Divide by Zero Exception Raising via Branch Coverage

Neelesh Bhattacharya, Abdelilah Sakti, Yann Gael Guehenuc, Giuliano Antoniol, Gilles Pesant

Department of Computer and Software Engineering
Pattern Trace Identification, Detection, and Enhancement in Java Lab
SOftware Cost-effective Change and Evolution Research Lab
École Polytechnique de Montréal, Québec, Canada

Sept 12, 2011

Outline

Motivation

- Consequences of uncaught exceptions may be dire: program crashes or security breaches.
- Uncaught exceptions might also lead to a significant increase in the cost of program testing.
- Historical evidence suggests that poorly-managed exceptions have had severe consequences on human beings or led to great economic losses (1996, Arianne 6 crash, USD 370 million).

$\Rightarrow$ We need to fix this issue.

Transformation (1/2)

Our work follows the work carried out by Tracey et. al [TCM00] on the generation of test data to raise exceptions for integers.

An example

```
int Z, x=4;
if (Z>1 AND Z<=5)
        return Z;
else
        return (x * 4)/(Z-1);
```

Neelesh Bhattacharya, Abdelilah Sakti, Yann Gael Guehenuc, Giuliano Antoniol, Gilles Pesant

Transformation (2/2)

```
int Z, x=4;
if (Z>1 AND Z<=5)
    return z;
else
    if (Z == 1)
        print "Exception raised";
    else
        return (x*4)/(Z-1);
```

Fitness Functions

- Tracey fitness function = *branch distance*
- Improved fitness function:

$$NBD = 1 - 1.001^{-branch\_distance} \quad (1)$$

$$Fitness\ function = Approach\ level + NBD \quad (2)$$

where NBD is the Normalized Branch Distance

Neelesh Bhattacharya, Abdelilah Sakti, Yann Gael Guehenuc, Giuliano Antoniol, Gilles Pesant

Approaches

- Random
- Hill Climbing (and three strategies)
- Simulated Annealing
- Genetic Algorithm
- Constraint Programming

Hill Climbing Strategies (1/3)

HC1:

- The neighbours of the input data are generated as the sum of the current value of the variable with a randomly-generated value drawn from a Gaussian distribution with zero mean and an initial standard deviation (SD).
- If there is no improvement after a given number of moves, the value of the SD is changed to a larger value to expand the neighbourhood and give the algorithm an opportunity to get out of the, possible local optimum.

Hill Climbing Strategies (2/3)

HC2:

- Unlike HC1, to avoid getting "stuck" IN a specific region, HC2 forces the search to take a jump away from unsuccessful neighbourhood in an attempt to move into a more favourable neighbourhood.
- The value of the length of the jump is added to the current value of the variable to aid the movement.
- The length of a jump and the number of jump depends on the search space.

Hill Climbing Strategies (3/3)

HC3:

- As an extension of HC2, the fitness values of the best neighbour of all previously-visited neighbourhoods is stored, before jumping to another neighbourhood.
- After visiting the various regions (HC2), HC3 returns to the best of the regions and continues a local search with a larger value of SD, to explore more of this region.

Research Questions

- **RQ1**: Which of the three proposed hill climbing strategies is best suited to raise a divide-by-zero exception?
- **RQ2**: Which of all the meta-heuristic techniques is best suited to raise a divide-by-zero exception?
- **RQ3**: Which of Tracey's fitness function and the improved fitness function is best suited to raise a divide-by-zero exception?
- **RQ4**: Which of the best-suited meta-heuristic technique and of the CP is best suited to raise a divide-by-zero exception?

Case Studies

- Tracey exemplary code
- GridCanvas Class from Eclipse 2.0.1
- ProcessStats Class from Android 2.0

Neelesh Bhattacharya, Abdelilah Sakti, Yann Gael Guehenuc, Giuliano Antoniol, Gilles Pesant

Choice of Parameters (1/3)

- We varied the domains from $[-100; +100]$ to $[-50,000; +50,000]$ for all the input variables.
- We repeated each computation 20 times to analyse the diversity in the observed values and conduct statistical tests.

Choice of Parameters (2/3)

- **Simulated Annealing**: We varied the initial temperature from as low as 0.5 to 150, α from 0.8 to 0.995, and the numbers of iterations from 10 to 500. Finally we used 20 for initial temperature, 0.99 for α, and 100 for the number of iterations.
- **Genetic Algorithm**: We used single-point cross-over and bit-flip mutation in the *jMetal* framework. A probability of 0.9 for the crossovers, 0.09 for the mutations, and population size of 100 was chosen after several trials.

Choice of Parameters (3/3)

- **HC1**: A value of 10 was chosen as the initial SD, and 100 as the number of iterations till which we check for an improvement.
- **HC2**: The length and number of jumps was varied depending on the input range.
- **HC3**: The final value of SD at the best position was chosen as 35.
- **Stopping Criterion**: All the strategies either continues until they reach the maximum number of iterations or generates test data that fires the targeted exception.

Results (1/5)

Addressing RQ1 and RQ2

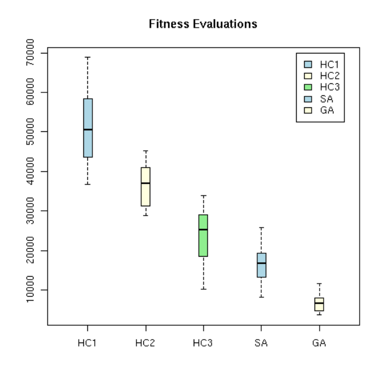

Figure: Tracey code

Results (2/5)

Addressing RQ1 and RQ2

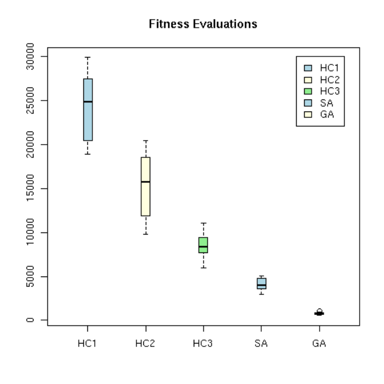

Figure: Eclipse UUT

Results (3/5)

Addressing RQ1 and RQ2

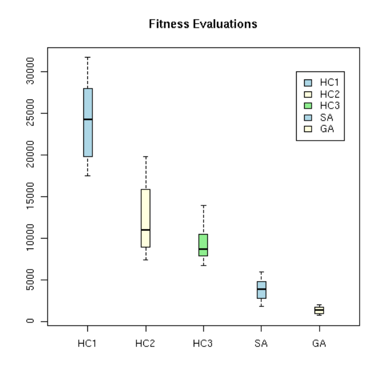

Figure: Android UUT

Results (4/5)

Addressing RQ3

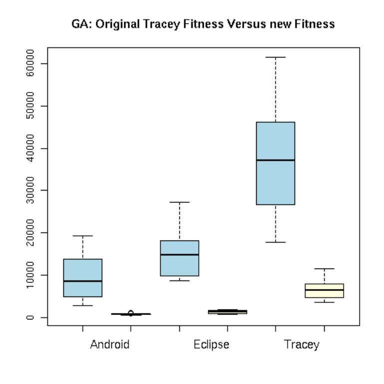

Figure: Tracey Fitness Function vs Improved Fitness Function

Results (5/5)

	Tracey Exemplary Code	Eclipse	Android
GA	8.067/1.439	2.129/1.149	1.926/1.177
CP	1.035/0.0135	0.01/0	0.01/0

Table: Comparison of GA against CP-SST in terms of average execution times (ms) and standard deviations for all UUTs

Conclusion

- We presented a novel testability transformation to generate test input data to raise divide-by-zero exceptions in software systems.
- We compared the performance of hill climbing (three strategies), simulated annealing, genetic algorithm, random search, and constraint programming when using this fitness function.
- We validated our fitness function and compared the search technique on three software units: one synthetic code fragment taken from [1] and two classes extracted from Eclipse and Android, respectively.

Future Work

- Validate our fitness function and choice of search technique with more complex input data types and different types of exceptions.
- Integrate a chaining approach to deal with data dependencies and study the testability transformations required to simplify the generation of test input data to raise exceptions.

Questions?

Thanks for your attention

Neelesh Bhattacharya, Abdelilah Sakti, Yann Gael Guehenuc, Giuliano Antoniol, Gilles Pesant

(1/3)

Comparisons	p-values	Cohen d values
HC1-HC2	9.261e-10	2.55246
HC1-HC3	6.376e-16	5.951003
HC2-HC3	6.868e-08	2.428475
HC3-SA	7.049e-14	4.147889
HC3-GA	2.2e-16	8.223645
SA-GA	8.763e-15	6.254793

Table: Cohen d and p values with Tracey UUT

(2/3)

Comparisons	*p*-values	Cohen *d* values
HC1-HC2	3.245e-10	2.682195
HC1-HC3	7.167e-13	4.111778
HC2-HC3	0.003998	0.9912142
HC3-SA	2.387e-11	3.239916
HC3-GA	1.933e-13	5.258295
SA-GA	9.989e-09	2.694495

Table: Cohen *d* and *p* values with Eclipse UUT

(3/3)

Comparisons	*p*-values	Cohen *d* values
HC1-HC2	1.438e-06	1.894204
HC1-HC3	2.981e-12	3.345377
HC2-HC3	7.438e-08	2.12037
HC3-SA	0.0003169	1.266088
HC3-GA	1.283e-10	3.481401
SA-GA	4.531e-09	2.696728

Table: Cohen *d* and *p* values with Android UUT

Neelesh Bhattacharya, Abdelilah Sakti, Yann Gael Guehenuc, Giuliano Antoniol, Gilles Pesant

N. Tracey, J. Clark, and J. Mcdermid.
Automated test-data generation for exception conditions.
Software - Practice and Experience, 30:61–79, 2000.