Code Coverage Aware Test Generation Using Constraint Solver

Krystof Sykora[1], Bestoun S. Ahmed[2], and Miroslav Bures[1]

[1] Department of Computer Science, Faculty of Electrical Eng, Czech Technical University, Prague, Czech Republic *{sykorkry,buresm3}@fel.cvut.cz*

[2] Department of Mathematics and Computer Science, Karlstad University, Sweden *bestoun@kau.se*

Motivation

Code Coverage Aware Test Generation - CCTG

- innovative automatic test generation (POC)
- combinatorial test generation techniques
- potential improvement for regression testing

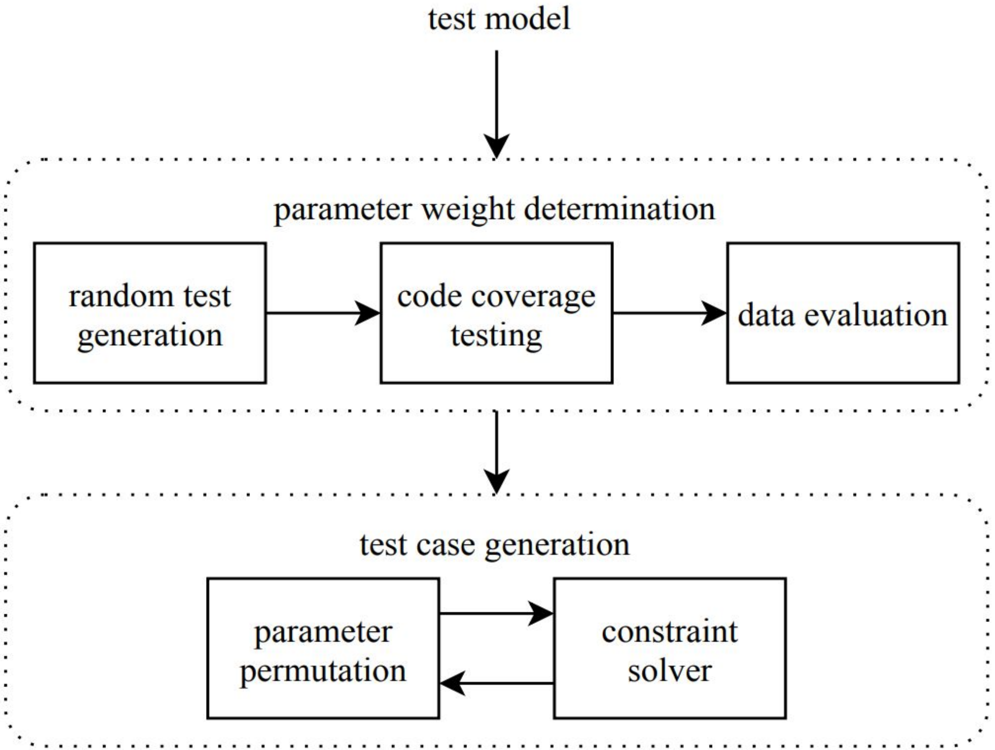
Code-coverage-based method

- created using coverage analysis
- based on parameter weight
- partially random test generation
- adjusted using constraint solver (Z3)

Case Study

- 3 C command line programs
- determine effectiveness of test cases

CCTG in 3 steps



1# model analysis

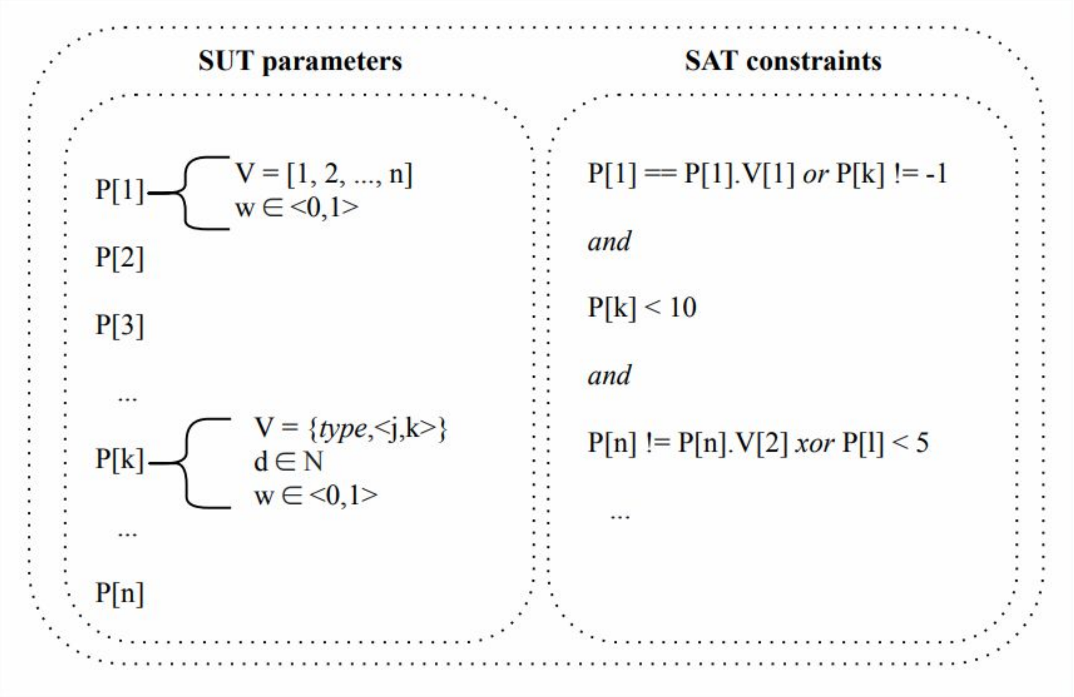
2# coverage testing & evaluation

3# test generation

#1 Test Model

- determine input parameters
- determine input types
- formulate constraints

constraint solver to adjust for false positives (--help)



#2 Combinatorial Coverage Testing

- test generation

semi-random, test depth level

- coverage measurement

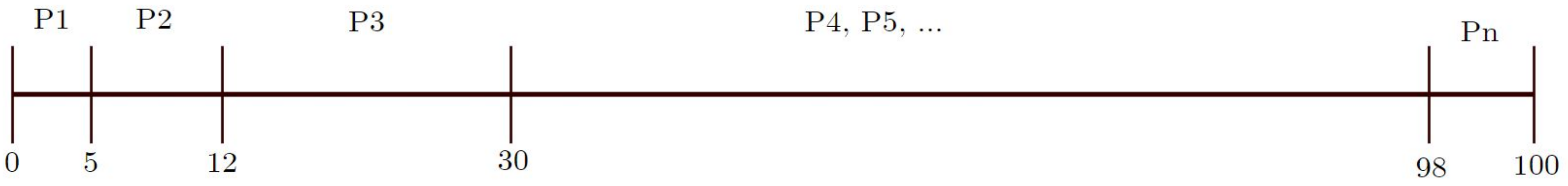
using Gcov tool (modular)

```
TC[1]    = [P[1].RV[1], P[2].RV[1], ... ,P[k].RV[1], ... P[n].RV[1]]
TC[2]    = [P[1].RV[2], P[2].RV[1], ... ,P[k].RV[1], ... P[n].RV[1]]
TC[3]    = [P[1].RV[2], P[2].RV[2], ... ,P[k].RV[1], ... P[n].RV[1]]
.
.
.
TC[k]    = [P[1].RV[2], P[2].RV[2], ... ,P[k].RV[1], ... P[n].RV[1]]
TC[k+1]  = [P[1].RV[2], P[2].RV[2], ... ,P[k].RV[2], ... P[n].RV[1]]
.
.
.
TC[n]    = [P[1].RV[2], P[2].RV[2], ... ,P[k].RV[2], ... P[n].RV[1]]
TC[n+1]  = [P[1].RV[2], P[2].RV[2], ... ,P[k].RV[2], ... P[n].RV[2]]
TC[n+2]  = [P[1].RV[3], P[2].RV[2], ... ,P[k].RV[2], ... P[n].RV[2]]
.
.
.
```

#2 Coverage Evaluation

- interested in change caused by single parameter
- aim to normalise parameter weights on scale

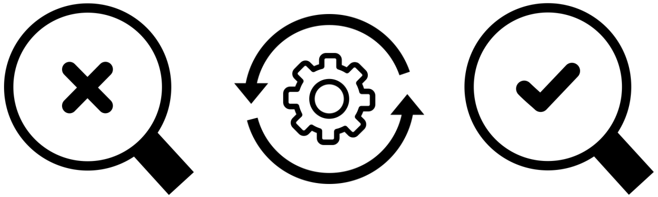
TC[1] = {[P[1].RV[1], P[2].RV[1], ... ,P[k].RV[1], ... P[n].RV[1]], CC}
TC[2] = {[P[1].RV[2], P[2].RV[1], ... ,P[k].RV[1], ... P[n].RV[1]], CC}
TC[3] = {[P[1].RV[2], P[2].RV[2], ... ,P[k].RV[1], ... P[n].RV[1]], CC}
.
.
.
TC[k] = {[P[1].RV[2], P[2].RV[2], ... ,P[k].RV[1], ... P[n].RV[1]], CC}
TC[k+1] = {[P[1].RV[2], P[2].RV[2], ... ,P[k].RV[2], ... P[n].RV[1]], CC}
.
.
.



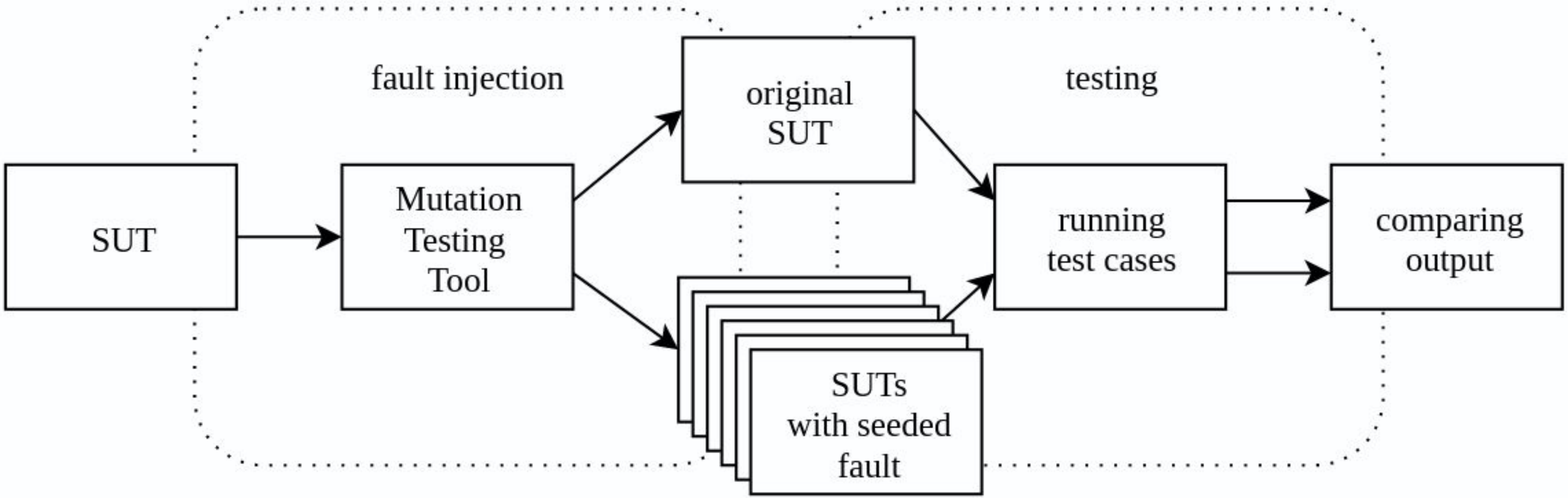
#3 Test Generation

- high weight parameter permutation increased
- constraint solver to remove nonsensical executions
- test count further modulated by input specifications

ai. scalable test suites

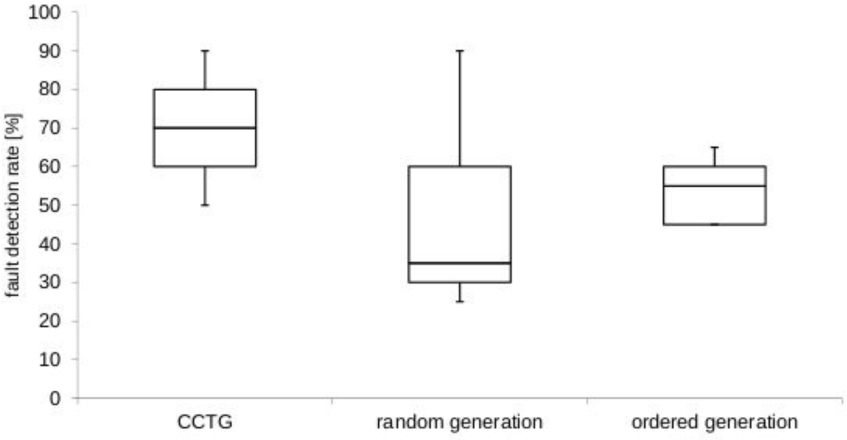


Case Study

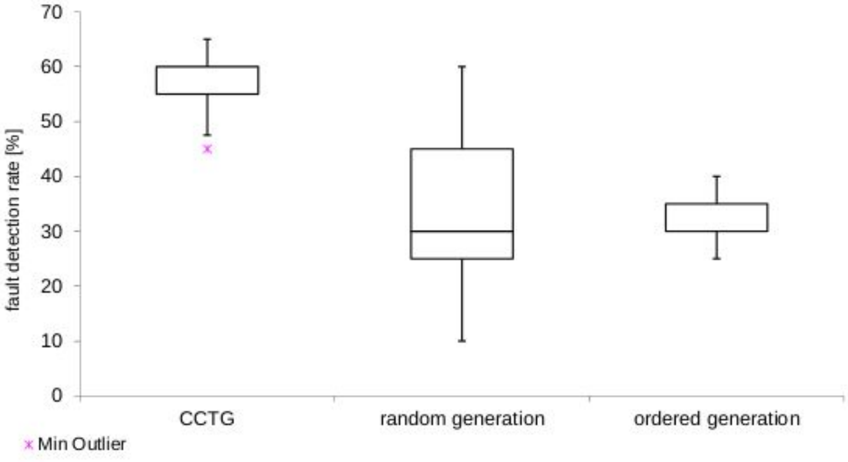

fault injection
testing
original SUT
SUT
Mutation Testing Tool
running test cases
comparing output
SUTs with seeded fault

Case Study

- results point out viability of concept
- detection rate of the CCTG method proved significantly greater



(a) Flex % of faults found



(b) Grep % of faults found

Conclusion

STILL
Software Testing IntelLigent Lab

- the first experiments indicate the concept to be viable
- applicable to automated generation of tests
- current version analyzes C code but can be easily extended to other languages

Future work:

- involvement of existing tests into test generation process to prevent duplication of already existing test cases
- more experiments, additional test generation strategies

Contact

Kryštof Sýkora, presenter

sykorkry@fel.cvut.cz

Miroslav Bureš

miroslav.bures@fel.cvut.cz

Bestoun S. Ahmed

bestoun@kau.se

kau.se

cvut.cz

still.felk.cvut.cz

@intelligent_lab