A Model for Ranking-based Software Diagnosis

Lee Naish
Hua Jie (Jason) Lee
Computer Science and Software Engineering
University of Melbourne

Outline

Background: Diagnosis using Program Spectra

A model buggy program

Multisets of execution paths

Results using the model

Results using the Siemens test suite

Conclusion

Background: Ranking-based Software Diagnosis

Basic idea:

Execute the program multiple times using a test suite where we can tell if each result is correct or not, gathering data about each execution

For each statement/basic block/..., estimate how likely it is to be buggy based on the data gathered

Rank the statements accordingly, then check the code manually, starting with the highest ranked statement until the bug is found (or we give up)

Background: Using program spectra

Collect data on which statements are executed for each test

For each statement we count the number of

1. passed tests in which it was not executed, a_{np},
2. failed tests in which it was not executed, a_{nf},
3. passed tests in which it was executed, a_{ep}, and
4. failed tests in which it was executed, a_{ef}

Ie, true and false negatives and positives (normally 0/1 is used instead of n/e and p/f)

Background: Using program spectra

	Raw Data					Summary			
	T_1	T_2	T_3	T_4	T_5	a_{np}	a_{nf}	a_{ep}	a_{ef}
S_1	e	**e**	e	e	**e**	0	0	3	2
S_2	n	**e**	e	n	**n**	2	1	1	1
S_3	n	**n**	e	e	**n**	1	2	2	0
S_4	e	**e**	n	n	**e**	2	0	1	2
Passed?	p	**f**	p	p	**f**				

Apply some metric $m(a_{np}, a_{nf}, a_{ep}, a_{ef})$ and rank statements according to the result

Eg, a high a_{ef} and low a_{ep} may give a high rank

Several metrics

Jaccard:

$$\frac{a_{ef}}{a_{ef} + a_{nf} + a_{ep}}$$

Anderberg:

$$\frac{a_{ef}}{a_{ef} + 2(a_{nf} + a_{ep})}$$

Sørensen-Dice:

$$\frac{2a_{ef}}{2a_{ef} + a_{nf} + a_{ep}}$$

Dice:

$$\frac{2a_{ef}}{a_{ef} + a_{nf} + a_{ep}}$$

Actually quite a few

Russell and Rao:

$$\frac{a_{ef}}{a_{ef} + a_{nf} + a_{ep} + a_{np}}$$

Simple Matching:

$$\frac{a_{ef} + a_{np}}{a_{ef} + a_{nf} + a_{ep} + a_{np}}$$

Rogers and Tanimoto:

$$\frac{a_{ef} + a_{np}}{a_{ef} + a_{np} + 2(a_{nf} + a_{ep})}$$

Tarantula:

$$\frac{\frac{a_{ef}}{a_{ef}+a_{nf}}}{\frac{a_{ef}}{a_{ef}+a_{nf}} + \frac{a_{ep}}{a_{ep}+a_{np}}}$$

Lots, in fact

Ample:

$$\left| \frac{a_{ef}}{a_{ef} + a_{nf}} - \frac{a_{ep}}{a_{ep} + a_{np}} \right|$$

Overlap:

$$\frac{a_{ef}}{\min\left(a_{ef}, a_{nf}, a_{ep}\right)}$$

Ochiai ($\equiv$ Cosine):

$$\frac{a_{ef}}{\sqrt{\left(a_{ef} + a_{nf}\right) * \left(a_{ef} + a_{ep}\right)}}$$

Ochiai2:

$$\frac{a_{ef} a_{np}}{\sqrt{\left(a_{ef} + a_{nf}\right) * \left(a_{ef} + a_{ep}\right) * \left(a_{np} + a_{nf}\right) * \left(a_{np} + a_{ep}\right)}}$$

And they keep coming...

Wong1: a_{ef}

Wong2: $a_{ef} - a_{ep}$

Wong3:

$$a_{ef} - h, \text{ where } h = \begin{cases} a_{ep} & \text{if } a_{ep} \leq 2 \\ 2 + 0.1(a_{ep} - 2) & \text{if } 2 < a_{ep} \leq 10 \\ 2.8 + 0.001(a_{ep} - 10) & \text{if } a_{ep} > 10 \end{cases}$$

Zoltar:

$$\frac{a_{ef}}{a_{ef} + a_{nf} + a_{ep} + \frac{10000 a_{nf} a_{ep}}{a_{ef}}}$$

Using Predicates

Instead of gathering data on which statements are executed, some systems identify predicates associated with code

Data on which predicates are "observed" and which are true in each passed and failed test is collected

Predicates allow domain knowledge to be incorporated

So does adding if-then-else with no-ops — predicates are not more expressive w.r.t. the data gathered

CBI Increase metric (special case):

$$\frac{a_{ef}}{a_{ef} + a_{ep}} - \frac{a_{ef} + a_{nf}}{a_{ef} + a_{nf} + a_{np} + a_{ep}}$$

And variations...

A model program — *ITE2*

```
if (t1())
    s1();          /* S1 */
else
    s2();          /* S2 */
if (t2())
    x = True;      /* S3 */
else
    x = t3();      /* S4 - BUG */
```

The intention is that `x` should be assigned `True`

S4 (or `t2`) is the "signal"

`t3` attenuates the signal

`t1` (*S1* and *S2*) is "noise"

Execution paths, Performance

We don't care about the details of the program or an input, just what execution path is used

We use two execution paths for *S4* to distinguish failed and passed executions

For symmetry we also use two paths for *S3*, *S1* and *S2*

There are thus 16 paths in total; 8 use *S4* and 4 of those are failed

Given t test cases/execution paths, generate the spectra, apply the metric(s) and rank the statements

The performance of a metric can be measured by how often *S4* is ranked highest

Multisets of execution paths

Given p paths and t tests the number of multisets of paths is

$$f(t,p) = \sum_{i=0}^{t} f(i,p-1)$$

This defines the Binomial numbers; $f(t,p) = C(t+p-1,t)$ where

$$C(n,k) = \frac{n!}{k! \times (n-k)!}$$

The multiset can be represented as a seqence of p numbers which sum to t, or a number with up to p digits (ordering the multisets)

Numbers 1 to $C(t+p-1,t)$ can be mapped to a these multisets

Details left as an exercise (for Algorithms and Data Structures students)

Multisets of execution paths

Two tests: $C(17, 15) = 136$ (only 58 have a failed test)

00000000000002
00000000000011
00000000000020
...

The 100^{th} number where the digits sum to 2 is 10000100000000 (see Online Encyclopedia of Integer Sequences)

Ten tests: $C(25, 15) = 3268760$ (99.98% have a failed test), the two-millionth (base > 10) number where the digits sum to 10 is 1000002003020020

A thousand tests:
$C(1015, 15) = 861684993367430042755986529814326$

We generate random numbers to sample the space...

Optimal metric

A metric is optimal if it maximises the number of points for which $S4$ is ranked highest (we also take account of ties)

Optimal metric (for any number of tests):

$$\begin{cases} -1 & \text{if } a_{nf} > 0 \\ a_{np} & \text{otherwise} \end{cases}$$

Key parts of the proof are

- There is a single bug, so for the buggy statement, $a_{nf} = 0$
- $S1$ is executed iff $S2$ is not executed
- $\forall y \ f(y+1,4)f(y,2) \geq f(y,4)f(y+1,2)$

Results using the model

Num. of tests	2	5	10	20	50	100	500	1000
Optimal	60.35	74.17	84.31	92.31	97.91	99.36	99.97	99.99
Zoltar	60.35	73.12	83.52	92.17	97.91	99.36	99.97	99.99
Wong3	56.90	64.39	79.84	91.88	97.91	99.36	99.97	99.99
Binary	53.45	62.29	72.88	84.56	95.14	98.41	99.92	99.98
Russell	53.45	62.29	72.88	84.56	95.14	98.41	99.92	99.98
Overlap	35.36	53.51	70.55	84.28	95.13	98.41	99.92	99.98
Ochiai	60.35	73.12	82.76	90.38	95.76	97.34	98.37	98.46
Ample2	56.90	69.55	80.57	88.86	94.84	96.59	97.75	97.86
Jaccard	60.35	73.12	81.84	88.77	93.98	95.69	96.94	97.07
Anderberg	60.35	73.12	81.84	88.77	93.98	95.69	96.94	97.07
Sørensen	60.35	73.12	81.84	88.77	93.98	95.69	96.94	97.07
Dice	60.35	73.12	81.84	88.77	93.98	95.69	96.94	97.07
Ochiai2	49.14	69.23	79.12	87.11	92.86	94.76	96.17	96.33
Tarantula	55.75	63.49	73.80	82.42	89.09	91.43	93.26	93.48
CBI	55.75	63.49	73.80	82.42	89.09	91.43	93.26	93.48
CBI log	25.00	50.08	67.23	80.58	88.42	90.73	93.16	93.51
CBI sqrt	25.00	46.53	64.76	79.38	87.98	90.26	92.29	92.58
Rogers	56.90	64.39	71.13	77.28	82.99	85.35	87.38	87.61
SimpleM	56.90	64.39	71.13	77.28	82.99	85.35	87.38	87.61
Ample	36.21	42.05	48.07	45.28	47.58	49.89	49.25	49.15

Equivalence of metrics

Jaccard, Anderberg, Sørensen-Dice and Dice are equivalent

Rogers and Tanimoto, Simple Matching and Wong2 are equivalent

Tarantula and CBI Increase with constant "Context" are equivalent

Russell and Rao and Wong1 are equivalent; Binary is also equivalent if there is a single bug

Bug consistency (100 tests); $q_e = \frac{a_{ef}}{a_{ef}+a_{ep}}$

q_e range	$\leq$0.05	.05–.1	.1–.2	.2–.5	.5–.9	0.9–1
% of cases	0.13	0.61	4.22	46.11	48.20	0.71
Opt, Zoltar, W3	74.19	89.25	96.12	99.28	99.90	100.00
Russell, Overlap	61.79	82.70	93.22	98.20	99.36	99.61
Ochiai	73.44	85.34	90.59	96.14	99.25	100.00
Ample2	73.08	84.74	89.83	95.14	98.73	99.99
Jaccard	70.25	78.60	84.24	93.60	98.92	100.00
Ochiai2	70.34	78.49	83.26	92.14	98.48	100.00
Tarantula	69.81	77.00	80.68	88.09	95.69	99.87
CBI log	47.92	76.22	80.99	88.32	94.10	96.81
CBI sqrt	45.97	75.99	80.96	88.30	93.22	94.86
Rogers	37.36	38.33	45.26	76.89	97.45	100.00

Proportion of failed tests (100 tests)

Failure range	$\leq$.05	.05–.1	.1–.2	.2–.5	.5–1
% of cases	1.63	6.89	29.77	59.45	2.24
Opt, Zoltar, W3	93.27	97.85	99.23	99.75	99.93
Russell, Overlap	83.14	94.67	98.09	99.39	99.81
Ochiai	92.07	95.51	96.80	97.93	99.06
Ample2	92.18	95.60	96.51	96.87	96.63
Jaccard	89.54	92.54	94.52	96.71	98.78
Ochiai2	90.01	92.60	93.94	95.48	97.53
Tarantula	88.88	90.87	91.50	91.56	90.81
CBI log	86.92	91.11	91.82	90.36	88.05
CBI sqrt	86.64	91.09	91.90	89.65	85.69
Rogers	68.21	72.52	79.58	89.69	97.64

Proportion of bug executions (100 tests)

$S4$ exec range	$\leq$0.1	.1–.3	.3–.5	.5–.7	.7–.9	0.9–1
% of cases	0.03	6.96	44.45	42.60	5.94	0.02
Opt, Zoltar, W3	100.00	99.86	99.54	99.14	99.02	99.15
Russell, Overlap	82.99	95.74	98.16	99.00	99.37	99.58
Ochiai	100.00	99.82	99.00	96.24	89.99	78.71
Ample2	100.00	99.86	99.20	95.50	81.15	37.18
Jaccard	100.00	99.76	98.46	93.80	83.79	68.38
Ochiai2	100.00	99.80	98.58	92.80	74.58	26.31
Tarantula	100.00	99.65	97.35	87.81	63.79	21.36
CBI log	94.56	99.37	96.77	86.88	63.39	21.53
CBI sqrt	94.56	99.31	96.38	86.26	63.03	21.40
Rogers	99.99	99.26	94.29	78.12	53.91	32.76

Comparison of metrics

Knowing whats optimal makes comparison much easier

P = number of passed tests, F = number of fails

Metric(s)	Equivalent Formula
Optimal'	$a_{ef} - \frac{1}{P+1} a_{ep}$
Russell and Rao, Wong1, Binary	a_{ef}
Simple M, Rogers and Tanimoto, Wong2	$a_{ef} - a_{ep}$
Ample2	$a_{ef} - \frac{F}{P} a_{ep}$
Ample	$\left\lvert a_{ef} - \frac{F}{P} a_{ep} \right\rvert$
Optimal"	$\log a_{ef} - \log (a_{ep} + P.F)$
Tarantula, Increase	$\log a_{ef} - \log a_{ep}$
Jaccard, Anderberg, Sørensen-Dice, Dice	$\log a_{ef} - \log (a_{ep} + F)$
Ochiai	$\log a_{ef} - \frac{1}{2} \log (a_{ep} + a_{ef})$

Siemens test suite

Standard suite for evaluating diagnosis

C programs with "single" bugs deliberately added

Program	Versions	LOC	Tests	Description
tcas	41	173	1578	altitude separation
schedule	9	410	2650	priority scheduler
schedule2	10	307	2680	priority scheduler
print_tokens	7	563	4056	lexical analyser
print_tokens2	10	508	4071	lexical analyser
replace	32	563	5542	pattern recognition
tot_info	23	406	1054	information measure

Performance using Siemens test suite

Two common ways to measure performance of a diagnosis method for a given program:

- How much of the program must be examined to find a bug?
- Can a bug be found by examining $\leq 10\%$ of the program?

We aggregate these measures over programs in the test suite

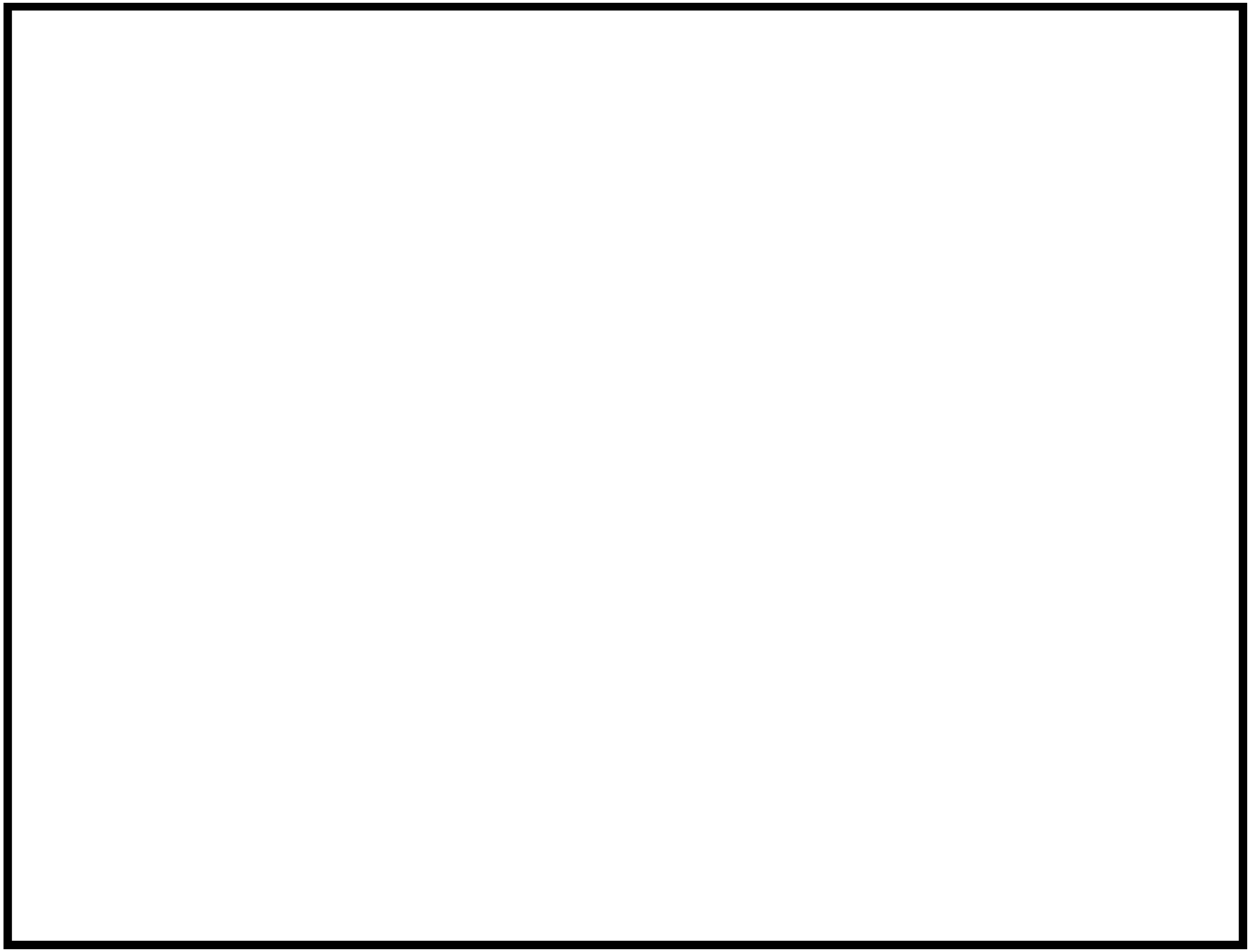
We (initally) eliminate programs with

- no failed tests,
- runtime errors, or
- multiple bugs,

retaining 112 out of the 132 programs

Siemens test suite results (112 programs)

Metric	Tcas	Sch	Sh2	PTok	PT2	Repl	TInf	Ave.
Opt.	12.66	0.85	17.57	0.71	11.94	2.74	1.80	6.89
Zoltar	12.66	0.85	17.57	0.71	11.94	2.76	1.82	6.90
Ochiai	12.62	1.34	20.70	0.71	14.16	4.63	2.51	8.10
Amp2	13.16	1.46	22.86	0.71	15.21	5.33	2.47	8.74
Jac.	13.21	2.25	22.94	2.40	15.77	5.57	3.56	9.39
Wong3	19.62	0.85	24.98	0.71	11.94	6.48	4.06	9.81
Taran.	13.25	2.43	22.94	4.26	15.93	6.37	3.76	9.85
CBI s	14.18	2.55	22.94	3.82	15.93	6.37	3.76	9.93
CBI l	15.04	2.43	22.94	3.99	15.93	6.37	3.75	10.06
Och2	14.52	2.43	25.58	0.80	15.69	8.66	3.15	10.12
Ample	14.31	1.46	27.46	0.71	15.53	7.44	4.59	10.21
Rus.	16.39	12.99	15.13	7.46	20.81	7.21	7.90	12.55
O-lap	18.45	23.01	21.69	18.56	41.17	11.04	21.59	22.22
Rog.	81.96	68.73	87.42	74.96	53.48	72.10	69.66	72.62



Siemens test suite results (112 programs)

Metric	Bugs	%
Optimal, Zoltar	76	67.86
Ochiai	72	64.29
Wong3	71	63.39
Ample2, Jaccard	69	61.61
CBIsqrt, CBIlog	68	60.71
Tarantula	67	59.82
Ochiai2	65	58.04
Ample	62	55.36
Russell	39	34.82
Overlap	13	11.61
Rogers	7	6.25

Siemens test suite results (all programs)

Metric	Bugs	Programs	%
Optimal, Zoltar	78	132	59.09
Ochiai, Ample2	75	132	56.82
Jaccard, Wong3	73	132	55.30
CBI log, CBI sqrt, Tarantula	72	132	54.55
Ochiai2	69	132	52.27
Ample	66	132	50.00
Other systems ...			
Ling Xiao	74	132	56.06
Sober	68	130	52.31
CBI	52	130	40.00
Cause-Transition	34	130	26.15

Conclusion

The *ITE2* program is very simple but results fit quite well with empirical results using the Siemens test suite, even with a uniform distribution

This model-based approach allows many experiments to be done easily

It also allows a more analytical approach, for example, developing optimal metrics

The optimal *ITE2* metrics are far simpler than previously proposed metrics but perform better than all of them on the Siemens test suite

The methodology can be extended to multiple bug programs, etc