Evaluating and Improving Hybrid Fuzzing

Ling Jiang, Hengchen Yuan, Mingyuan Wu,
Lingming Zhang, Yuqun Zhang

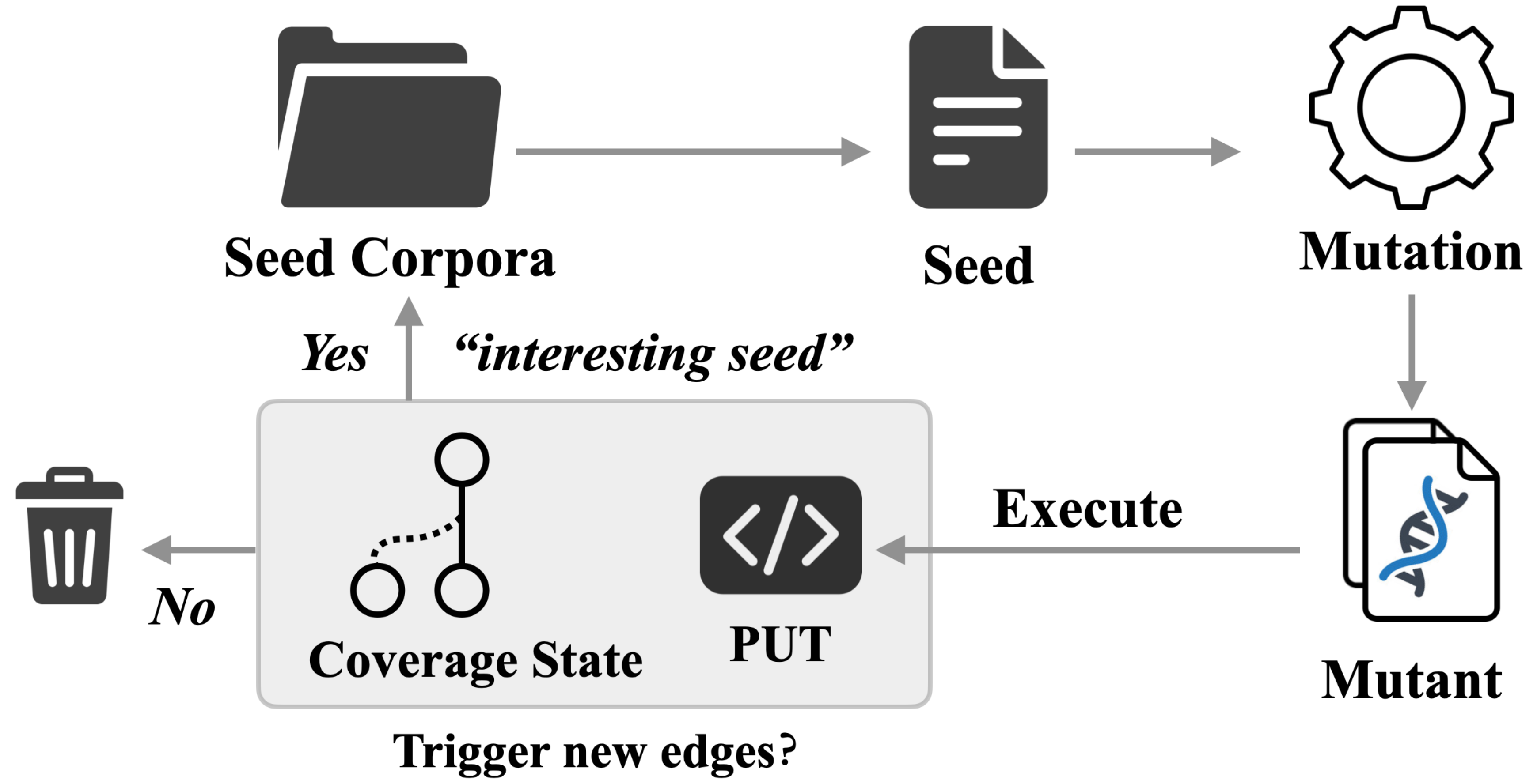
SUSTech Southern University of Science and Technology

University of Illinois Urbana-Champaign

Hybrid Fuzzing = Fuzzing + Concolic Execution

Fuzzing

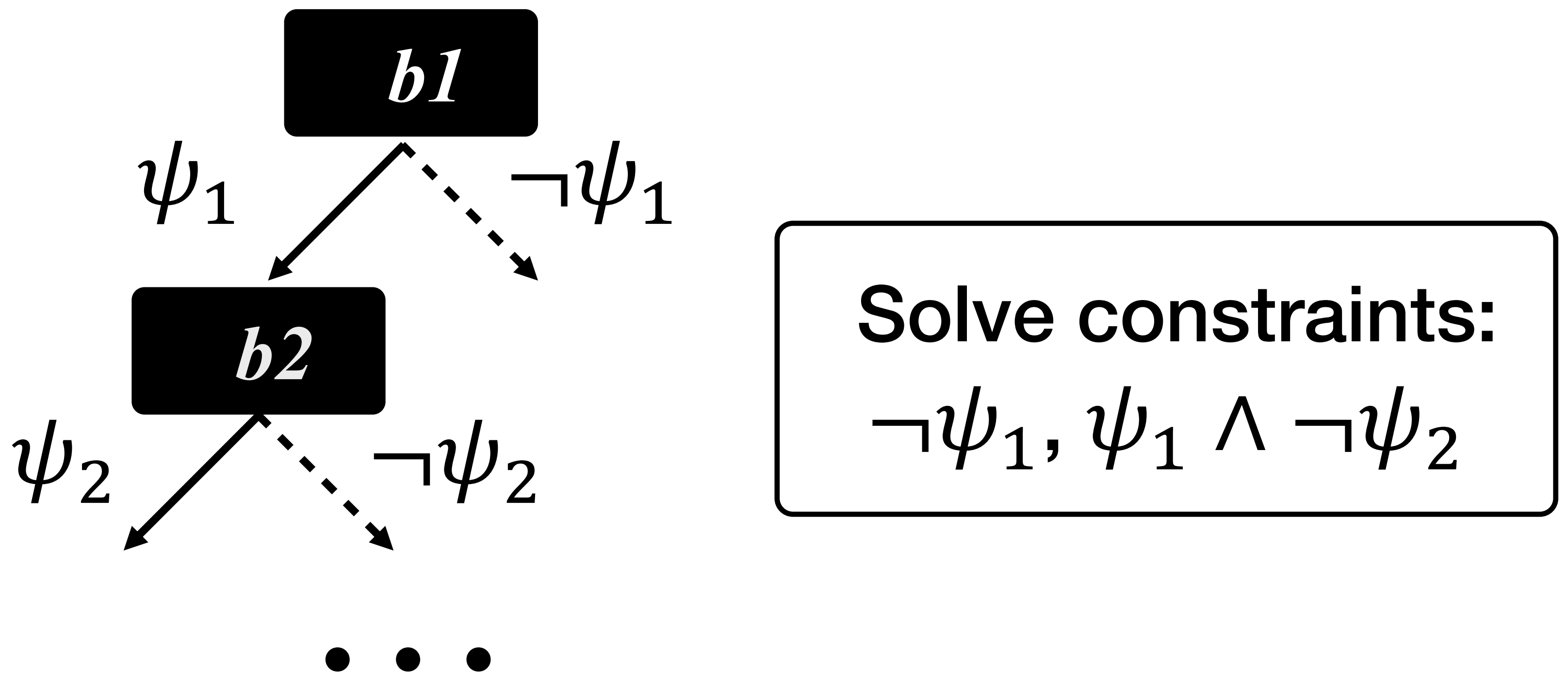
- Light-weight automated software testing technique by generating random test inputs
- Promptly explore the program states, but get stuck by hard-to-cover branches



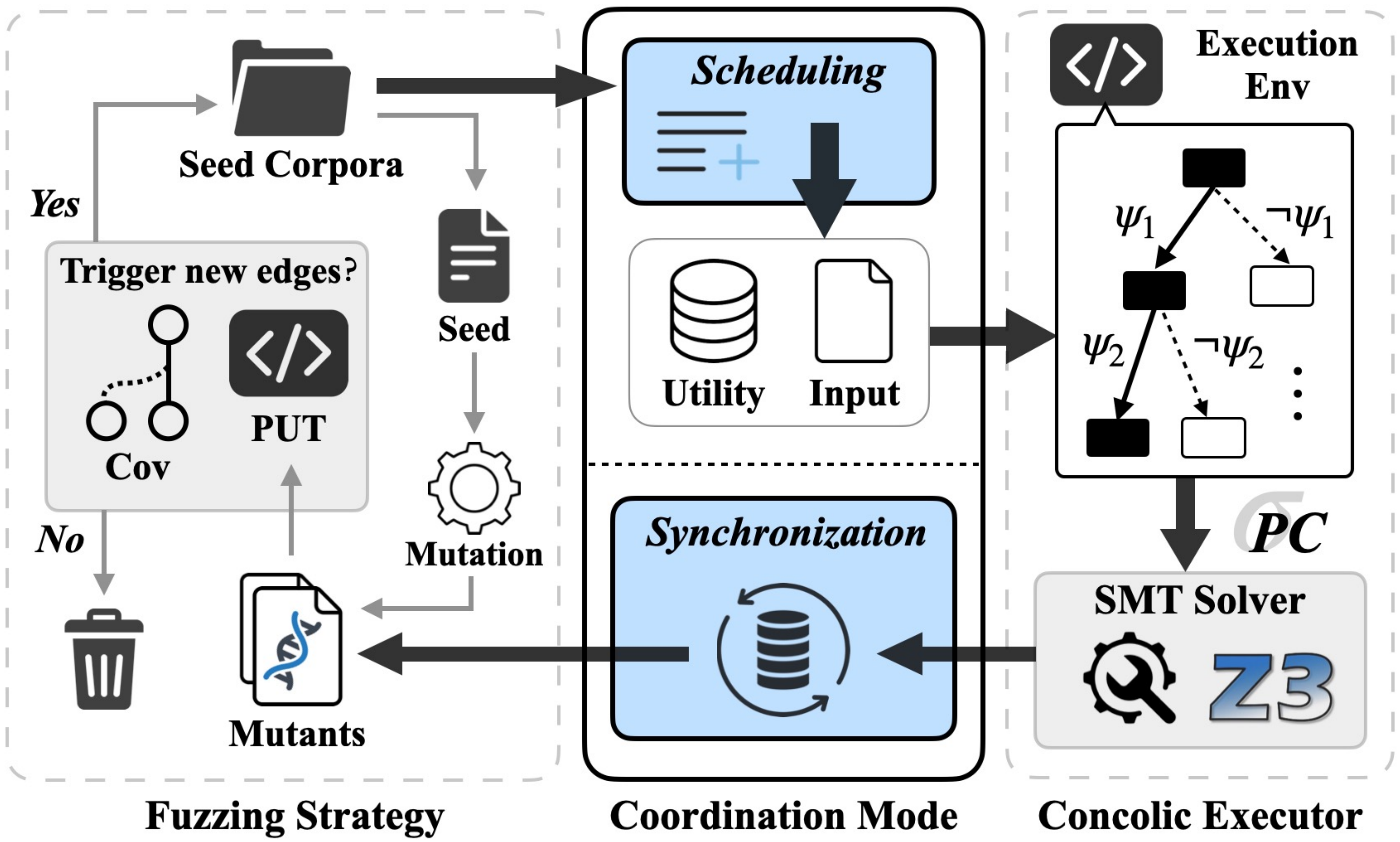
Concolic Execution

(a.k.a. dynamic symbolic execution)

- Generate test inputs by solving the corresponding path constraints via SMT solver
- Significant overhead for symbolic emulation and constraint solving



Hybrid Fuzzing: Three Components



The overall framework of hybrid fuzzing

- ***Coverage-guided Fuzzing Strategy***
- ***Concolic Execution (ce)***
- ***Coordination Mode***

Scheduling

Selecting subjects for performing concolic execution

Synchronization

Synchronizing the solutions to fuzzing to further explore new program states

Scheduling (*seed-oriented*)

- **Baseline** - Random seed selection
- **DigFuzz (NDSS'19)** - Monte Carlo-based probabilistic path prioritization model
- **MEUZZ (RAID'20)** - Machine learning-based regression model

Synchronization

- **Baseline** - Directly adding the solutions to the seed corpora of fuzzing
- **Pangolin (S&P'20)** - Convert solutions to "polyhedral abstraction domain" and generate mutants by Dikin walk algorithm

Challenges

- Inequivalent evaluation setups (benchmarks & seed corpora) of existing hybrid fuzzers
- Limited performance comparisons among existing hybrid fuzzers
- Lack of insight into the impact of different components on hybrid fuzzing

RQ1: *How do hybrid fuzzers perform on top of our benchmark programs?*

- **Performance comparisons among existing hybrid fuzzers**
- **Impact of fuzzing strategy and concolic execution**

RQ2: *How do existing coordination modes impact hybrid fuzzers?*

- **Impact of coordination mode**

Studied Subjects

- Hybrid fuzzers published in prestigious conferences
- Conventional coverage-guided fuzzers for comparison (AFL, FairFuzz, and AFL++)

Name	Fuzzing Strategy	Concolic Executor	Coordination Mode	
			Sch†	Sync‡
QSYM [9]	AFL	QSYM-ce	R	D
Angora [10]	AFL	Angora-ce	R	D
Eclipser [11]	AFL	Eclipser-ce	R	D
Intriguer [12]	AFL	Intriguer-ce	R	D
DigFuzz [13]	AFL	QSYM-ce	MC	D
MEUZZ [14]	AFL	QSYM-ce	ML	D
Pangolin [15]	AFL	QSYM-ce	R	PD

†**Scheduling** - R: Random, MC: Monte Carlo, ML: Machine Learning
‡**Synchronization** - D: Default, PD: Polyhedral Path Abstraction + Dikin Walk

Studied hybrid fuzzers

Benchmarks

15 commonly adopted projects in the studied subjects with their latest versions

Program	Version	Input format	Argument
readelf	binutils-2.37	ELF	-a @@
nm	binutils-2.37	ELF	-C @@
objdump	binutils-2.37	ELF	-D @@
strip	binutils-2.37	ELF	@@
tcpdump	commit-465a8f	PCAP	-r @@
libxml2	2.9.12	XML	@@
libjpeg	v9c	JPEG	@@
jhead	commit-f0a884	JPEG	@@
libpng	1.7.0	PNG	@@
libtiff	4.2.0	TIFF	@@
file	commit-d17d8e	FILE	-m magic @@
bento	commit-7ddec0	MP4	@@
wavpack	commit-36b08d	WAV	-y @@
cyclonedds	commit-53cf7c	IDL	@@
libming	commit-04aee5	SWF	@@

Studied real-world benchmark

RQ1: Performance of hybrid fuzzers

Edge coverage results of the studied fuzzers within 24 hours

Program	AFL	FairFuzz	AFL++	QSYM	Angora	Eclipser	Intriguer	DigFuzz	MEUZZ	Pangolin
readelf	9,176	9,198	9,154	9,512	9,632	9,220	9,620	9,378	9,487	**10,053**
nm	5,127	5,258	5,216	5,602	6,255	5,327	5,154	5,209	5,907	**7,082**
objdump	7,358	7,317	7,415	**8,304**	7,356	7,455	7,743	7,285	7,203	7,894
strip	6,340	7,104	6,788	7,624	7,582	7,210	6,906	7,541	7,195	**7,940**
tcpdump	9,782	9,879	9,955	**10,279**	10,025	10,170	9,302	10,018	9,502	9,320
libxml2	5,876	5,860	5,929	**7,888**	5,909	5,935	5,844	5,945	5,958	5,797
libjpeg	2,902	2,806	2,981	**3,183**	3,101	3,169	2,988	3,120	3,147	3,168
jhead	304	304	304	**885**	304	823	796	747	812	304
libpng	1,496	1,517	1,503	2,058	**2,170**	1,523	1,466	1,485	2,083	1,915
libtiff	3,546	3,642	3,508	3,793	**3,883**	3,764	3,710	3,704	3,761	3,697
file	2,283	2,327	2,346	2,553	**2,571**	2,148	2,342	2,466	2,331	2,391
bento	3,001	3,020	3,135	4,017	3,937	3,566	3,495	3,356	3,855	**4,119**
wavpack	5,703	5,721	5,683	5,797	5,756	5,780	5,612	5,745	5,633	**5,803**
cyclonedds	4,822	4,871	5,012	**5,612**	5,260	4,914	4,885	5,017	5,402	4,956
libming	8,197	8,742	8,775	**9,335**	9,021	8,847	8,941	8,794	9,048	8,983
AVG	5,061	5,171	5,180	5,763	5,517	5,323	5,254	5,321	5,422	5,561

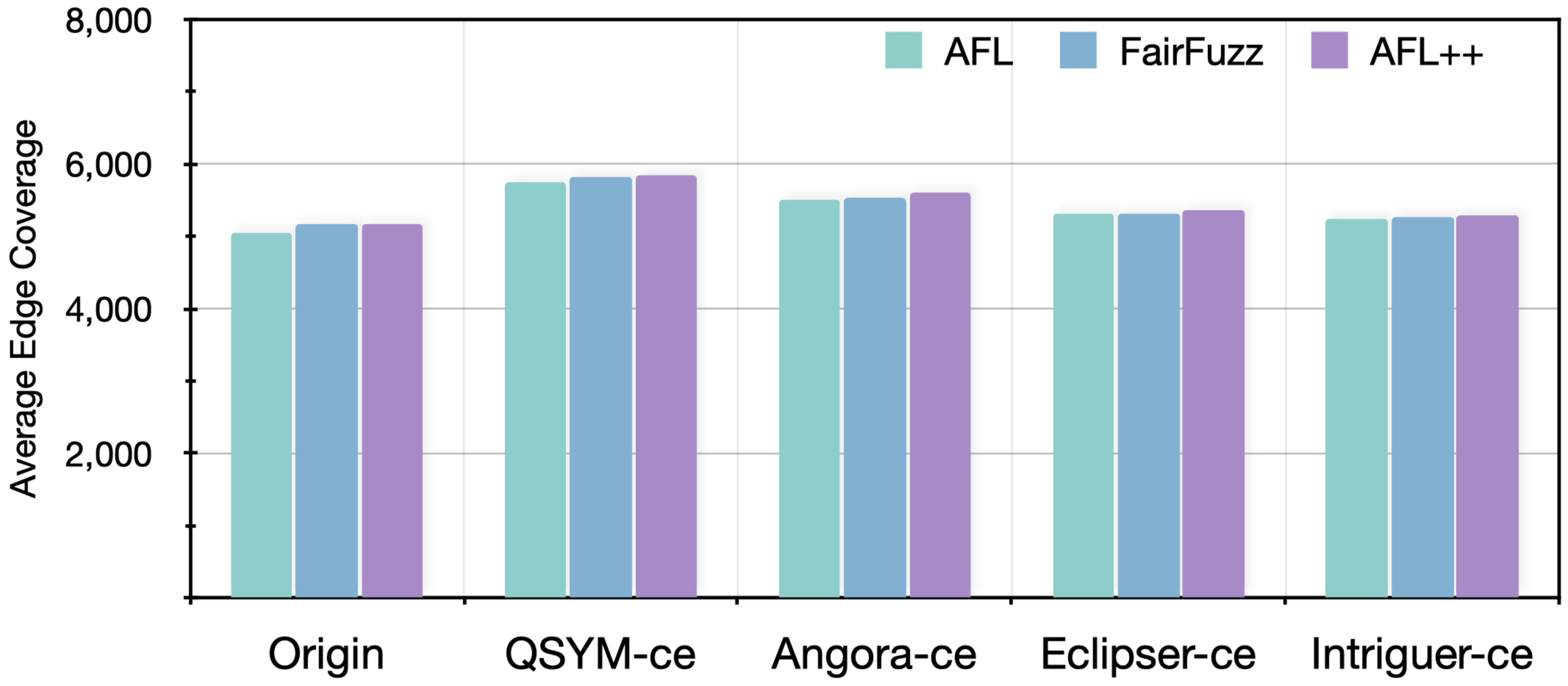
The optimal result in the conventional coverage-guided fuzzers is highlighted in blue and the corresponding inferior results in the hybrid fuzzers are highlighted in green.
The optimal result for each program among all the studied fuzzers is marked in **red**.

Finding: The edge coverage advantages of hybrid fuzzers over conventional coverage-guided fuzzers are somewhat limited, indicating that the power of concolic execution has not been fully leveraged.

RQ1: Performance of hybrid fuzzers

Fuzzing strategy: *AFL, FairFuzz, AFL++*

Concolic executor: *QSYM-ce, Angora-ce, Eclipser-ce, Intriguer-ce*



Average edge coverage of reassembled hybrid fuzzing variants

Finding: Simply updating fuzzing strategies or concolic executors alone in hybrid fuzzers leads to limited edge coverage impact.

Unique crash:

Testcases that trigger crash with unique execution path

Program	AFL	FairFuzz	AFL++	QSYM	Angora	Eclipser	Intriguer	DigFuzz	MEUZZ	Pangolin
readelf	5	6	5	6	7	5	7	4	4	8
nm	10	10	11	10	13	12	7	9	9	13
objdump	3	3	3	3	3	2	3	0	0	3
tcpdump	6	6	5	7	6	4	8	4	3	4
libjpeg	36	30	30	34	28	22	26	18	21	34
jhead	0	2	5	16	0	15	15	12	16	0
libtiff	0	2	3	2	0	1	1	0	0	2
bento	11	19	20	25	28	15	15	21	20	16
cyclonedds	28	30	36	34	32	25	27	29	42	37
libming	10	8	10	10	12	6	10	8	8	10
Total	109	116	128	147	129	107	119	105	123	127

Number of unique crashes on real-world benchmark

Finding: Most studied hybrid fuzzers incur rather limited or even no advantages over conventional coverage-guided fuzzers in exposing unique crashes upon real-world benchmark programs.

Metric: Redundant edge ratio

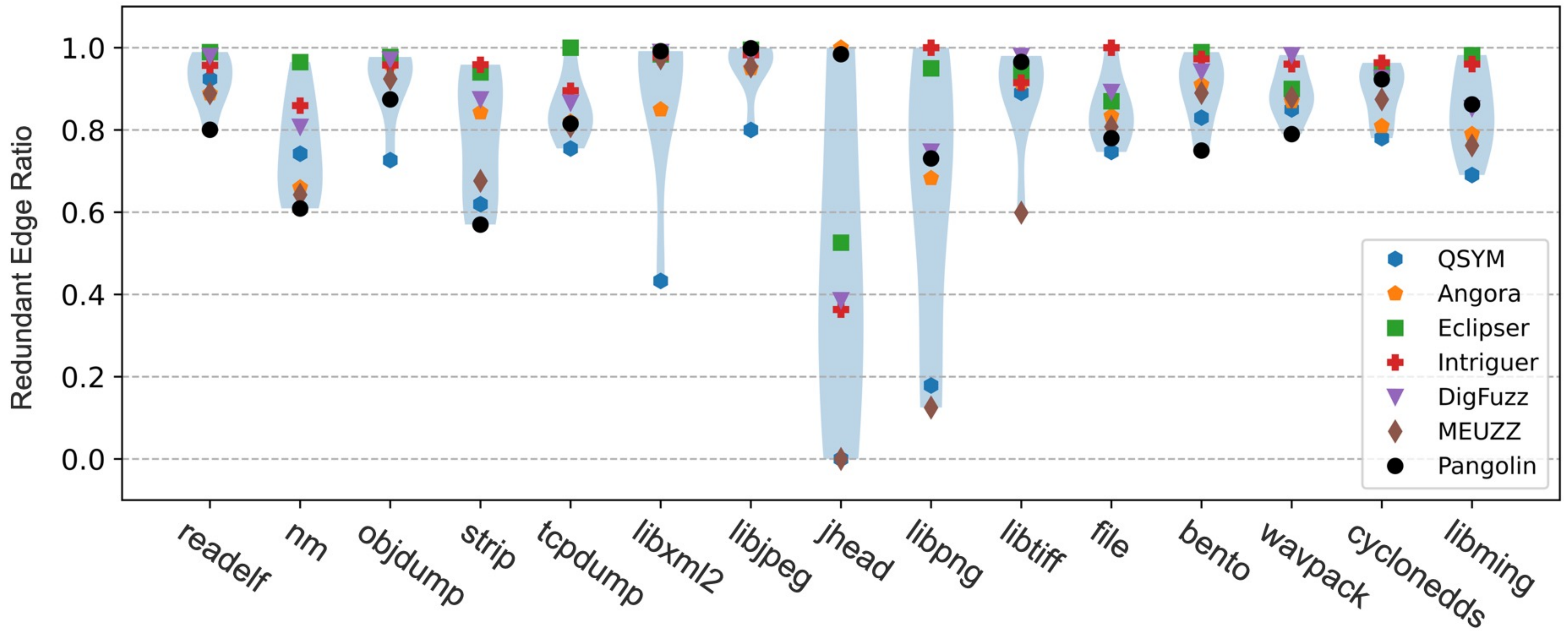
The magnitude of the **common edges** explored by the fuzzing strategy and the concolic executor

$$\phi(F, C) = \frac{|F \cap C|}{|C|}$$

Hybrid fuzzer	Average edge coverage	Average redundant edge ratio
QSYM	**5,763**	**0.65**
DigFuzz	5,321	0.87
MEUZZ	5,422	0.71
Pangolin	5,561	0.82

Average edge coverage and redundant edge ratio

Redundant edge ratio of studied fuzzers



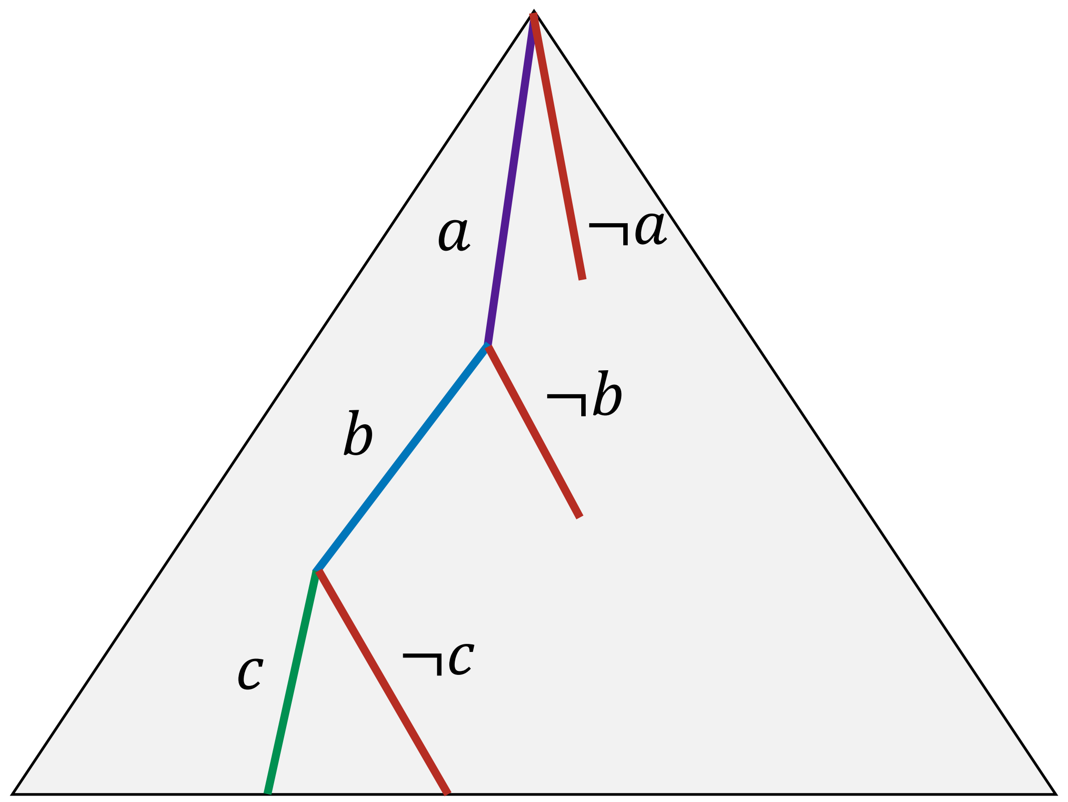
Findings:

- The hybrid fuzzing effectiveness is reflected by redundant edge ratio which is highly relevant to their coordination modes.
- The existing effort on strengthening the scheduling and synchronization mechanisms causes limited impact on edge coverage performance of hybrid fuzzers.

Limitations of seed scheduling

Ineffectiveness of alleviating redundant edge exploration

- Each edge is negated along the execution path during concolic execution
- Coverage updates of fuzzing strategy and concolic executor are mutually **non-transparent until synchronization**

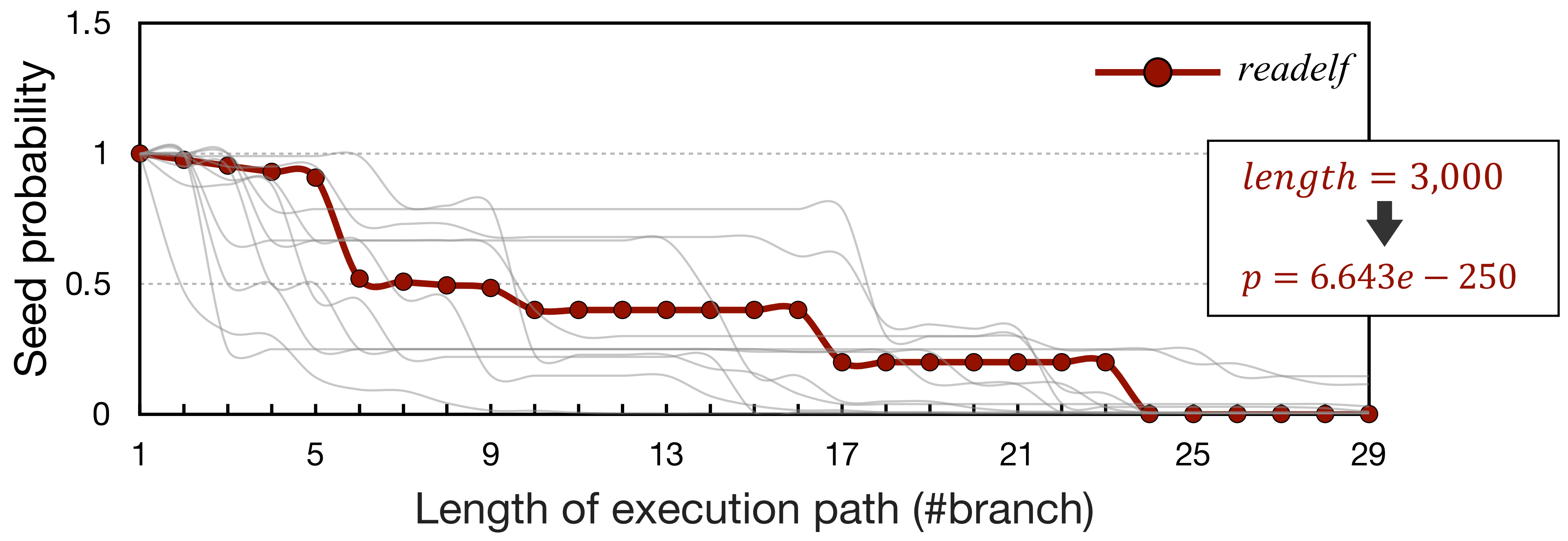


Execution path space

Poor scalability of seed utility prediction

- Inaccurate modeling for seed utility due to the features of the massive edges along the execution path

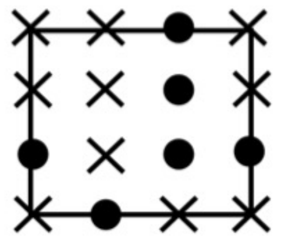
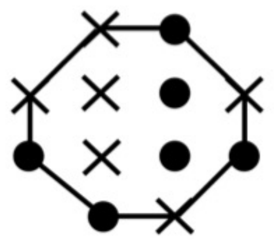
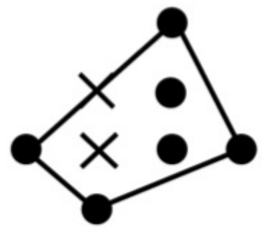
Fine-grained edge-oriented scheduling is essential



Seed utility in DigFuzz

Impact of abstract domains:

- Close effects on limiting the mutation space
- False positives can be quickly filtered by AFL
- One could adopt the **interval abstract domain** with the lowest time cost

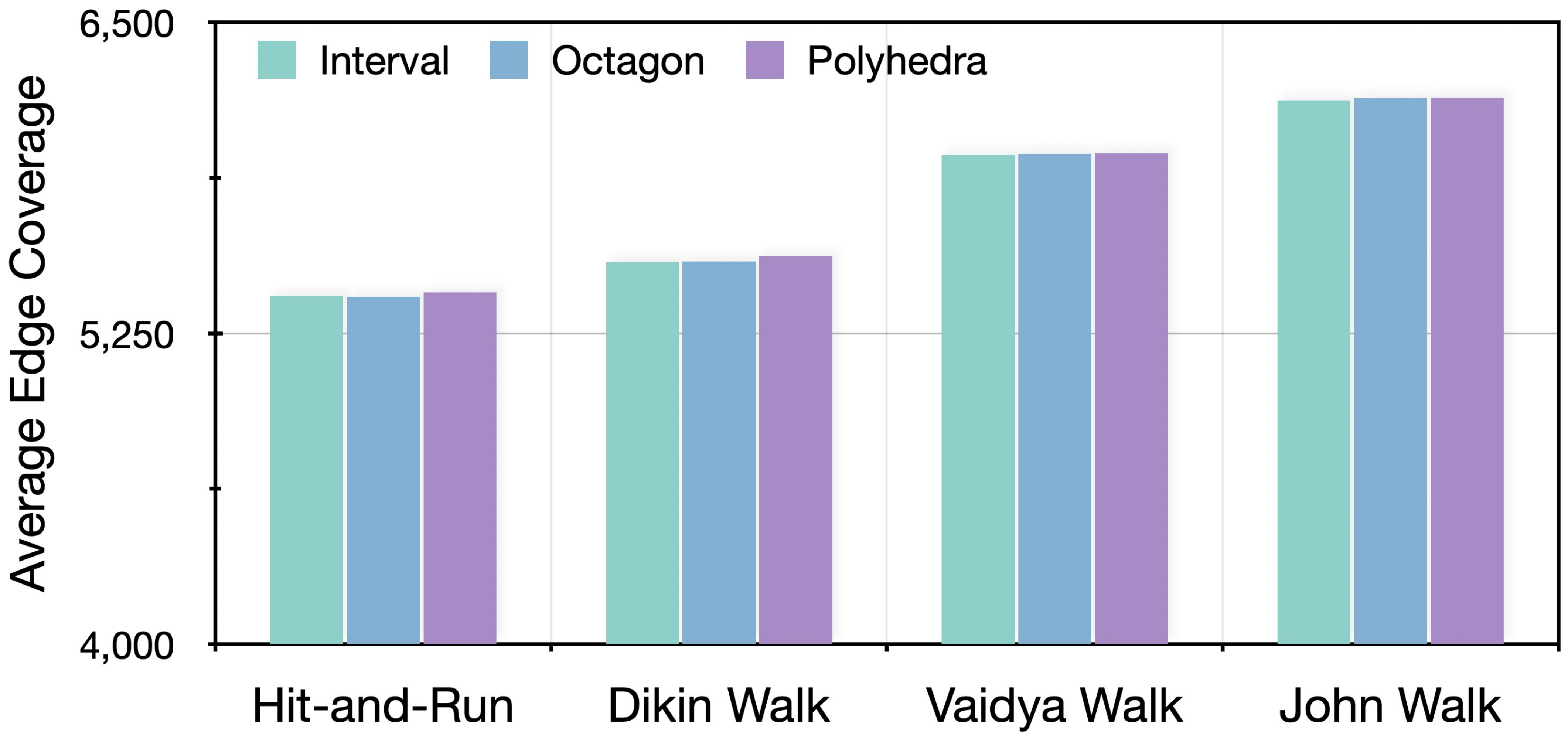
Abstract Domain	Interval	Octagon	Polyhedra
Formulation	$X_i \in [a_i, b_i]$	$\pm X_i \pm X_j \leq c_{ij}, \forall i, j$	$\sum_i a_{ij} X_i \leq b_j$
Precision	Low	Medium	High
Time cost	Non-relational	Cubic	Exponential
Shape			

Sampling Algorithm	Hit-and-run	Dikin walk	Vaidya walk	John walk
Per sample cost	$O(n^3 d^3)$	$O(n^2 d^3)$	$O(n^{1.5} d^{3.5})$	$O(nd^{4.5})$

Different abstraction domains and sampling algorithms

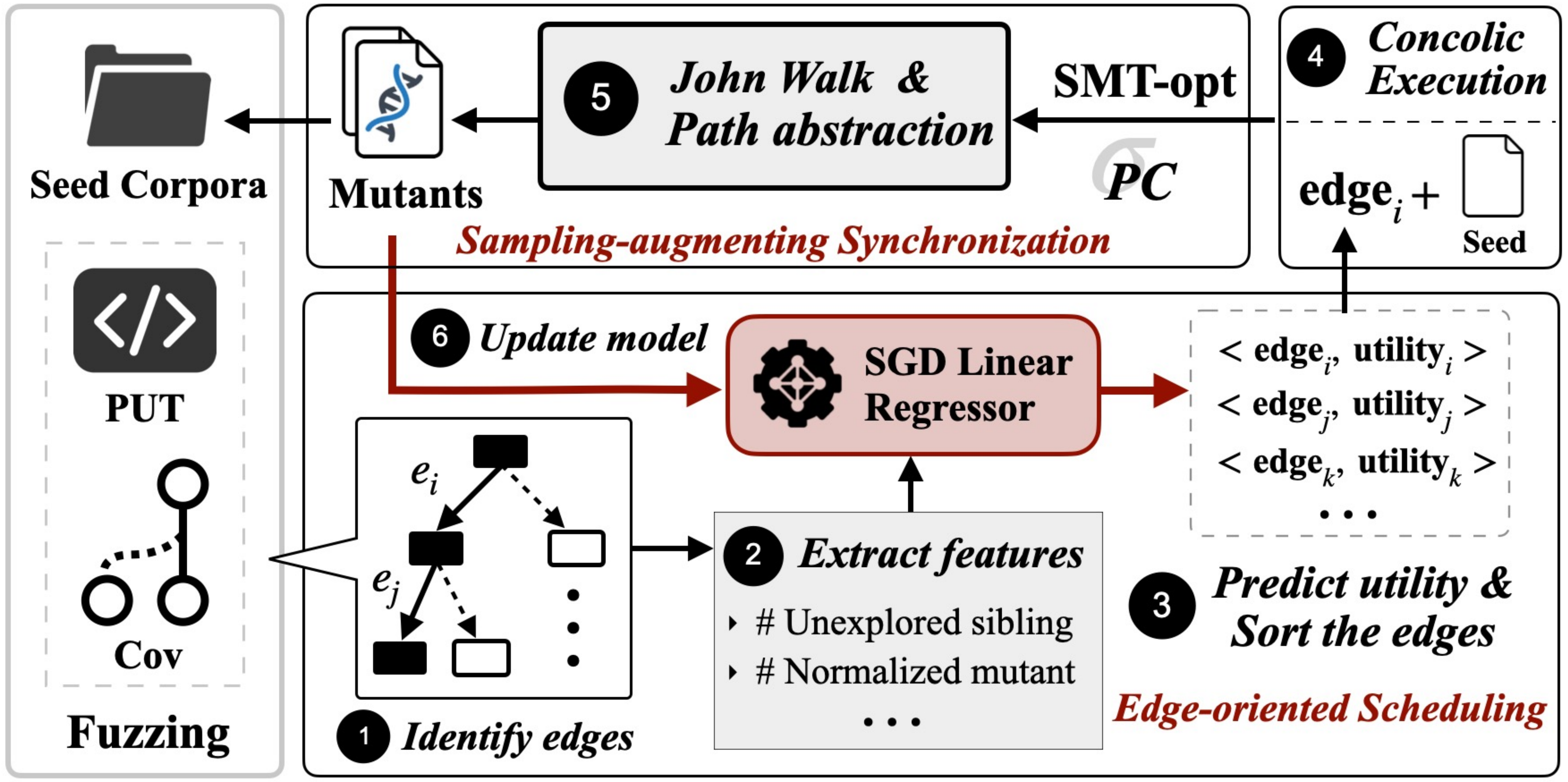
Impact of sampling algorithms:

- John walk and Vaidya walk achieve better performance than Dikin walk and Hit-and-Run
- Generally $n \gg d$ in concolic execution ($n \geq 2d$)
- More efficient to apply **John walk**



Average edge coverage of Pangolin variants

CoFuzz - Coordinated Hybrid **Fuzz**ing Framework with Advanced Coordination Mode



The framework of CoFuzz

Coverage-guided Fuzzing Strategy

- AFL-2.57b

Concolic Execution

- QSYM-ce

Coordination Mode

- Edge-oriented Scheduling ***(discussion 1)***
- Sampling-augmenting Synchronization ***(discussion 2)***

Edge-oriented Scheduling

- Schedule edges to perform concolic execution
- Predict edge utility using the online regression model with *Stochastic Gradient Descent (SGD)*
- **Feature engineering**
  - *Count of unexplored sibling edges*
  - *Normalized mutant amount*
  - *Conditional branch type & bit width*
  - …

Algorithm 1: Coordination Mode of CoFuzz

Input: *Model*
Result: *res*

Function *PerformingCoordinationMode*:
  res ← set()
  candidates ← Set of edges with unexplored sibling edges
  utility ← *Model.predict(candidates)* ; ▷ predict using the linear regression model with SGD
  critical_edges ← *edgeSchedule(candidates, utility)* ; ▷ schedule the edges with high utility
  for *all edge* e_i *in critical_edges* **do**
    s_i ← Identify the seed covering e_i
    pc ← *concolicExec*(s_i, e_i)
    $\hat{\varphi}$ ← *SMTopt(pc)*
    sample_set ← *JohnWalk*$(s_i, \hat{\varphi})$
    for *mutant in sample_set* **do**
      if *increaseCoverage(mutant)* **then**
        res.add(mutant)
    end
    cov_i ← Increased coverage
    Model.update$(e_i,\ cov_i)$
  end
  return *res*

scheduling

CoFuzz: Advanced Coordination Mode

Sampling-augmenting Synchronization

- **John walk within the interval abstraction domain**
  - Increase the edge coverage by generating mutants sampled within the limited mutation space
- **Incremental learning**
  - Collect the coverage updates as labels to update the online SGD regressor
  - Enhance the prediction accuracy during scheduling

Algorithm 1: Coordination Mode of CoFuzz

Input: *Model*
Result: *res*

Function *PerformingCoordinationMode*:
  res ← set()
  candidates ← Set of edges with unexplored sibling edges
  utility ← *Model.predict(candidates)* ; ▷ `predict using the linear regression model with SGD`
  critical_edges ← *edgeSchedule(candidates, utility)* ; ▷ `schedule the edges with high utility`
  for *all edge* e_i *in critical_edges* **do**
    s_i ← Identify the seed covering e_i
    pc ← *concolicExec*(s_i, e_i)
    $\hat{\varphi}$ ← *SMTopt(pc)*
    sample_set ← *JohnWalk*(s_i, $\hat{\varphi}$)
    for *mutant in sample_set* **do**
      if *increaseCoverage(mutant)* **then**
        res.add(mutant)
    end
    cov_i ← Increased coverage
    Model.update(e_i, cov_i)
  end
  return *res*

synchronization

Evaluation: Edge Coverage

Edge Coverage Result

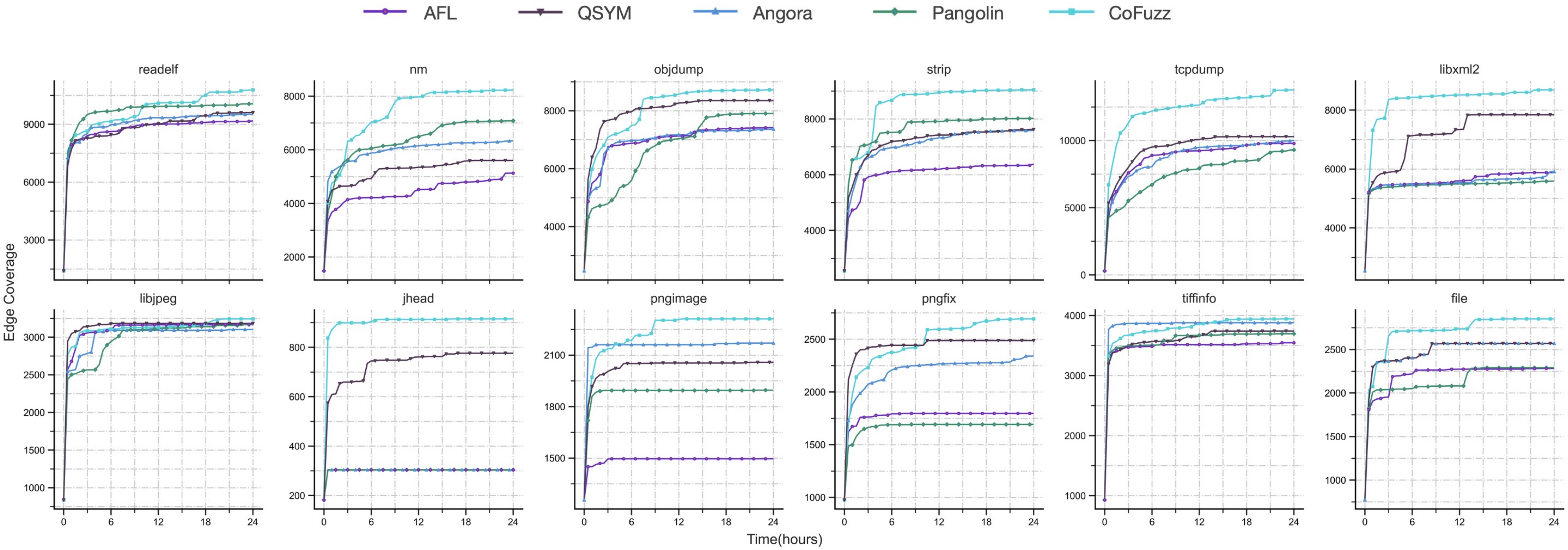
- CoFuzz outperforms AFL by **32.44%** and QSYM by **16.31%**
- Mann Whitney U-test with a one-tailed hypothesis to measure the significance against QSYM *(level = 0.05)*
- Ablation study with CoFuzz variants

$CoFuzz_{sch}$ - *edge-oriented scheduling* only
$CoFuzz_{sync}$ - *sampling-augmenting synchronization* only

Program	AFL	QSYM	$\textbf{CoFuzz}_{sch}$	$\textbf{CoFuzz}_{sync}$	CoFuzz	p-value
readelf	9,176	9,512	10,407	10,236	10,786	0.00596
nm	5,127	5,602	7,824	7,573	8,234	0.00609
objdump	7,358	8,304	8,512	8,453	8,710	0.00609
strip	6,340	7,624	7,839	8,598	9,094	0.00609
tcpdump	9,782	10,279	12,661	10,348	13,130	0.00609
libxml2	5,876	7,888	8,493	7,946	8,640	0.00609
libjpeg	2,902	3,183	3,192	3,190	3,210	0.01059
jhead	304	885	897	890	915	0.00199
libpng	1,496	2,058	2,239	2,197	2,311	0.00609
libtiff	3,546	3,793	3,820	3,842	3,974	0.01059
file	2,283	2,553	2,652	2,730	2,851	0.00609
bento	3,001	4,017	5,624	5,398	6,179	0.00609
wavpack	5,703	5,797	5,832	5,857	5,863	0.00596
cyclonedds	4,822	5,612	5,832	5,713	5,932	0.00609
libming	8,197	9,335	10,177	9,846	10,719	0.00609
AVG	5,061	5,763	6,400	6,188	6,703	0.00640

Edge coverage result of CoFuzz

Evaluation: Edge Coverage



The edge coverage results of CoFuzz over time

Evaluation: BUG Detection

LAVA-M Dataset

- Fully expose the injected bugs with less bug survival time in the subjects *base64, md5sum, uniq*
- Expose the most bugs in *who* and outperform Angora by **23.66%**

Fuzzer	base64		md5sum		uniq		who	
	N	T_m	N	T_m	N	T_m	N	T_m
QSYM	44/44	8.48	57/57	31.77	28/28	4.55	1,332/2,136	300.00
Angora	48/44	6.75	57/57	16.37	29/28	7.15	1,547/2,136	300.00
Eclipser	46/44	128.33	57/57	147.35	29/28	155.83	1,030/2,136	300.00
Intriguer	46/44	205.07	57/57	132.60	29/28	187.22	1,350/2,136	300.00
DigFuzz	46/44	7.53	57/57	57.30	28/28	4.32	1,146/2,136	300.00
MEUZZ	44/44	7.28	57/57	40.35	28/28	6.50	1,205/2,136	300.00
Pangolin	48/44	9.37	57/57	132.75	29/28	13.27	1,342/2,136	300.00
CoFuzz	48/44	1.07	57/57	1.75	29/28	0.50	1,913/2,136	300.00

Bug results of CoFuzz on LAVA-M

Real-world Benchmarks

- 2X more unique crashes
- 37 previously unknown bugs with **8 new CVEs**

Program	Function	Bug Type	Count	Bug Status
readelf	process_object	memory leaks	1	Confirmed
nm	demangle_path	stack-buffer-overflow	1	Confirmed
	str_buf_append	stack-buffer-overflow	1	Patched
objdump	unknow module	invalid memory reference	1	Patched
strip	bfd_getl32	heap-buffer-overflow	3	CVE-2022-38533 & Fixed
	bfd_getl32	invalid memory reference	2	Patched
	group_signature	heap-use-after-free	1	Patched
libjpeg	jpeg_read_scanlines	use-of-uninitialized-value	1	Confirmed
jhead	ReadJpegSections	use-of-uninitialized-value	1	CVE-2022-37165
libtiff	_tiffMapProc	use-of-uninitialized-value	2	Reported
	tiffcp	heap-buffer-overflow	1	Confirmed & Fixed
file	file_tryelf	allocation-size-too-big	1	Confirmed & Fixed
bento	ParseExtension	heap-buffer-overflow	1	CVE-2022-37167 & Fixed
	WriteBytes	heap-buffer-overflow	1	CVE-2022-37169
	AP4_HvccAtom	heap-buffer-overflow	3	CVE-2022-37690
	AP4_StsdAtom	invalid memory reference	2	CVE-2022-37166 & Fixed
	AP4_AvccAtom	invalid memory reference	1	CVE-2022-37168 & Fixed
	Create	memory leaks	2	CVE-2022-37691
wavpack	MD5_Final	heap-buffer-overflow	2	Confirmed & Fixed
cyclonedds	parse_line	heap-buffer-overflow	3	Confirmed & Fixed
	idl_reference_node	heap-use-after-free	2	Confirmed & Fixed
	idlc_parse	stack-buffer-overflow	2	Confirmed & Fixed
	idl_parse	invalid memory reference	2	Confirmed & Fixed
libming	newVar_N	heap-buffer-overflow	2	Reported
	decompileAction	invalid memory reference	3	Reported

Extensive Study

- Study the state-of-the-art hybrid fuzzers on a comprehensive benchmark suite
- The performance of existing hybrid fuzzers may not well generalize to other experimental settings
- The coordination mode can be a crucial factor to augment the performance of hybrid fuzzers

Technique Improvement

- Hybrid fuzzing framework CoFuzz based on findings that significantly outperform the existing hybrid fuzzers in terms of edge coverage and bug detection
- Open-sourced fuzzing approach