IEEE WCCI 2020
IEEE World Congress on Computational Intelligence
Virtual Conference – July 19-24, 2020

Injecting Shortcuts for Faster Running Java Code

Alexander E.I. Brownlee[1], Justyna Petke[2], Anna F. Rasburn[1]

[1] University of Stirling, sbr@cs.stir.ac.uk

[2] University College London

Outline

- Motivation + intro to genetic improvement of software
- Proposed edit operators
- Research Questions
- Experimental framework
  - Gin
  - Target applications
- Results
  - Random sampling
  - Timing sampling
- Summary

Motivation

- Programming (well) is hard!

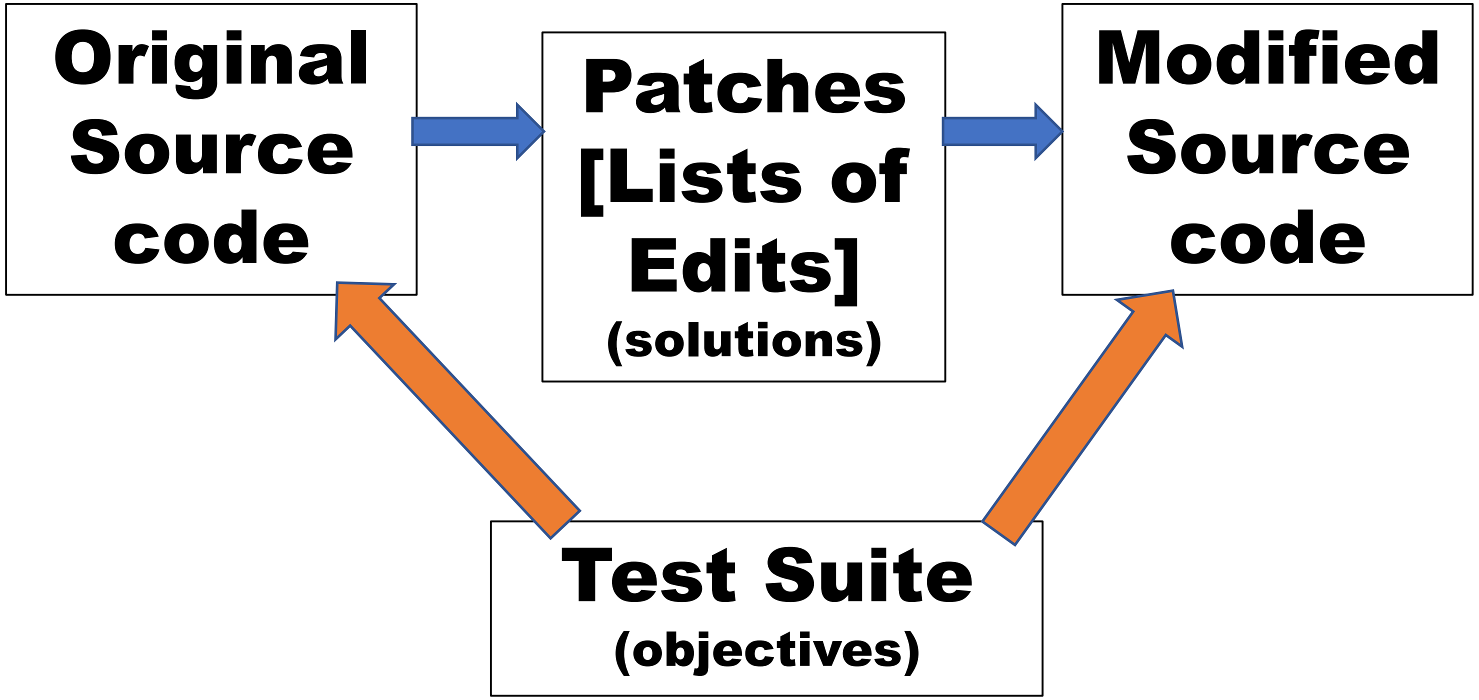
- Many aspects of getting code right can be formulated as search problems
    - e.g. what's the best way to implement X?
    - what's the best order in which to perform XYZ?

- Genetic Improvement of Software: Let's use machines to do the search (the boring bit) so humans can focus on being creative

Genetic Improvement of Software

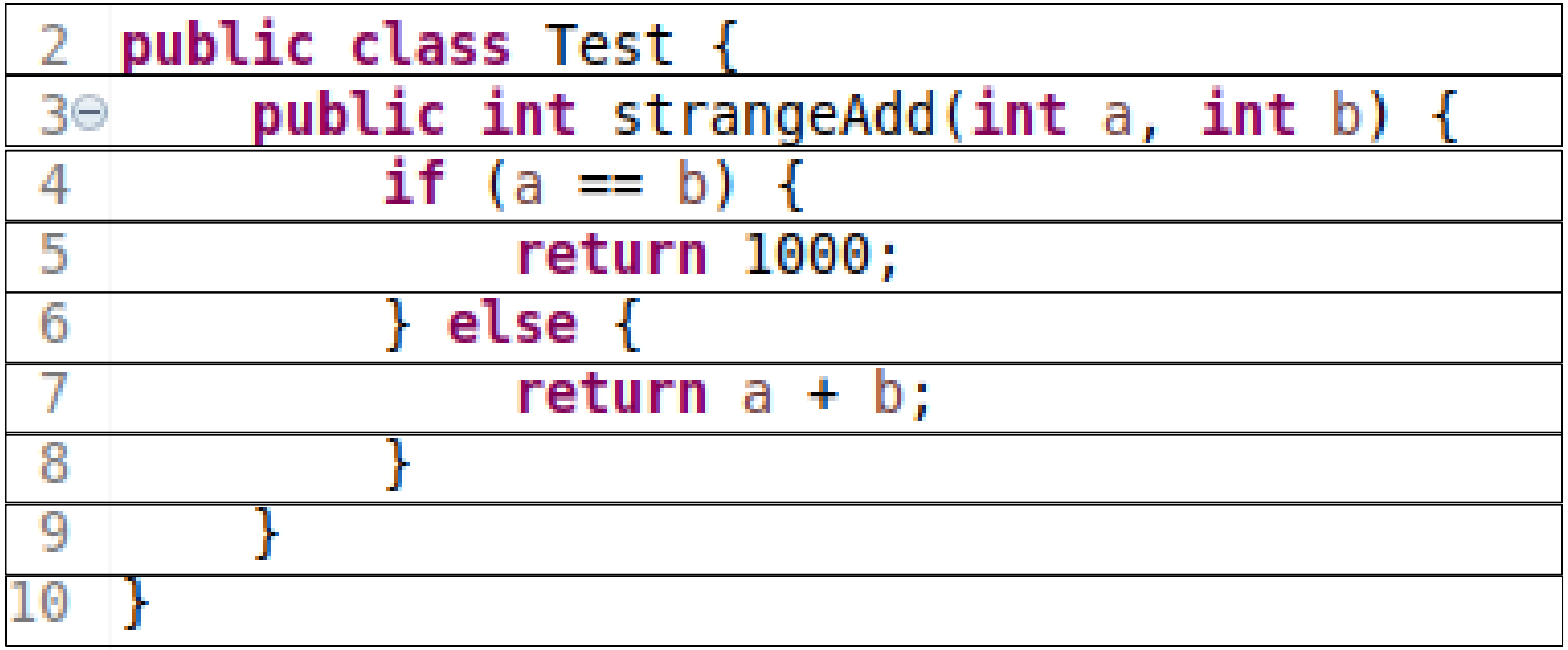
- Applying search to find improvements in software (fix bugs, speed, memory, energy…)
- Apply edits to source code: delete, move, copy, replace… lines, statements, operators, constants…
- Run tests to validate changed code
- Some impressive results! Particularly for specialising generic code

  E.g. Langdon & Harman (2014) – 70x speedup of Bowtie2 DNA Sequencing System, written in C++



Edits

- Most approaches use fairly simple edits

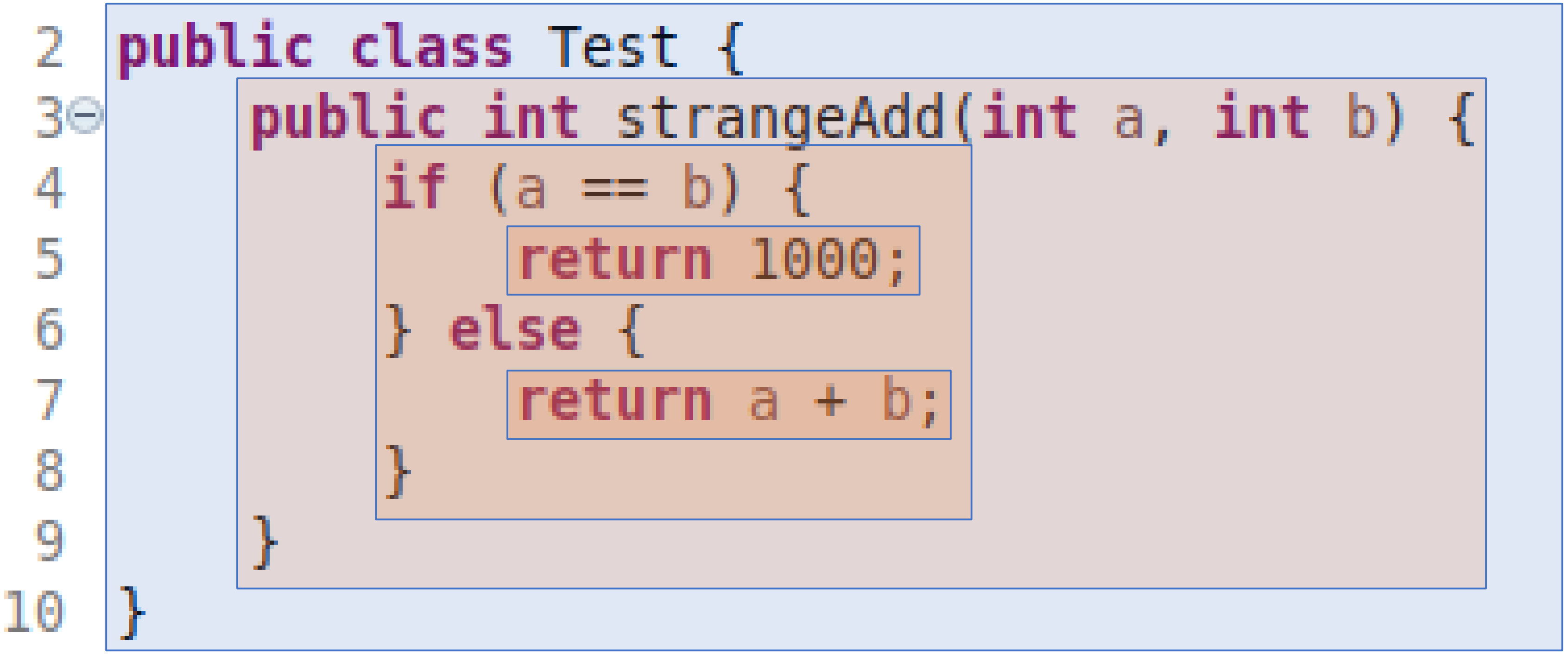


```
public class Test {
    public int strangeAdd(int a, int b) {
        if (a == b) {
            return 1000;
        } else {
            return a + b;
        }
    }
}
```

Copy, Delete, Replace, Swap Lines

Edits

- Most approaches use fairly simple edits



```java
public class Test {
    public int strangeAdd(int a, int b) {
        if (a == b) {
            return 1000;
        } else {
            return a + b;
        }
    }
}
```

Copy, Delete, Replace, Swap Statements

Edits

- In automated program repair, more work dedicated to finding more efficient operators
  - E.g. swapping + to - ; swap < to > ; adding exception handling

- Less for non-functional properties like run time

- Idea for new edits: can we skip some parts of code to get the same (or similar) behaviour, but save on execution time?

New Edits

- Insert statements into existing code… (focus on Java)
- $\mathcal{B}$: `break;`
- $\mathcal{C}$: `continue;`
- $\mathcal{R}$: `return;`
- $\mathcal{B}_{if}$: `if (a) break;`
- $\mathcal{C}_{if}$: `if (a) continue;`
- $\mathcal{R}_{if}$: `if (a) return;`

If conditions

- “a” is a boolean expression
- An in-scope primitive variable **v** is chosen at the insertion point and one of the following is chosen at random to make “a”…

- v or !v // if v is boolean

- v<0 v<=0 v==0 v=>0 v>0 // if v is any other primitive

Research Questions

- **When applying our six operators:**
- How often do we produce compiling code?
- How often do we produce working code?
- How often do we obtain speedups?

Experimental Framework

- Operators implemented in Gin
  - https://github.com/gintool/gin

- Gin identifies “hot methods” by profiling unit tests and finding places where CPU spends most time

- Apply operators to hot methods

- Run modified code on unit tests, check results and time

Experimental Framework

- Targeted three projects:
  - jCodec 0.2.3 (135k lines of code)
  - spark 2.7.2 (15k lines)
  - spatial4j 0.7 (14k lines)

Experimental Framework

- Enumeration experiment
  - All possible applications of $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{R}$ to the hot methods
- **Random sampling**
  - **Generate 10k edits of each type and apply to hot methods**
- Local search
  - Take top 5 most promising target methods in jCodec and apply hillclimber to find run time improvements
- **New results: random samping, run time measurements**

Random Sampling Experiment

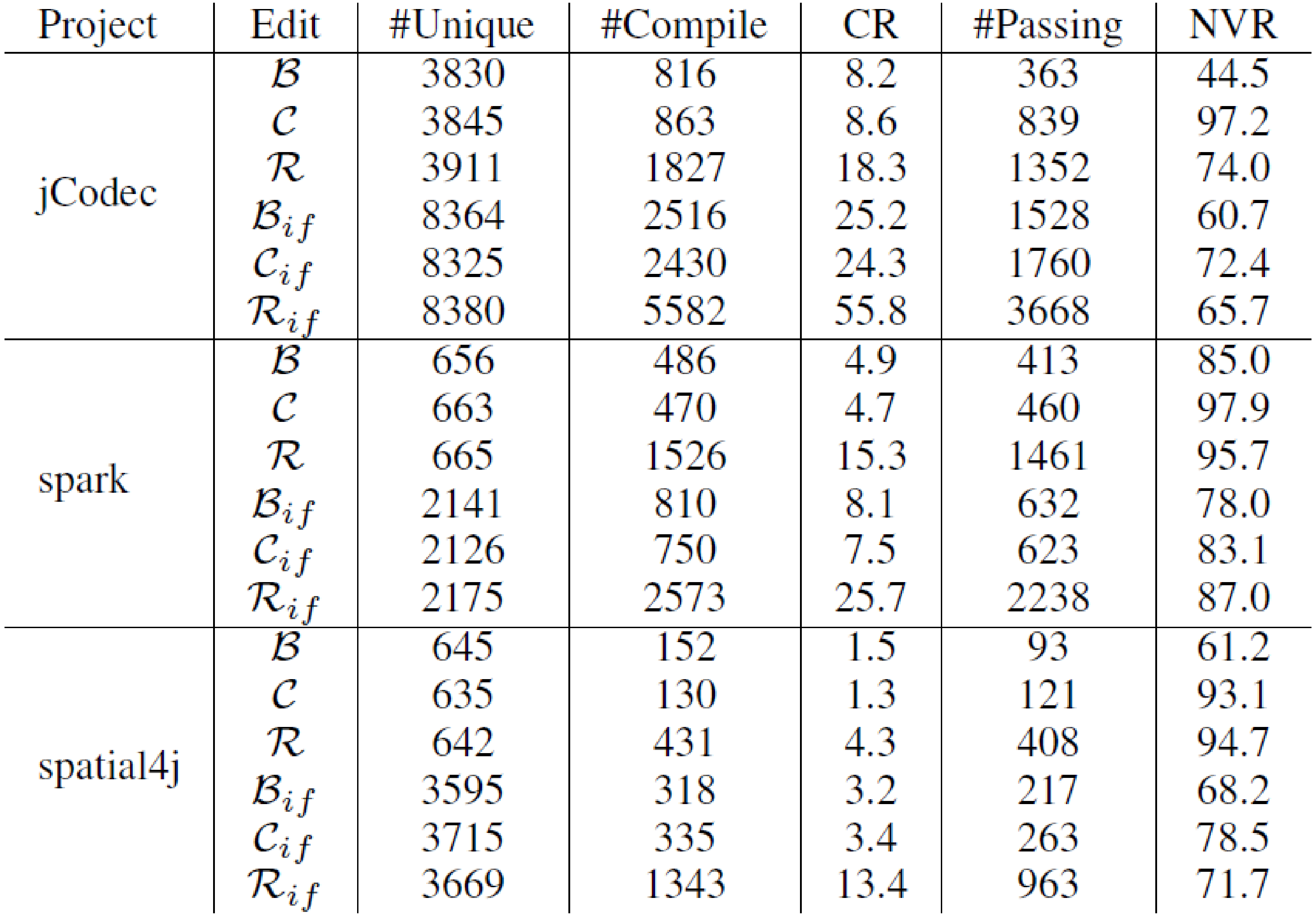
- Generate 10k variations of each edit type
- Apply the 60k edits to the hot methods of each project
- Measure:
  - How many compiled?
  - How many ran and passed the unit tests?
- Calculate:
  - How many of the 10k are unique edits?
  - Compilation Rate: % of 10k edits compiling
  - Neutral Variant Rate: % of compiling edits that pass all tests

Random Sampling Experiment

- Main obstacle is passing compilation
  - NVR rates > 44% in all cases
  - Many neutral variants: good potential for exploring non-functional properties
- Adding “if” improves compilation rate
  - No surprise!
- Inserting

  `if (a) return;`

  seems to be best



Project	Edit	#Unique	#Compile	CR	#Passing	NVR
jCodec	$\mathcal{B}$	3830	816	8.2	363	44.5
	$\mathcal{C}$	3845	863	8.6	839	97.2
	$\mathcal{R}$	3911	1827	18.3	1352	74.0
	$\mathcal{B}_{if}$	8364	2516	25.2	1528	60.7
	$\mathcal{C}_{if}$	8325	2430	24.3	1760	72.4
	$\mathcal{R}_{if}$	8380	5582	55.8	3668	65.7
spark	$\mathcal{B}$	656	486	4.9	413	85.0
	$\mathcal{C}$	663	470	4.7	460	97.9
	$\mathcal{R}$	665	1526	15.3	1461	95.7
	$\mathcal{B}_{if}$	2141	810	8.1	632	78.0
	$\mathcal{C}_{if}$	2126	750	7.5	623	83.1
	$\mathcal{R}_{if}$	2175	2573	25.7	2238	87.0
spatial4j	$\mathcal{B}$	645	152	1.5	93	61.2
	$\mathcal{C}$	635	130	1.3	121	93.1
	$\mathcal{R}$	642	431	4.3	408	94.7
	$\mathcal{B}_{if}$	3595	318	3.2	217	68.2
	$\mathcal{C}_{if}$	3715	335	3.4	263	78.5
	$\mathcal{R}_{if}$	3669	1343	13.4	963	71.7

Random Sampling - Times

- jCodec only
- 7000 edits sampled uniformly at random from the 6 types over the hot methods
- For those which compiled and passes tests:
  - repeat 30 times:
    - Run unit tests on original code
    - Run unit tests on patched code
    - Log wall-clock time for each
  - (aiming to reduce impact of caching etc)

Random Sampling - Times

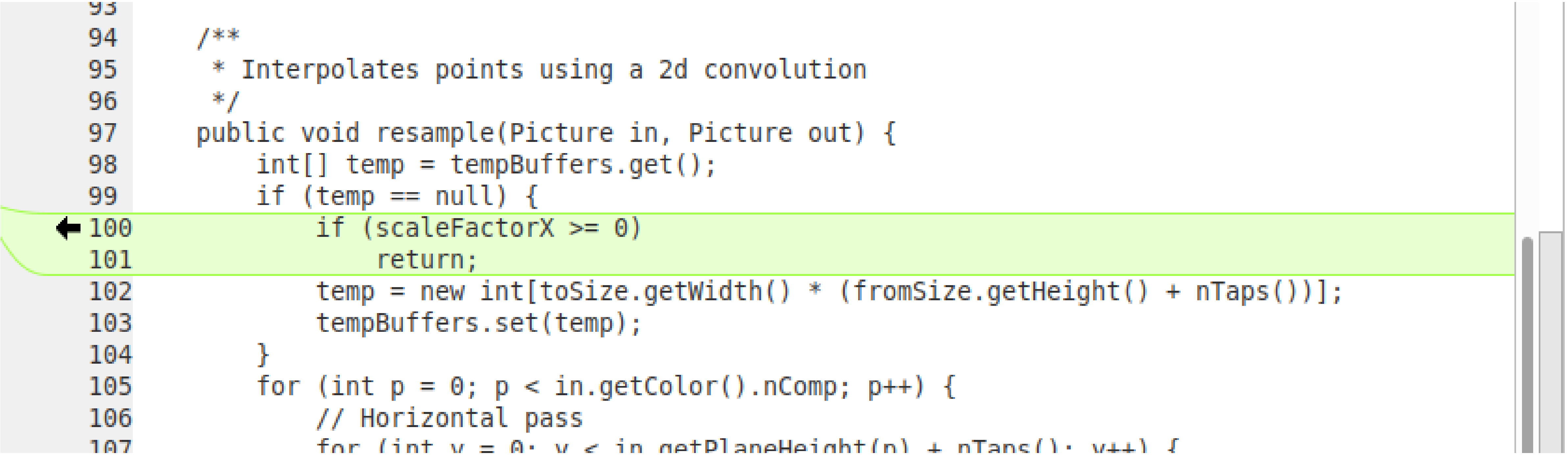
- 3509 / 7000 edits compiled and passed the tests
- 1366 of these (~20% of 7000) offered a significant decrease in run time over the original code
  - t-test $p < 0.05$ on the 30 repeats
  - these were:

B	C	R	B_{if}	C_{if}	R_{if}
134	8	352	166	121	585

- 84 edits produces a significant increase in run time
- 15.1% mean reduction in run time for those with a significant reduction

Best improvement

- 28% speedup
- org.jcodec.scale.BaseResampler.java



```
    /**
     * Interpolates points using a 2d convolution
     */
    public void resample(Picture in, Picture out) {
        int[] temp = tempBuffers.get();
        if (temp == null) {
            if (scaleFactorX >= 0)
                return;
            temp = new int[toSize.getWidth() * (fromSize.getHeight() + nTaps())];
            tempBuffers.set(temp);
        }
        for (int p = 0; p < in.getColor().nComp; p++) {
            // Horizontal pass
            for (int y = 0; y < in.getPlaneHeight(p) + nTaps(); y++) {
```

Summary

- Six new edit types for genetic improvement of Java code
- Compilation rates higher for "if" edits; up to 55.8%
- High neutral variant rates; >44.5%
- Modified code often offers a speedup
- Future work:
  - Non-zero comparisons for the "if" (tricky to sample)
  - Return non-nulls
  - Static analysis to target insertion points
  - More "grammar aware" edits
- Edits available at https://github.com/gintool/gin
- sbr@cs.stir.ac.uk