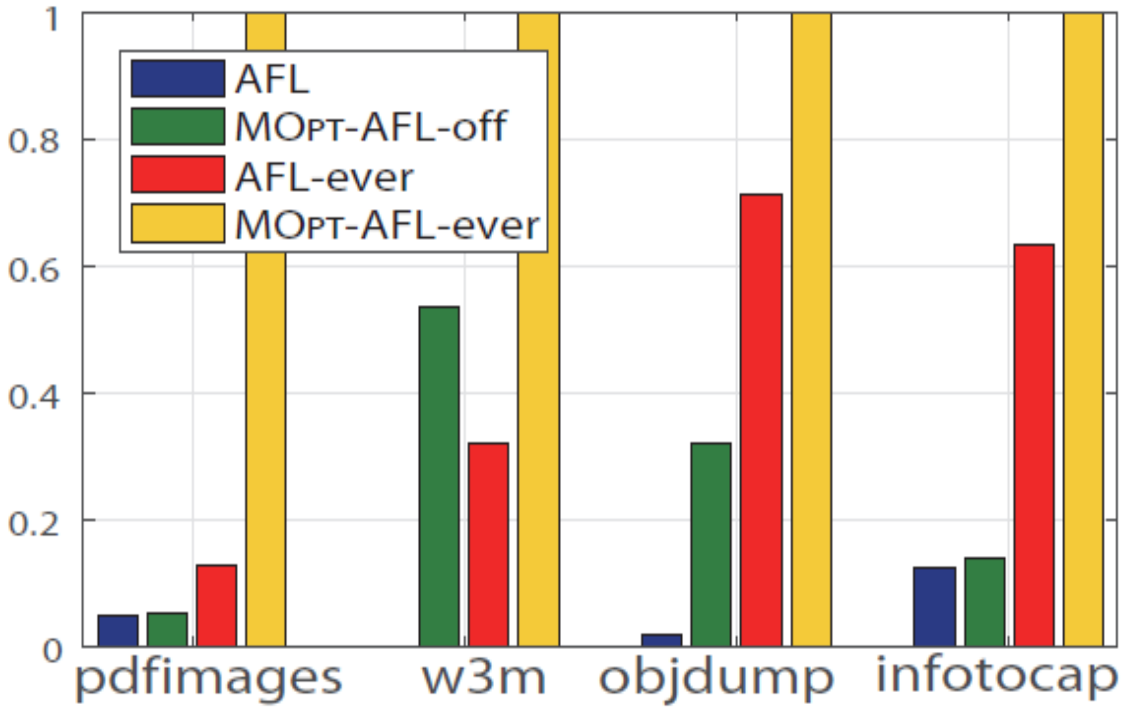
Local best position L_{best}

- L_{best} is the position of the particle where the corresponding operator yields the most interesting test cases (given the same amount of invocations).
- For each particle, we measure its local efficiency eff_{now} (the number of interesting test cases contributed by this operator divided by the number of invocations of this operator during one iteration).
- L_{best} is the position where the operator obtains highest eff_{now} in history.

Pacemaker fuzzing mode

Stepwise analysis

- **Additional fuzzers:**
  - **AFL-ever**: AFL only implementing the pacemaker fuzzing mode
  - **MOPT-AFL-off:** MOPT-AFL-ever while disable the pacemaker fuzzing mode



The ratio of the unique crashes discovered by 4 fuzzers, with MOPT-AFL-ever as the baseline.

- ➢ **Both the MOPT main framework and pacemaker fuzzing mode can improve fuzzing performance.**
- ➢ **The combination of both parts would result in an even better performance.**