Bad Pairs in Software Testing

Dan Hoffman and Mitch Chang
Department of Computer Science
University of Victoria

Gary Bazdell and Brett Stevens
Department of Mathematics and Statistics
Carleton University

Kevin Yoo
Wurldtech Security Technologies

Bad Pairs: Intuition

a	b	c	P/F
1	0	0	P
1	2	1	P
1	**2**	**3**	F
⋮	⋮	⋮	⋮
2	0	0	P
2	1	1	P
3	**2**	**3**	F
⋮	⋮	⋮	⋮
4	0	0	P
5	1	3	P
7	**2**	**3**	F
⋮	⋮	⋮	⋮

Overview

- Open questions
  - What is a useful formalization of *bad pair*?
  - How common are bad pairs?
  - What is the effect of input selection on bad pairs?
  - What is the relationship between faults and bad pairs?
- Experiments: triangle, TCAS
- Case study: industrial network vulnerability testing
- Powerful new theory result: error-locating arrays

Test Table, Singleton, and Pair

Test Table

Input Table			**Results Vector**
a	*b*	*c*	P/F
1	0	0	P
1	2	1	P
1	2	3	F
2	0	0	P
2	1	1	P
3	2	3	F
4	0	0	P
5	1	3	P
7	2	3	F
⋮	⋮	⋮	⋮

Bad Singleton

Test Table

Input Table			Results Vector
a	**b**	c	P/F
⋮	⋮	⋮	⋮
0	0	2	P
3	**1**	2	**F**
1	**1**	3	**F**
4	**1**	3	**F**
3	**1**	3	**F**
1	**1**	4	**F**
1	2	3	P
⋮	⋮	⋮	⋮

Dependent Bad Pair

Test Table

Input Table			Results Vector
a	**b**	**c**	P/F
⋮	⋮	⋮	⋮
0	0	2	P
3	1	2	F
1	**1**	**3**	**F**
4	**1**	**3**	**F**
3	**1**	**3**	**F**
1	1	4	F
1	2	3	P
⋮	⋮	⋮	⋮

Independent Bad Pair

Test Table

Input Table			**Results Vector**
a	**b**	**c**	P/F
⋮	⋮	⋮	⋮
0	2	2	P
⋮	⋮	⋮	⋮
3	2	2	F
1	**2**	**3**	**F**
4	**2**	**3**	**F**
3	**2**	**3**	**F**
1	2	4	P
⋮	⋮	⋮	⋮
1	3	3	P
⋮	⋮	⋮	⋮

Triangle: Experimental Design

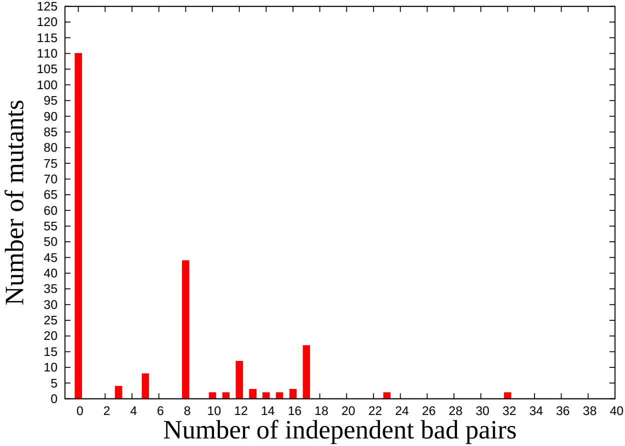
- Gold code: triangle program, hand-translated to Java
- Input table (216): $[0..5] \times [0..5] \times [0..5]$
- Mutants: single (213), double (212)
- Execution pseudocode

for each mutant M
    open log file L_M
    for each test case t
        run M with input t
        run the gold code with input t
        if M and the gold code produce the same output
            write t followed by 'P' to L_M
        else
            write t followed by 'F' to L_M
    close log file L_M

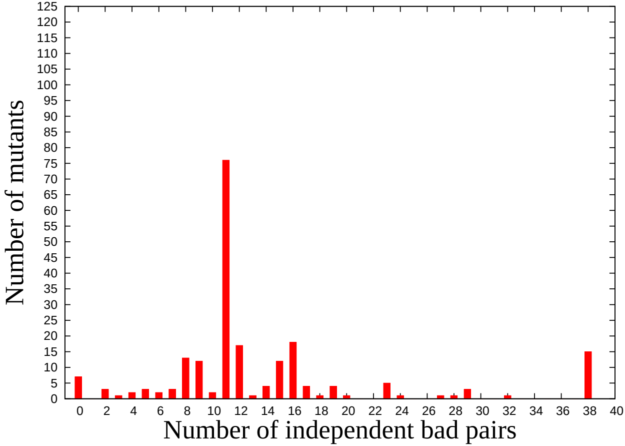
Triangle: source code

```
public static String triangle(
    int side1, int side2, int side3)
{
    int triang;
    if (side1 <= 0 || side2 <= 0 || side3 <= 0) {
        return "illegal";
    }
    triang = 0;

    if (side1 == side2) {
        triang = triang + 1;
    }
    if (side1 == side3) {
        triang = triang + 2;
    }
    if (side2 == side3) {
        triang = triang + 3;
    }

    if (triang == 0) {
        if (side1 + side2 <= side3 ||
            side2 + side3 <= side1 ||
            side1 + side3 <= side2) {
            return "illegal";
        } else {
            return "scalene";
        }
    }

    if (triang > 3) {
        return "equilateral";
    } else if (triang == 1 && side1 + side2 > side3) {
        return "isosceles";
    } else if (triang == 2 && side1 + side3 > side2) {
        return "isosceles";
    } else if (triang == 3 && side2 + side3 > side1) {
        return "isosceles";
    }

    return "illegal";
}
```

Triangle: Single Mutation Results



Triangle: Double Mutation Results: ROR



TCAS Experiments

- Gold code: TCAS program, hand-translated to Java
  - Seven functions; 109 LOC
- Input table
  - input parameters (12): values selected with equivalence partitioning
  - cross product: roughly 6 million elements
  - selected inputs (34): two-cover of the cross product
- Mutants (250): single mutation only
- Execution pseudocode: identical to Triangle

Network Vulnerability Testing: Case Study

- Gold code: object code in a programmable logic controller
  - no information on source code: language, size or structure
- Test table
  - packet capture file with 4734 IP packets
  - input parameters (11): field values extracted from IP header
  - two of the IP packets caused failures
- Which pairs in the two packets caused the failures?
  - each failed packet contains a single independent bad pair:
    1. (protocol:TCP, total length:0)
    2. (protocol:ICMP, total length:0)

Graph-based algorithms for bad pairs generation

- *Error-locating test case*: a test case containing exactly one independent bad pair
- *Error-locating array*: an input table containing at least one error-locating test case for each parameter pair

Conclusions

- Open questions
  - What is a useful formalization of *bad pair*?
  - How common are bad pairs?
  - What is the effect of input selection on bad pairs?
  - What is the relationship between faults and bad pairs?
- Triangle experiments
- TCAS Experiments
- Case study: industrial network vulnerability testing
- Powerful new theory result: error-locating arrays