Software Mutational Robustness

A presentation by Rebecca Sousa

Presentation Overview

Introduction

Background

Technical Approach

Experimental Results

Applications

Discussion

Introduction

Terms and Definitions

- **Neutral mutation:** a random change to a program that still passes the test suite
- **Software mutational robustness** measures the fraction of neutral mutations
- Infinite number of ways to implement an algorithm in code
- Quicksort example:

```
if (right > left) {
  // code elided ...
  quick(left, r);
  quick(l, right);
}
```

→

```
quick(l, right);
quick(left, r);
```

Background

Biology

- Environmental and mutational robustness
- Neutral neighbors and neutral spaces

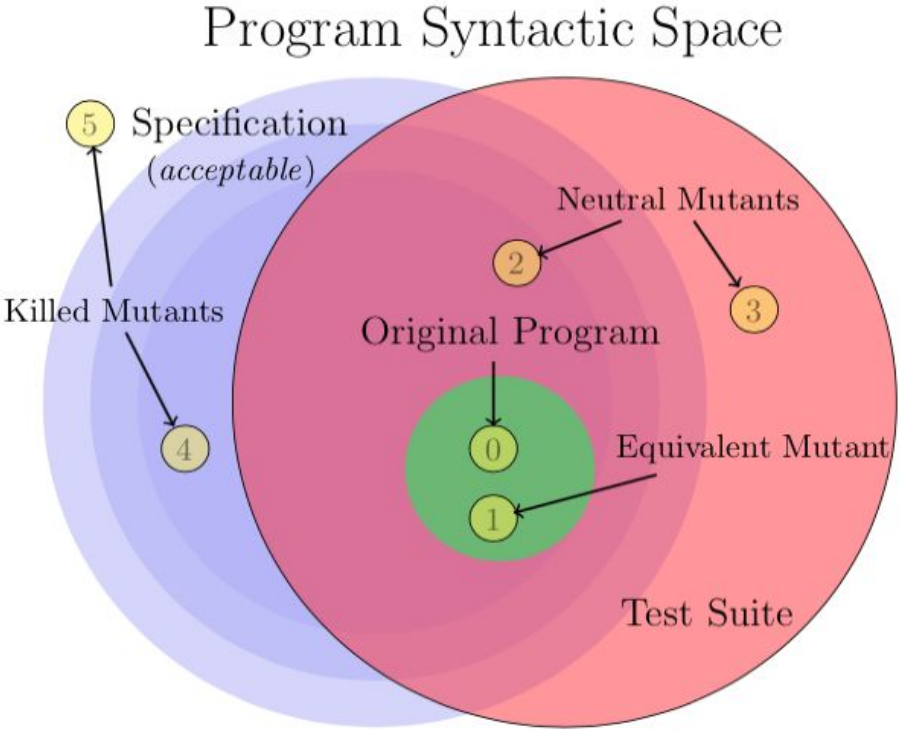
Evolutionary Computation

- Genetic programming (GP)

Software Engineering

- Mutation Testing
- N-Version Programming

Mutation Testing



Mutation Testing

```
/*
 * Spec (S):
 *     Pre: parameter P is an array of three integer elements
 *     Post: returns the smallest of the three input elements
 */

int a(int p[]) {
  if (p[0] <= p[1] && p[0] <= p[2]) return p[0];
  if (p[1] <= p[2] && p[1] <= p[0]) return p[1];
  else return p[2];
}

int b(int p[]) {
  sort(p, "ascending");
  return p[0];
}
```

Technical Approach

- Program P
- Variant P’
- Mutation operators M (copy, swap, delete)
- Test suite T
- Finding: MutRB does not depend strongly on P or T

$$MutRB(P, T, M) = \frac{|\{P' \mid m \in M.\ P' = m(P)\ \wedge\ T(P') = \mathsf{true}\}|}{|\{P' \mid m \in M.\ P' = m(P)\}|}$$

Representation and Operators

Representation

- Abstract syntax trees (AST)
- Low-level assembly code (ASM)

Operators

- Copy
- Delete
- Swap

Representation and Operators

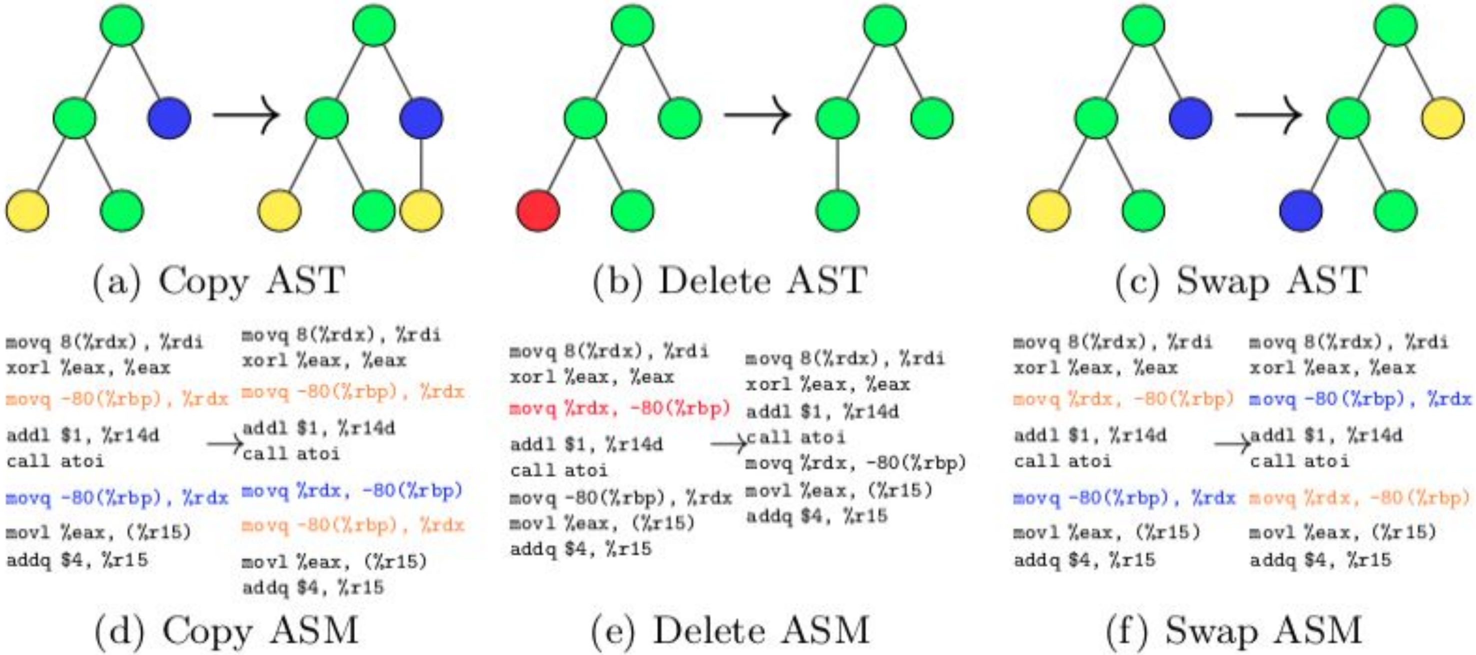
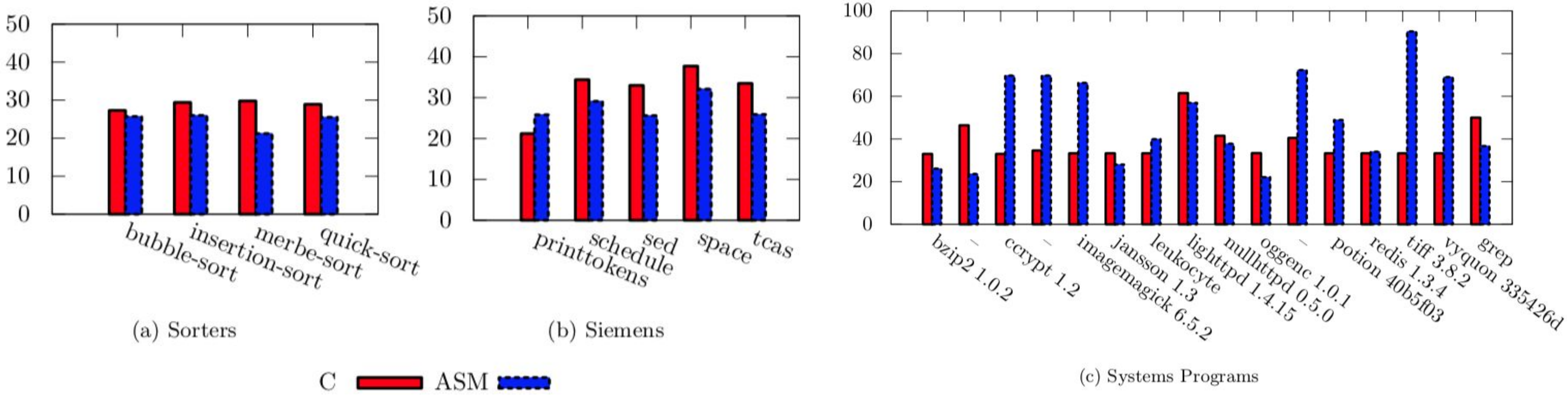


Figure 3: Mutation operators: Copy, Delete, Swap.

Experimental Results

- 22 benchmark programs with test suites
  - Sorters
  - Siemens
  - Off-the-shelf
- **First order mutation**: apply a single random mutation to copy of program
- Want to rule out trivial mutations that produce equivalent assembly code
- **36.8%** of variants continue to pass all test cases (?!)
- What is the cause of this?
  - Inadequate test suites (bad)
  - Semantically equivalent mutations (good)

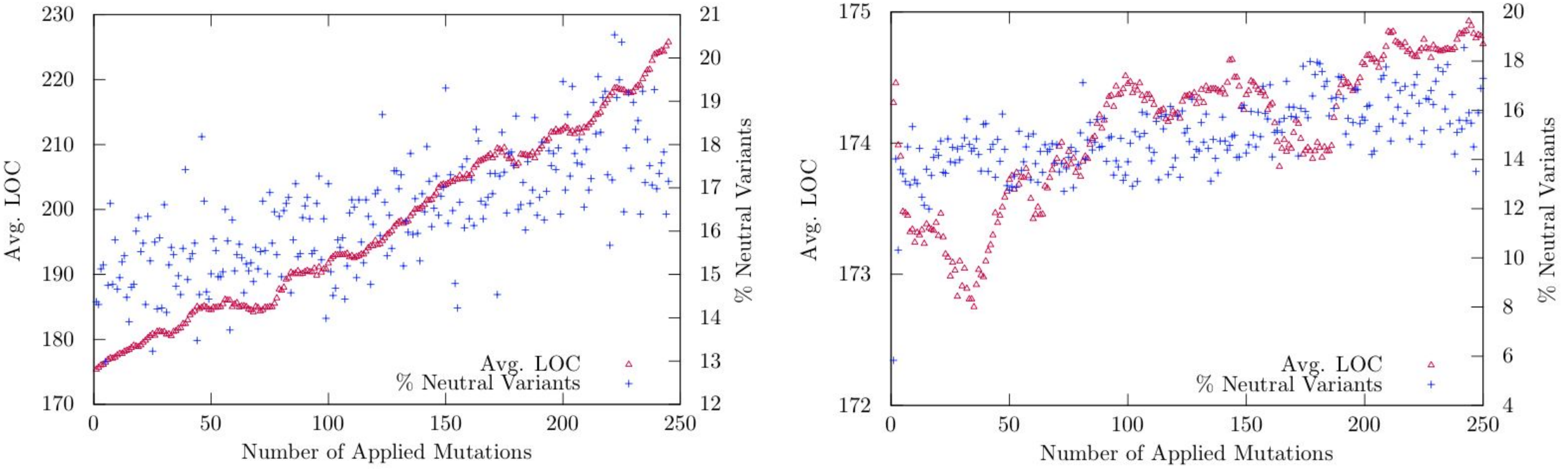
Does robustness depend on test suite?



Taxonomy of Neutral Variants

#	Functional Category	Frequency/35
1	Different whitespace in output	12
2	Inconsequential change of internal variables	10
3	Extra or redundant computation	6
4	Equivalent or redundant conditional guard	3
5	Switched to non-explicit return	2
6	Changed code is unreachable	1
7	Removed optimization	1

Cumulative Robustness



(a) Program size not controlled.

(b) Program size controlled.

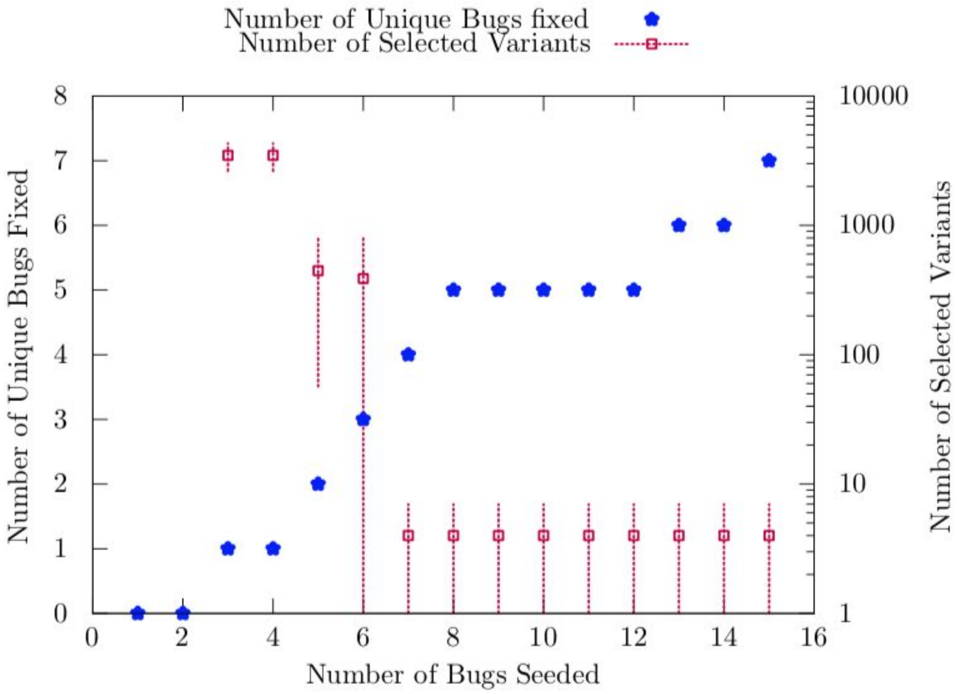
Multiple Languages

	C	C++	Haskell	OCaml	Avg.	Std.Dev.
bubble	25.7	28.2	27.6	16.7	24.6	5.3
insertion	26.0	42.0	35.6	23.7	31.8	8.5
merge	21.2	46.0	24.9	22.7	28.7	11.6
quick	25.5	42.0	26.3	11.4	26.3	12.5
Avg.	24.6	39.5	28.6	18.6	27.9	
Std.Dev.	2.3	7.8	4.8	5.7	3.1	

Table 3: Mutational robustness of sorting algorithms at the assembly instruction level with 100% test suite coverage, for different algorithms and source language.

Application: Proactive Bug Repair

Program	Fraction of Bugs Fixed	Bug Fixes
bzip	2/5	63
imagemagick	2/5	8
jansson	2/5	40
leukocyte	1/5	1
lighttpd	1/5	73
nullhttpd	1/5	7
oggenc	0/5	0
potion	2/5	14
redis	0/5	0
tiff	0/5	0
vyquon	1/5	1
average	1.0/5	18.8



Application: N-Version Programming

- Want to develop N independent software instances
- Separate teams of human programmers likely to create similar programs - creating independence is hard!
- Solution: generate independent programs through neutral mutations

Discussion

Threats to Validity

- Choice of mutation operators
- Insufficient test suites

Further Investigation

- Landscape of neutral variants
- Markov Chain Monte Carlo

Applications to Software Engineering

- Optimization
- Automated program repair
- Mutation testing

Comparison to Biology

- Role of neutrality in evolution

Thank You!

Any questions?