An Observation-based Model for Fault Localization

R. Abreu P. Zoeteweij A.J.C. van Gemund

Delft University of Technology

WODA, July 2008

Motivation

Situation

- SW Debugging is overly expensive
- Many debugging tools/approaches
- Model-based (MBD)
- Dynamic/statistics-based (SFL)

Rationale

- MBD needs a model as input which is often not available
- SFL cannot distinguish components with the same execution pattern

In this presentation, a new novel approach...

- Dynamic information to extract a model ▷ inspired by SFL
- Candidates ranked using Bayes' update ▷ inspired by MBD

Outline

Components, Runs, Program Spectra...

- A program under analysis comprises a set of **M** components
  - Statements in the context of this paper
- The program is executed using **N** test cases (runs)
- Component activity is recorded in terms of *program spectra*
  - program spectra = abstractions of program traces
- Program spectra is a set of counter or flags for each component
  - In this presentation, *statement-hit spectra* is used

Observation Matrix

- Row $O_{i\star}$ indicates whether a component was involved in run i
- Column $O_{\star j}$ indicates in which runs component j was involved
- The error vector e indicates whether a run has failed or passed

$$\text{N spectra} \quad O = \overset{\text{M components} \qquad \text{error vector}}{\left[\begin{array}{cccc|c} o_{11} & o_{12} & \cdots & o_{1M} & e_1 \\ o_{21} & o_{22} & \cdots & o_{2M} & e_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ o_{N1} & o_{N2} & \cdots & o_{NM} & e_N \end{array}\right]}$$

- Input to the debugging method is **only** O

Outline

Model Generation

- Compile the observation matrix into propositional logic
  - More specifically, conjunctions of disjunctions
- Suppose the following source code and program spectra

```
    (y1,y2) 3inv(bool x) {
1.    w = !x
2.    y1 = !w;
3.    y2 = w; //fault: ! missing
      return (y1,y2);
    }
```

c_1	c_2	c_3	e	
1	1	0	0	obs_1
1	0	1	1	obs_2
0	1	1	1	obs_3

- Yields the following propositional logic

$$(\neg h_1 \vee \neg h_3) \wedge (\neg h_2 \vee \neg h_3)$$

Solve the Model

- Compute the minimal hitting set
  - NP complete
  - TUDelft heuristic: STACCATO
- The solution for the example's model

$$(\neg h_1 \vee \neg h_3) \wedge (\neg h_2 \vee \neg h_3)$$

$$\Downarrow$$

$$(\neg h_3) \vee (\neg h_1 \wedge \neg h_2)$$

- Thus, either **c_3** is faulty **or** **c_1** and **c_2** are faulty

Ranking Diagnoses

- Set of diagnosis candidates can be large
- Bayes' update to compute probabilities

$$\Pr(d_k|obs) = \frac{\Pr(obs|d_k)}{\Pr(obs)} \cdot \Pr(d_k)$$

where

- $\Pr(d_k) = p^{|d_k|} \cdot (1-p)^{M-|d_k|}$ and, e.g., $p = 0.01$
- $$\Pr(obs|d_k) = \begin{cases} 0 & \text{if SD} \wedge obs \wedge d_k \models \perp \\ 1 & \text{if } d_k \rightarrow obs \wedge \text{SD} \\ \varepsilon & \text{if } d_k \rightarrow \{obs_1 \wedge \text{SD}, \ldots, obs \wedge \text{SD}, \ldots, obs_k \wedge \text{SD}\} \end{cases}$$
- $$\varepsilon = \begin{cases} g(d_k)^t & \text{if run passed} \\ 1 - g(d_k)^t & \text{if run failed} \end{cases}$$

Ranking Diagnoses, ctd'ed

- g estimates the probability that components in d_k produce a correct output

$$g(d_k) = \frac{\sum\limits_{i=1..N} [(\bigvee\limits_{j \in d_k} o_{ij} = 1) \wedge e_i = 0]}{\sum\limits_{i=1..N} [\bigvee\limits_{j \in d_k} o_{ij} = 1]}$$

- Back to our example...

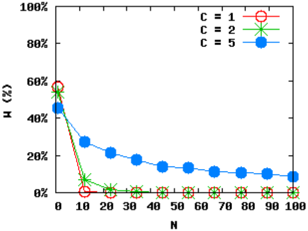
d_k	$\Pr(d_k)$
$\{3\}$	0.995
$\{1,2\}$	0.005

- Meaning that, one would start by inspecting component 3

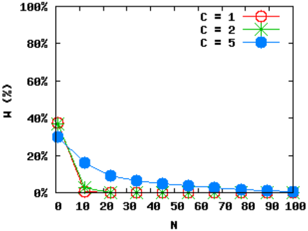
Outline

Experimental Setup

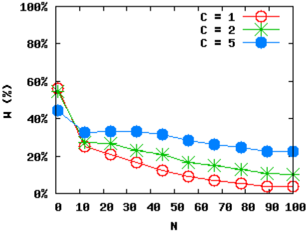
- Study the effects of the following on the diagnostic accuracy
  - Number of Failing Runs
  - Behavior for Small Number of Runs
  - Behavior for Large Number of Runs
- Observation matrices built based on
  - Probability a component is touched r
  - Probability a faulty component fails g
  - Fault cardinality C
- Evaluation Metric: *Wasted Effort*



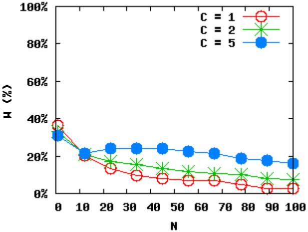
(c) $g = 0.1$ and $r = 0.6$



(d) $g = 0.1$ and $r = 0.4$



(e) $g = 0.9$ and $r = 0.6$



(f) $g = 0.9$ and $r = 0.4$

Optimal N^* for perfect diagnosis ($r = 0.6$)

g	0.1				
C	1	2	3	4	5
N^*	13	31	90	120	250
N_F	5	19	71	111	245

g	0.9				
C	1	2	3	4	5
N^*	200	300	500	1000	1700
N_F	12	36	84	219	459

Outline

Experimental Setup

Programs

- Siemens set of programs
  - 7 programs with several (single fault) faulty versions
  - O(100) LOC
  - O(1000) test cases
- GNU gcov to obtain the observation matrix

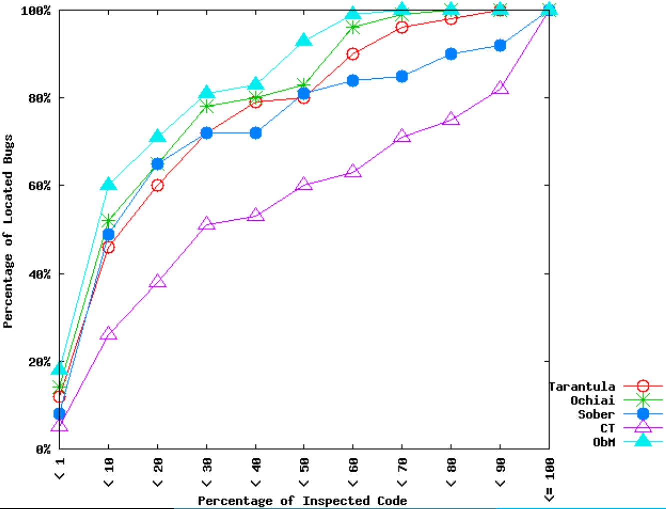
Evaluation Metric

- Percentage of code that needs to be inspected

$$Effort = \frac{\text{position of fault location}}{LOC}$$

Experimental Results

Cumulative Percentage of Located Faults



Outline

Conclusions

- Fault localization approach
  - Uses abstraction of program traces to generate a (dynamic, sub-) model
  - The set of traces for pass/fail executions is used to reason about the observed failures
- Set of candidates also contains multiple-fault explanations
- Theoretically, given sufficient test cases are available, this approach will reveal the true faulty state
- Results using the Siemens set have shown that our approach outperforms other state-of-the-art approaches

Future Work

- Study the diagnostic performance for multiple-fault programs
- Study the possibility of engaging several developers to find the faults
- Reducing the hitting set algorithm complexity
  - STACCATO is under development
- Apply to other (real) programs (e.g., space)

?

For more info:

- http://www.st.ewi.tudelft.nl/~abreu
- email: r.f.abreu@tudelft.nl