

OSCAR

A Noise Injection Framework for Concurrent Java Programs



Filipe de Luna, João M.S. Lourenço
NOVA Laboratory for Computer Science and Informatics
NOVA University Lisbon
Lisbon, Portugal
https://nova-lincs.di.fct.unl.pt/

Jeremy S. Bradbury
Software Engineering & Education Research Lab
Ontario Tech University
Oshawa, Canada
https://www.seerlab.ca





Concurrent programs are hard to test

Everywhere

Multicore processors are ubiquitous, so software must use concurrency to perform well.

Exploding

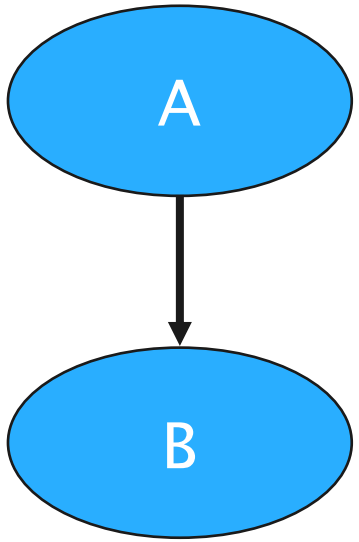
The number of possible thread interleavings grows exponentially with threads and instructions.

Hidden

Rare interleavings depend on the scheduler, so bugs are hard to reproduce and tests give false confidence.

© 2026 Jeremy Bradbury

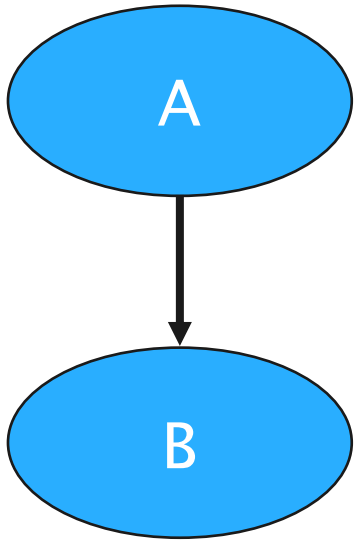
Single vs. Multi-Threaded Programs



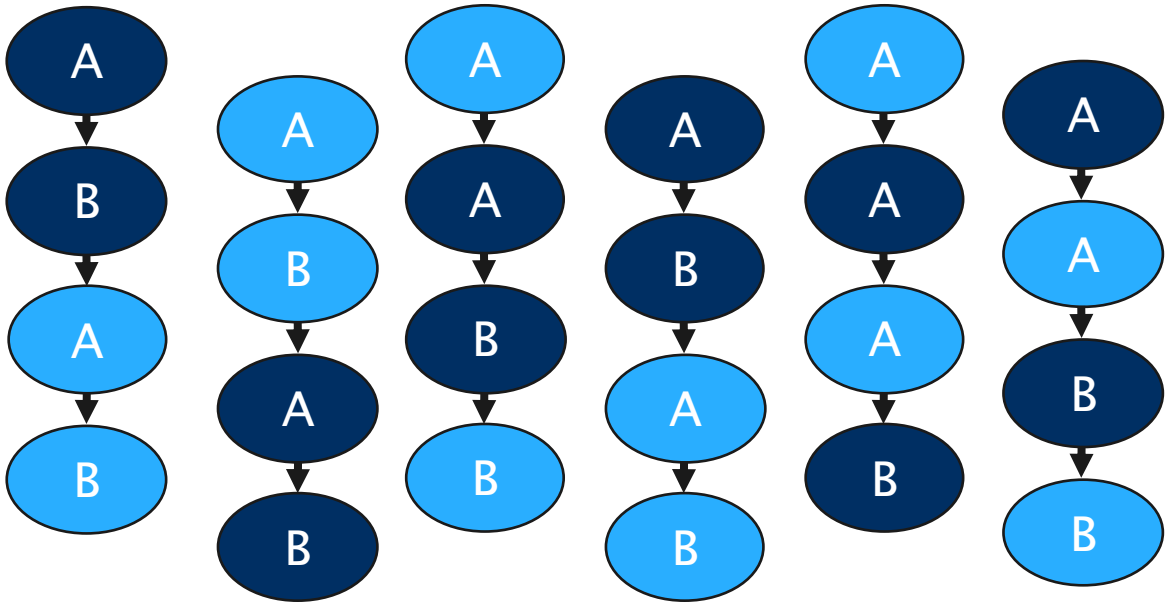
Single-Thread

© 2026 Jeremy Bradbury

Single vs. Multi-Threaded Programs



Single-Thread



Multi-Threaded (2)

© 2026 Jeremy Bradbury

Concurrent Programming Challenges

- Concurrent execution is **non-deterministic**
- Different interleavings exhibit **different** probabilities of occurring
- Without **actively** exploring the interleaving space, successive executions may generate the same subset of interleavings

© 2026 Jeremy Bradbury

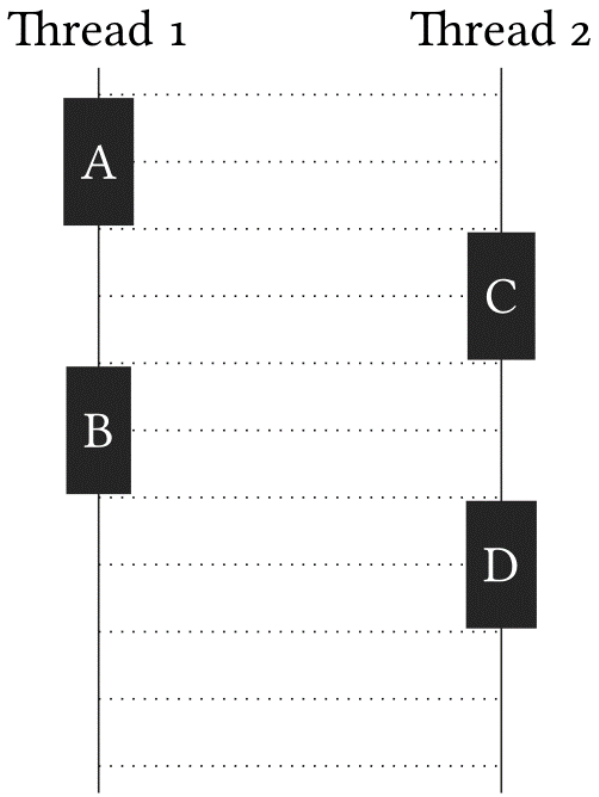
Noise injection

Noise injection heuristics explicitly varies thread schedules by inserting random delays into execution.

The probe effect: a best-effort technique, not a guarantee of exploring every interleaving.

© 2026 Jeremy Bradbury

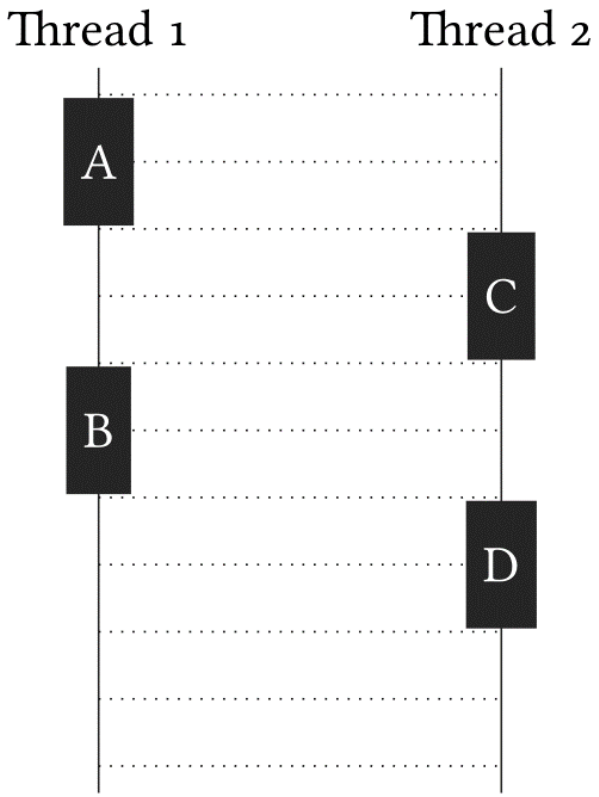
Noise Injection Example



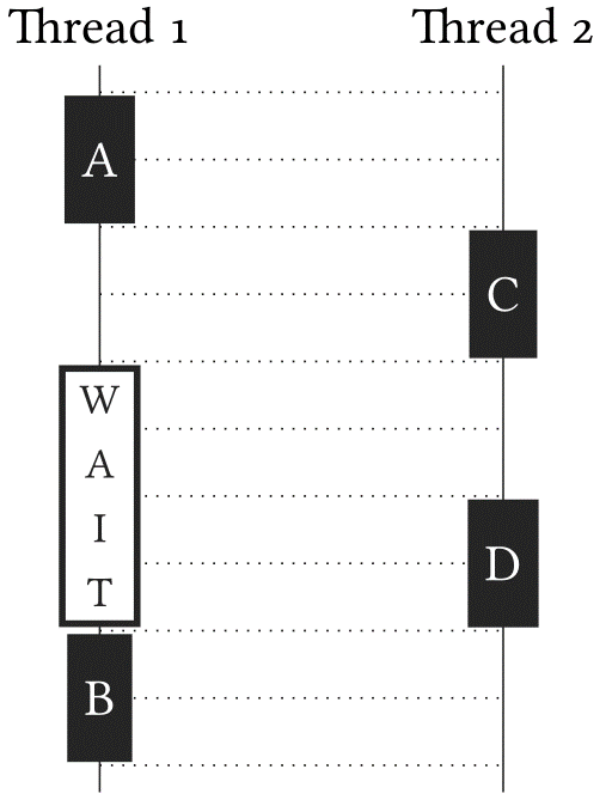
Without Noise

© 2026 Jeremy Bradbury

Noise Injection Example



Without Noise



With Noise

© 2026 Jeremy Bradbury

Noise Injection

- Noise injection heuristics allow a tester to maximize the impact of the noising on the scheduler
- These heuristics are made up of 3 components:
  - ***Noise Type***
    - *Sleep, Yield, Wait, Suspend and Priority*
  - ***Placement***
  - ***Seeding***
    - *Intensity (e.g., delay length as interval)*
    - *Probability (e.g., frequency as %)*
- No “silver bullet” heuristic exists [1]

[1] V. Hrubá, B. Křena, Z. Letko, S. Ur, and T. Vojnar. Testing of Concurrent Programs Using Genetic Algorithms. In Search Based Software Engineering, pages 152–167. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-33119-0_12

© 2026 Jeremy Bradbury

Java lacks an open noise injection framework

Previous standard noise tool is gone

The best-known Java framework for noising (ConTest) is no longer available or under active development.

Concurrent testing techniques hard to compare

Without standard noising practices, performance differences between analysis tools may come from the noising, not the analysis.

We need a reusable, open-source building block for Java.

© 2026 Jeremy Bradbury

Contributions

1 **OSCAR, an open-source noise injection framework**

Instruments Java bytecode with Soot; github.com/seer-lab/OSCAR

2 **A taxonomy of noise injection heuristics**

A common vocabulary for heuristics proposed by different researchers.

3 **A metric for the distance between interleavings**

Traces mapped to strings and compared with Jaro similarity, to measure interleaving space coverage.

© 2026 Jeremy Bradbury

Noise heuristics: Location

Synchronisation-block

Noise before, after or inside synchronised blocks.

Read/Write

Noise around shared-variable accesses, to expose data races.

Function

Noise inside, before or after function calls, including monitored ones.

Thread

Noise triggered only for a specific thread.

Pattern

Noise where code shows a pattern, such as repeated access to the same variables.

Random

A random mix of the others. It performed poorly and can mask errors.

Categories are not mutually exclusive: heuristics can be combined.

© 2026 Jeremy Bradbury

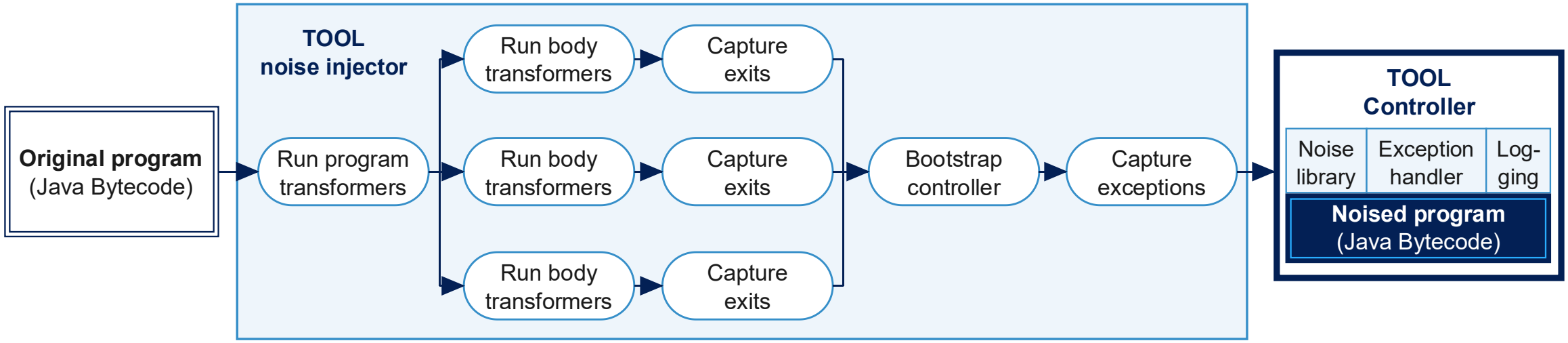
Noise heuristics: Type

Placement decides which locations and runs get noise. Seeding decides the noise type, frequency and intensity.

Noise type	What it does	Trade-off
Sleep	**Pauses the thread for t ms**	**Most effective; large slowdown**
Yield	**Hints the scheduler to switch threads**	**Almost no slowdown; may be ignored**
Wait	Waits, releasing the monitor	BusyWait is good but slow
Priority	Changes a thread’s priority	Makes threads lazy or eager
Suspend	Suspends until resumed	Deprecated; prone to deadlock

© 2026 Jeremy Bradbury

OSCAR Architecture



© 2026 Jeremy Bradbury

Example: noising a synchronized block

Before

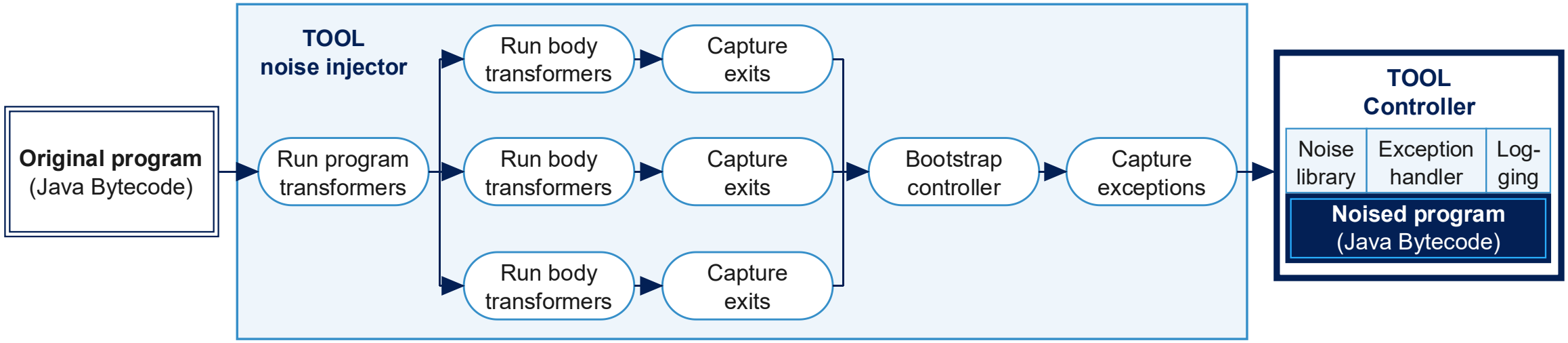
```
public static void add() {
  synchronized (counter) {
    Counter.add();
  }
}
```

After

```
public static void add() {
  Controller.noise(
    BEFORE_SYNC_BLOCK, UUID);
  synchronized (counter) {
    Counter.add();
  }
  Controller.noise(
    AFTER_SYNC_BLOCK, UUID);
}
```


© 2026 Jeremy Bradbury

OSCAR Architecture



© 2026 Jeremy Bradbury

What OSCAR can noise and configure

Noise locations

- Synchronized methods and blocks
- Reentrant lock and unlock operations
- Thread start and thread routine
- Shared fields, found by static analysis
- Class constructors

Run-time options

- Noise type: sleep or yield
- Intensity: random between lower and upper bounds
- Probability of triggering at each location
- Which heuristics and locations are active
- Trace capture, output and class blacklist

© 2026 Jeremy Bradbury

Measuring coverage: tracing interleavings

1 **Trace at the noise locations**

An event is a thread passing a location: a (thread ID, location ID) pair.

2 **Normalize the trace**

Renumber threads by order of appearance, so equivalent interleavings collapse, then map each pair to one character.

3 **Compare the strings**

Jaro similarity was the best of five string metrics studied, and it handles loops well.

© 2026 Jeremy Bradbury

Understanding Coverage of the Interleaving Space

Unique interleavings

Distinct traces among the runs. Higher means more of the state space explored.

Trace similarity

We use Jaro similarity. Mean over all pairs of traces, from 0 to 1. Lower means more diverse interleavings.

© 2026 Jeremy Bradbury

Trace Example

Thread ID	Trace Location (UUID)	Final (mapped) ID
1	b686818e	
5	ec765900	
1	ec765900	
1	b686818e	
5	ec765900	
12	12869a7c	

© 2026 Jeremy Bradbury

Trace Example

Thread ID	Trace Location (UUID)	Final (mapped) ID
1	b686818e	
2	ec765900	
1	ec765900	
1	b686818e	
2	ec765900	
4	12869a7c	



Thread IDs are mapped to their order of appearance

© 2026 Jeremy Bradbury

Trace Example

Thread ID	Trace Location (UUID)	Final (mapped) ID
1	b686818e	A
2	ec765900	B
1	ec765900	C
1	b686818e	A
2	ec765900	B
4	12869a7c	D



(ID, UUID) pairs are mapped to a single character

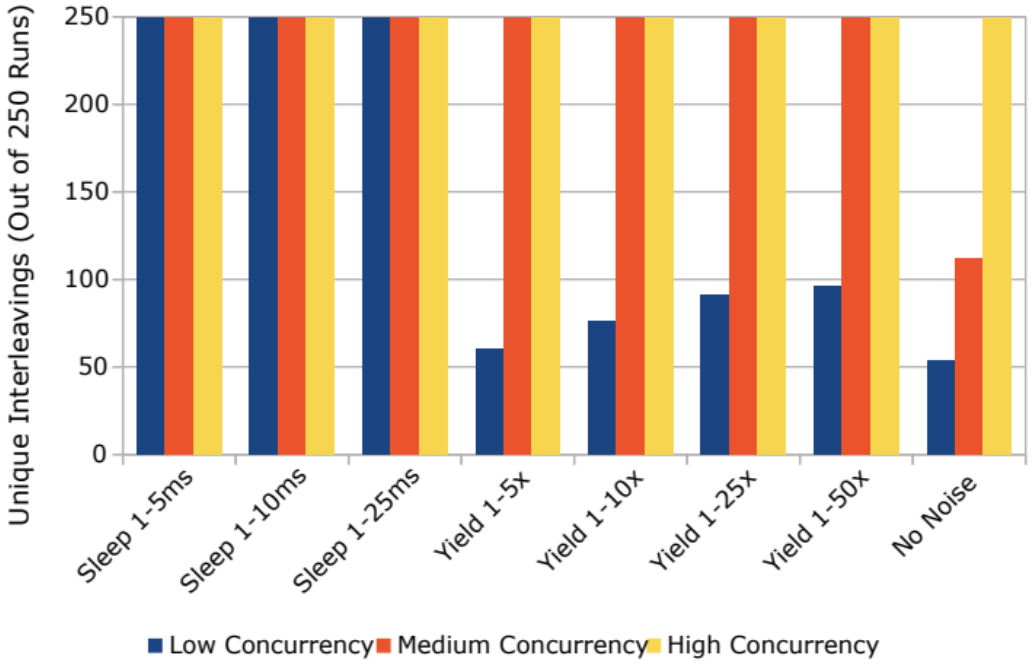
© 2026 Jeremy Bradbury

Trace Analysis

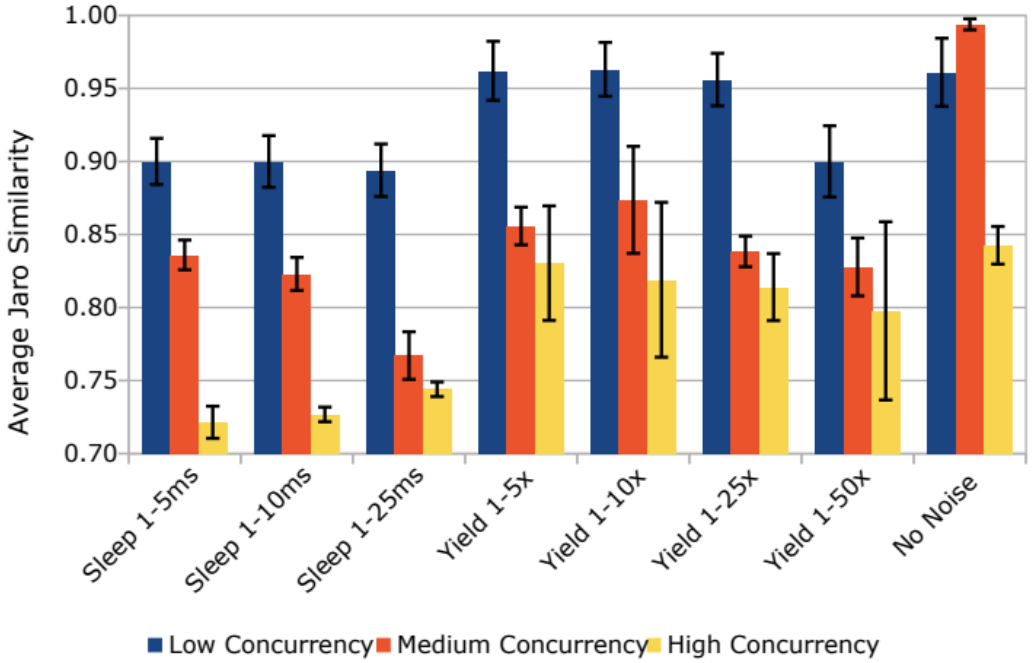
- Several string comparison metrics were considered and explored:
  - Levenshtein distance
  - Jaro/Jaro-Wrinkler similarity
  - Hamming distance
  - Jaccard similarity (MinHashing [1]) – *want to investigate further*
- We opted for **Jaro Similarity** which considers matching characters, transpositions, string length

© 2026 Jeremy Bradbury

Trace Analysis



Uniqueness of interleavings in *lottery* program



Similarity of interleavings in *lottery* program

© 2026 Jeremy Bradbury

Evaluation on the IBM Concurrency Benchmark

24

publicly available benchmark programs

250

runs per program and heuristic

Research questions

1. Is OSCAR feasible for large code bases and long executions?
2. Does it trigger latent bugs?
3. Does it increase coverage of the interleaving space?

© 2026 Jeremy Bradbury

RQ1: OSCAR scales to large programs

< 1 min

to instrument a medium-size project on modern hardware

10M

instrumentation points in the largest synthetic test

56 ns

per runtime library call

The callback is negligible next to even the shortest 1 ms sleep, so it has limited impact on the program's timing and behaviour.

© 2026 Jeremy Bradbury

RQ2: OSCAR exposes latent bugs

18/24

programs had a latent bug exposed, two thirds of the benchmark

4

programs with bugs hard to see without noise, triggered in virtually every noised run

1

additional bug found that the original developer had not documented

No universal heuristic. Sleep usually beat yield, with exceptions, and noise sometimes masked an error by separating the code blocks that race.

© 2026 Jeremy Bradbury

RQ3: Noise broadens interleaving coverage

More interleavings

Noised programs consistently showed more unique interleavings than non-noised runs.

More diverse

They were also less similar to each other, with lower average Jaro similarity.

Sleep leads

Sleep consistently explored more of the state space than yield, but it can mask errors that are not latent.

Tested on account, pingpong and lottery, at low, medium and high concurrency. Some programs kept producing new interleavings, so their state space is probably much larger.

© 2026 Jeremy Bradbury

Future work

- Replay of recorded interleavings
- A richer interleaving distance that includes timing and code statements
- Mixed noise types in one run
- Automatic search for the best heuristic and configuration

© 2026 Jeremy Bradbury

Conclusions

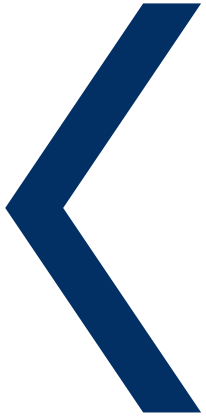
OSCAR makes noise injection for Java open, measurable and reusable.

- Noise consistently produced more, and more diverse, interleavings.
- Sleep is the most effective; yield is the efficient option when time is tight.
- No heuristic fits every program, so tools should support several.



github.com/seer-lab/OSCAR

© 2026 Jeremy Bradbury



OSCAR

A Noise Injection Framework for Concurrent Java Programs



Filipe de Luna, João M.S. Lourenço
NOVA Laboratory for Computer Science and Informatics
NOVA University Lisbon
Lisbon, Portugal
https://nova-lincs.di.fct.unl.pt/

Jeremy S. Bradbury
Software Engineering & Education Research Lab
Ontario Tech University
Oshawa, Canada
https://www.seerlab.ca



