Conditional Entropy and Failed Error Propagation in Software Testing

Rob Hierons
Brunel University London

Joint work with: Kelly Androutsopoulos, David Clark, Haitao Dan, Mark Harman

White-box testing

- White box testing is:
  - testing based on the structure of the code.

- Good at finding certain classes of faults (e.g. extra special cases) but poor at finding others (e.g. missing special cases).

Example: where white-box testing helps

- Suppose someone implements the absolute value function as:

```
if (x>0) return x; else if (x==-12)
  return 5; else return -x;
```

- Without seeing the code we have no reason to believe that the value -12 is special.
- These kind of cases are only likely to be found with white-box testing

Coverage

- We look at certain types of constructs (e.g. statements, branches)
- We might then:
  - measure the proportion of these that are executed/covered in testing; or
  - insist that in testing we achieve at least a certain percentage coverage (often 100%).

Code coverage criteria

- The most widely used type of test criterion.
- Examples include:
  - Statement Coverage
  - Branch Coverage
  - MC/DC
  - Path Coverage
  - Dataflow based criteria

Motivation

- Mandated in some standards (e.g. automotive, avionics).
- Failing to achieve coverage clearly demonstrates that testing is weak.
- However, it is syntactic: what does achieving coverage tell us?

Finding Faults

- To find a fault in statement *s* a test case must:
  - Execute *s*.
  - Infect *s*.
  - Propagate this to output.
- (The PIE framework.)

Propagation and dependence

- For a difference in x at statement s to be observed we require:
  - The output depends on the value of x at s
- Examples where does not hold:
  - x=f(…); … x=1; …
  - x=f(…); z=g(x); return(y);
  - z=1; x=f(…); if (z<0) y=g(x); return(y);

Propagation

- Dependence is necessary but not sufficient:
  - Consider e.g. statement y = x mod 2;
  - The expected value of x is 7
  - The actual value of x is 3248943

  - There is dependence but no propagation

Failed Error Propagation (FEP)

- This occurs when:
  - A test case leads to execution and infection but not propagation.
- Makes testing less effective.
- Empirical evidence suggests:
  - Affects approximately 10% of test cases but this can be as high as 60% for some programs.

FEP and Coverage

- The ‘hope’ in coverage is that:
  - If a test case executes e.g. a statement s and this contains a fault then the test case will find this fault.
- This already looks weak (need ‘infection’).
  - Also need to avoid FEP.
- Could help explain evidence of limited effectiveness of coverage.

Failed Error Propagation (FEP)

The basic idea

- In test execution FEP occurs through the following:
  - The program state at statement s should be σ but is σ'.
  - The code after this maps σ and σ' to the *same* output.
- There has been a *loss of information*.

- Underlying assumption: only one fault.

Shannon Entropy

- Context:
  - A message is sent from a transmitter to a receiver through a channel.
  - Messages can be modified by the channel.
  - The receiver tries to infer the message sent by the transmitter.
- Shannon entropy is the expected value of the information that can be inferred about the message.

Shannon Entropy

- Given random variable X with probability distribution p, the Shannon Entropy is

$$\mathcal{H}(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$

- This is a measure of the information content (or entropy) of X.
- Basic idea: rare events provide more information but are less likely.

Extreme cases

- If a random variable X has only one possible value:
  - Shannon entropy is 0
    - No information
    - The value of X does not ‘tell us anything’
- Uniform distribution (all values are equiprobable), with n values
  - Shannon entropy is log(n) – number of bits required to represent X.

Squeeziness

- This is the loss of entropy (uncertainty) during computation.
- For function f with input domain *I* this is:

$$Sq(f, I) = \mathcal{H}(I) - \mathcal{H}(O)$$

- Where

$$\mathcal{H}(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$

Another representation

- Given function f on *I*:

$$Sq(f, I) = \sum_{o \in O} p(o)\ \mathcal{H}(f^{-1} o)$$

Extreme cases

- Recall $$Sq(f, I) = \mathcal{H}(I) - \mathcal{H}(O)$$
- If all inputs are mapped to the same output
  - The entropy of the output is zero.
  - All information lost.
  - The output tells us nothing about the input.
- The function *f* is a bijection
  - Squeeziness of 0: no loss of information.
  - The output uniquely identifies the input.

A model of FEP

- FEP happens after the fault.
- Suppose the program follows path

$$\pi = \pi_u \pi_l$$

- Where π_l is the lower path that follows the fault.

- One can argue that FEP occurs due to π_l

A first measure

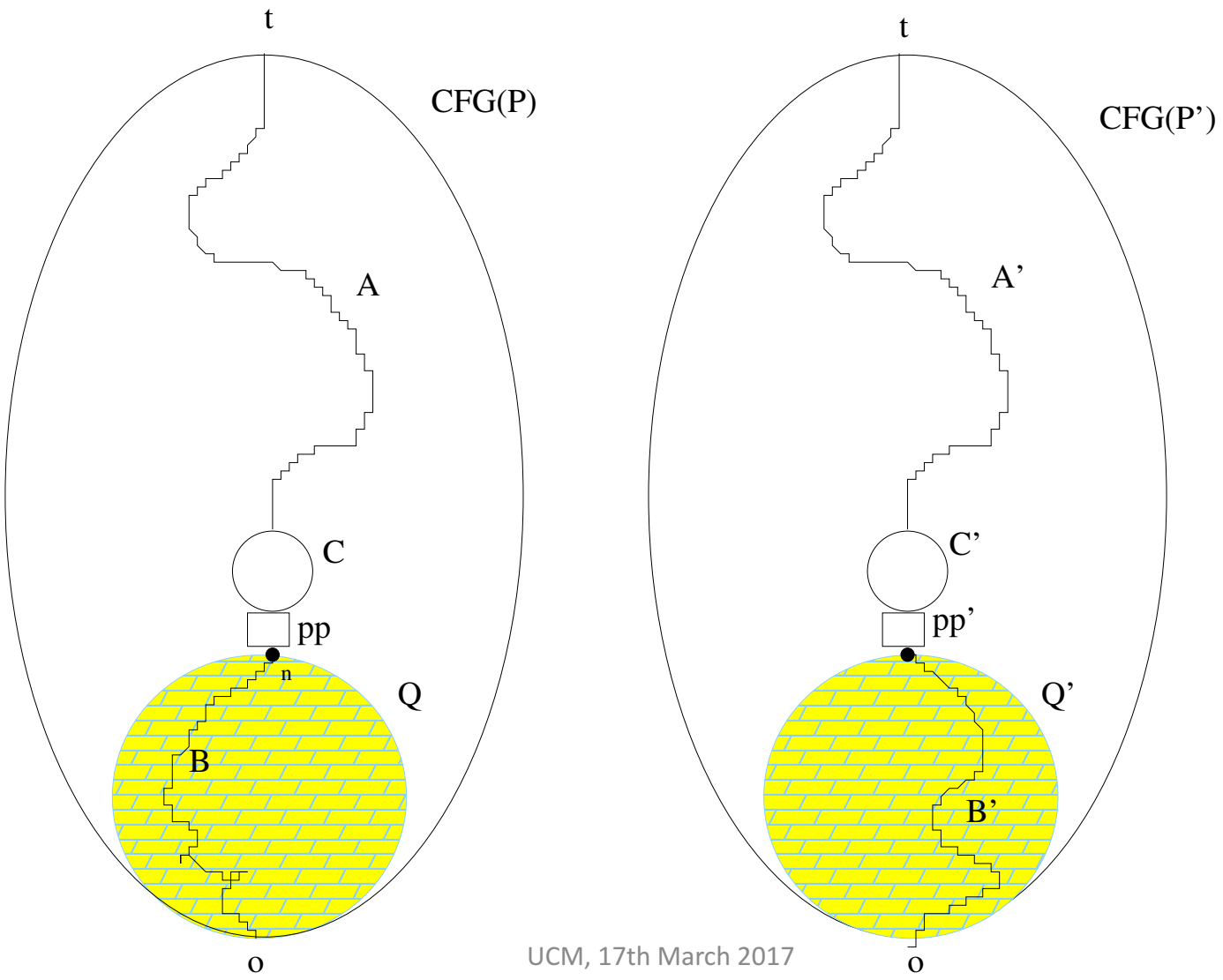
- FEP occurs after the fault.
- The code executed is π_l

- Simply use the Squeeziness of π_l

A complication

- Any FEP involves two programs:
  - A ‘ghost’ (correct) program P
  - The actual program P’
- We assume there is a single fault in a component:
  - Component C of P
  - Corresponding component C’ of P’.

- FEP is not just about P’ or π_l (P might follow a different path).


t
CFG(P)
A
C
pp
n
Q
B
o
t
CFG(P')
A'
C'
pp'
Q'
B'
o

Estimating the probability of FEP

- Using test case t, FEP is caused by a lack of information flow after a fault (in statement s).
- We could use:
  - The Squeeziness of the code that follows s.
    - The QIF of Q; or
    - The QIF of the path.
- The former captures the computation; the latter might approximate this.

- Should we consider the code *before* s?

Possible measures

- M1: Squeeziness of Q (on the states at pp')
- M2: M1 + Squeeziness of R (code before)
- M3: Squeeziness of Q on states reachable via a given upper path π_u
- M4: M3 + Squeeziness of (upper/initial) path π_u
- M5: Squeeziness of (lower/final) path π_l

Experimental study

- For a program p we:
  - Randomly generated a sample T of 5,000 inputs from a suitable domain.
  - Generated mutants of p.
  - For mutant m (mutated statement s), input t in T:
    - Determine whether m and p have the same state after s.
    - Determine whether m and p have the same output.
  - A different ‘outcome’ denotes FEP.

Comparison made

- We compared our measures with the true (for the sample) probability of FEP:

$$p(FEP) = \frac{\#\text{tests with different state after s but same output}}{\#\text{number of tests with different state after s}}$$

Experimental subjects

- Three groups, all written in C:
  - 17 toy programs.
  - 10 functions from R.
  - 3 functions from GRETL (Gnu Regression, Econometrics and Time-Series Library).
- R functions: between 137 and 2397 LOC.
- GRETL functions: between 270 and 688 LOC.

Results: all programs

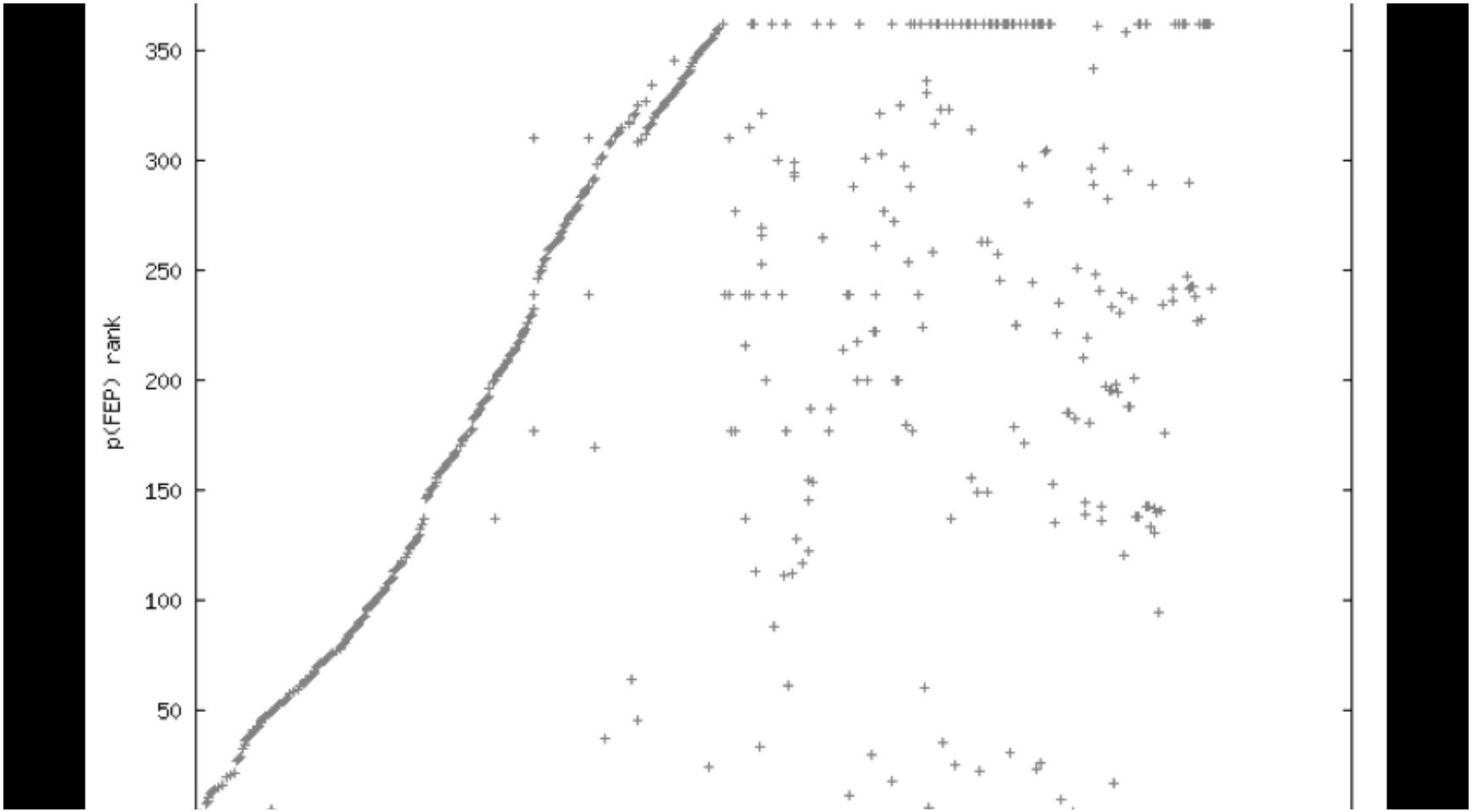
- Rank correlations:

Experiment	**Correlation**
EXP1	0.715267
EXP2	0.699165
EXP3	0.955647
EXP4	0.948299
EXP5	0.031510

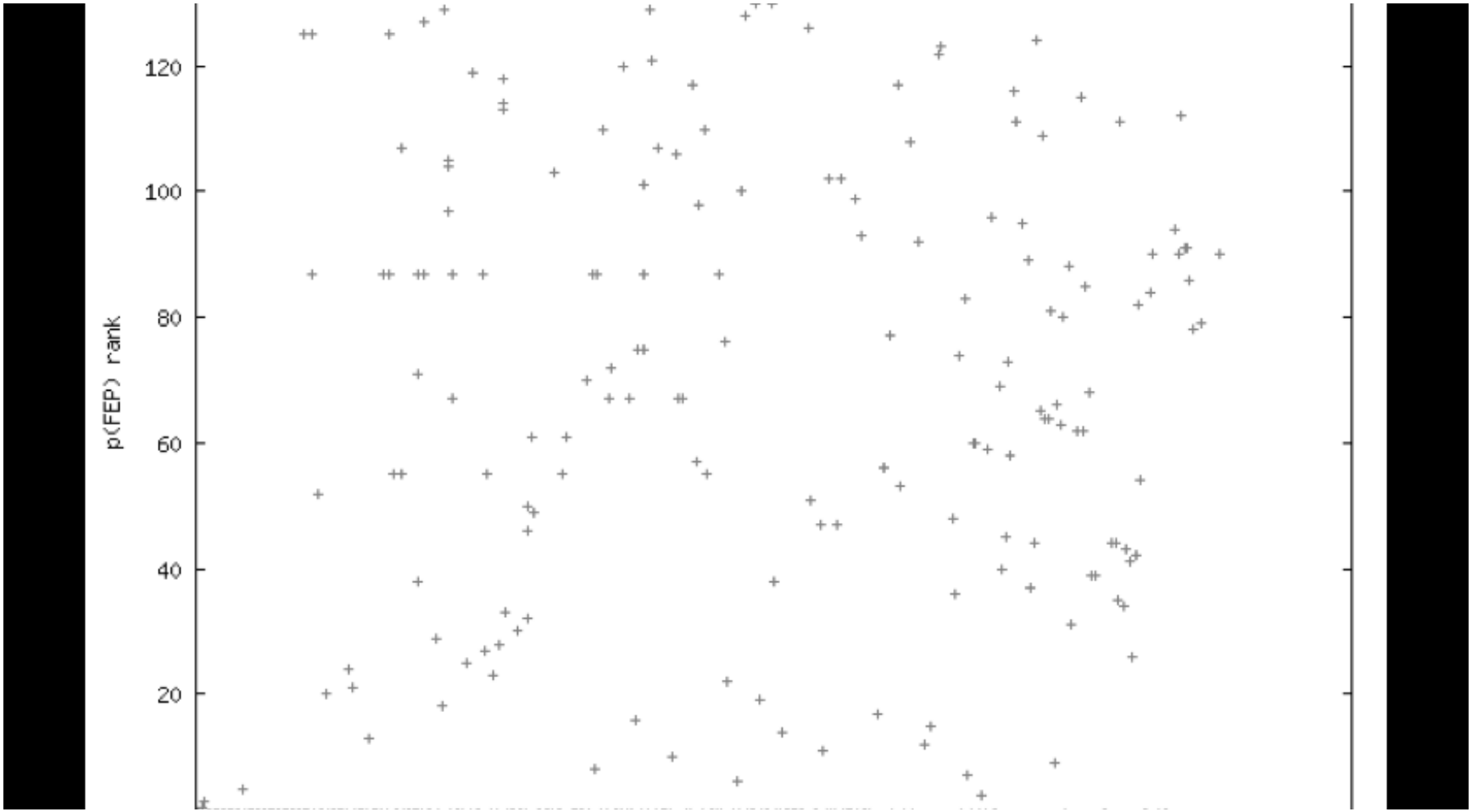
Results – real programs

Experiment	**Correlation**
EXP1	0.974459
EXP2	0.974459
EXP3	0.998526
EXP4	0.998526
EXP5	-0.001361

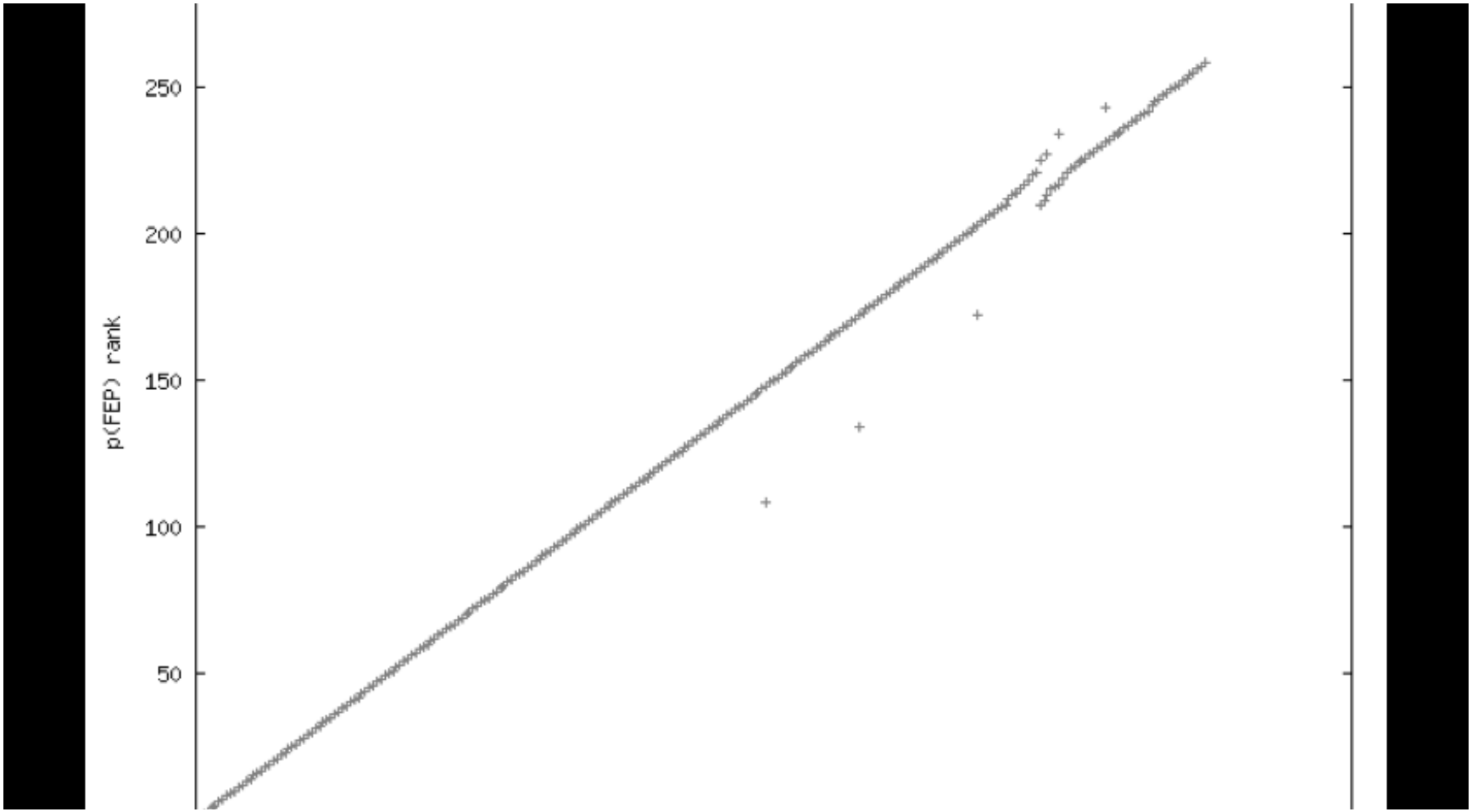
All programs (M2)



Toy programs (M2)



Real programs (M2)



Consequences

- Potential to use Information Theory based measures to predict the likelihood of FEP.
- In practice we might:
  - Use as measure of testability (help us to decide e.g. how many tests cases to use?).
  - Try to cover e.g. a statement s with a test that follows it with code that has low FEP.
  - Have more test cases for ‘hard to test’ parts of the code.

References

- The work is contained in:
  - D. Clark and R. M. Hierons, Squeeziness: An Information Theoretic Measure for Avoiding Fault Masking, *Information Processing Letters*, 112, pp. 335-340, 2012.
  - K. Androutsopoulos, D. Clark, H. Dan, R. M. Hierons, and M. Harman: An Analysis of the Relationship between Conditional Entropy and Failed Error Propagation in Software Testing, *36th International Conference on Software Engineering (ICSE 2014)*.

Other possible uses of Information Theory

Feasibility

- A path has an associated path condition c(π)
  - A predicate on inputs: c(π) is satisfied by an input if and only if the input leads to π being followed.

- A path π is feasible if
  - One or more input values satisfy c(π).

More on feasibility

- Test generation techniques can waste time in trying to generate test input for infeasible path.

- Observations:
  - An infeasible path has no information flow.
  - A path with no information flow is either infeasible or maps all values to one state.

Research Question

- Can we use Squeeziness to address feasibility?

Diversity

- A test suite is diverse if:
  - The test cases are quite ‘different’.

- There is evidence that diverse test suites tend to be effective.

More on diversity

- Easy to see how we measure this for numbers.
- What about:
  - Strings
  - Data structures
  - …
- This has limited use of diversity in testing.

Complexity

- Which are more complex?
  - xyxyxyxyxyxyxyz
  - xyzabcxyzabcpq
  - Fdo3ewr0esr9w2
  - xxxxxxxxxxxxxx
- How about?
  - {xy, xyxyz, sxyxyxyxy}

Kolmogorov Complexity

- Given an object, the Kolmogorov complexity of that object is:
  - The length of the shortest computer program that can generate that object.

- This provides a measure of the complexity of the objects.

Using KC

- A low Kolmogorov Complexity indicates repetition.
  - A computer program could have functions that generate the repeated elements.

- Can be used as a measure of diversity.
- In practice use:
  - The degree to which the object (set of test cases) can be compressed.

Potential uses

- With a more general measure of diversity we can:
  - Assess how diverse test suites are
  - Generate highly diverse test suites
    - Use search-based techniques?

Oracle placement

- We might insert oracles into the code:
  - These provide information about the program state.

- Commonly used in debugging.
- Help also in testing.

What do oracles achieve?

- They:
  - extend the output – we also get values from these oracles.
- This potentially:
  - increases information flow to output;
  - helps avoid FEP.

Where best to place oracles?

- We might want to minimise potential for FEP.
- However, not so simple:
  - We could just place oracles at all program statements.
  - Not practical but will eliminate FEP.

Future plans

- Explore the use of Information Theory in Testing.
- Develop methods for estimating Information Theory based metrics.
- Implement and integrate into automated test generation tools.

Questions?