

Integrating Data and Control Dependencies for MC/DC Evolutionary Testing

Zeina Awedikian, Kamel Ayari, Giuliano Antoniol
SOCCER Lab (Software Cost-effective Change and Evolution
Research)

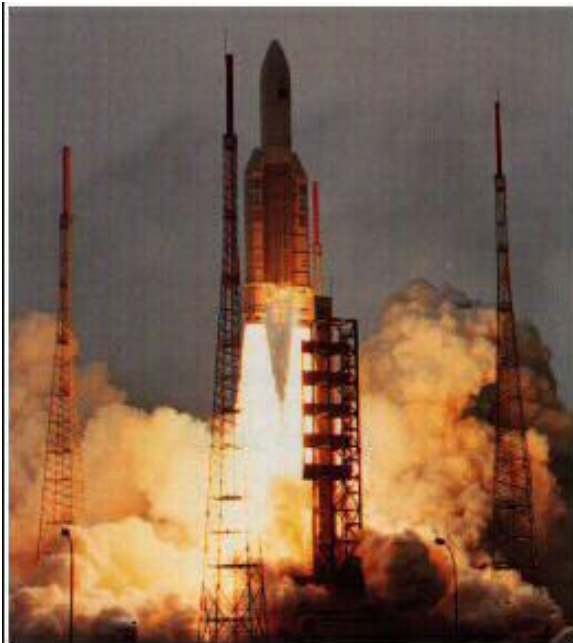
Ecole Polytechnique de Montreal

© 2008-2009



Ariane 5 – Root Cause

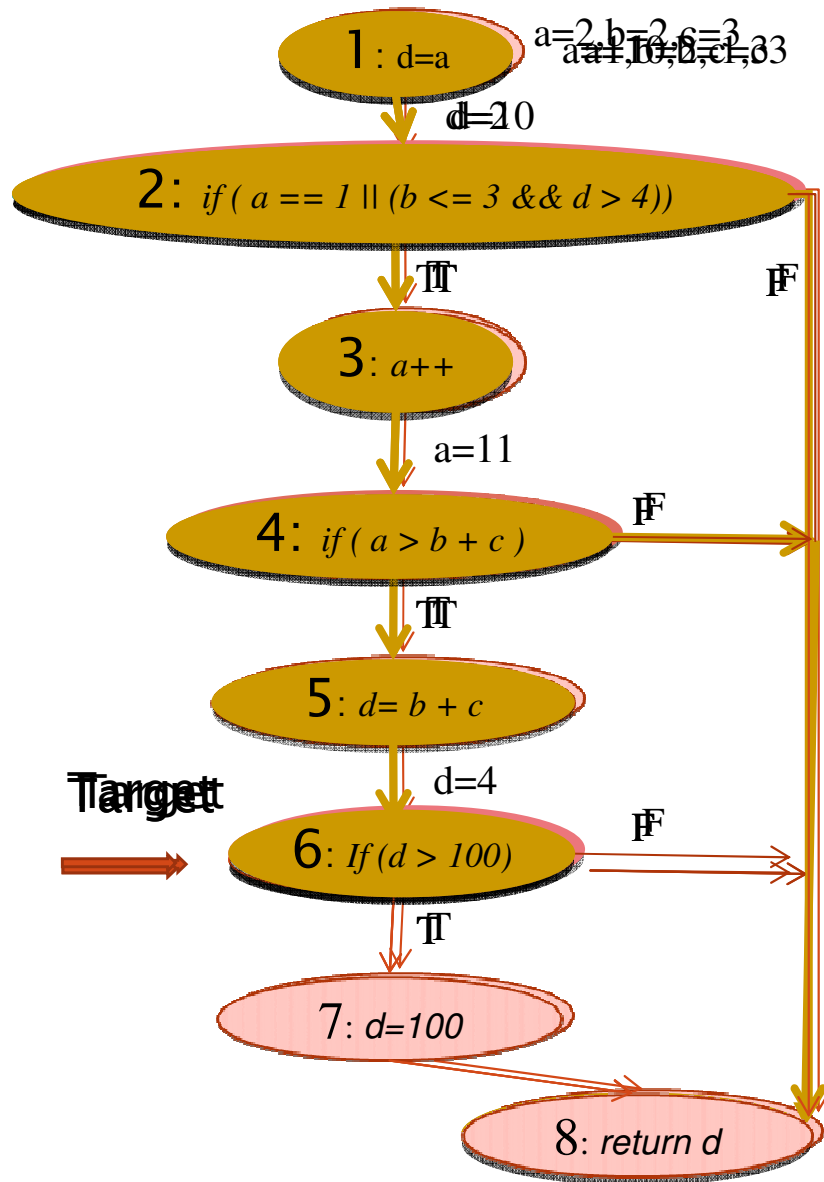
- Ariane 5 is a European expendable launch system.
- The rocket Ariane 5 Flight 501 (1996) is an example of software failure.
- Source of the failure reported by Inquiry Board:
A program segment for converting a floating point number to a signed 16 bit integer was executed with an input data value outside the range representable by a signed 16 bit integer.



Importance of testing

- Airborne computer software must meet the requirements of the Federal Aviation Administration (FAA).
- Software should be compliant with the RTCA/DO-178B document that treats system safety assessment.
- One of the objectives of DO-178B software is to generate tests that achieve the modified condition / decision coverage (MC/DC).

Decision coverage and MC/DC coverage



- **Decision coverage:**
Ensure we are testing all decisions at least once.
Ensure we are reaching these decisions at least once.
- **MC/DC coverage:**
Once a decision is reached, ensure all MC/DC test cases are covered.

Our Approach

Approach steps in order to automatically generate data to satisfy MC/DC :

Step 1: For each predicate compute sets for MC/DC coverage

Step 2: Instrument the code under test to compute fitness

- 2.1) Control dependencies
- 2.2) Data dependencies
- 2.3) Branching fitness function

Step 3: Generate test data using metaheuristic algorithms

Structural coverage criterium MC/DC

- If (A or B) then ...

A	B	Z
F	F	F
F	T	T
T	F	T
T	T	T

Tests cases:

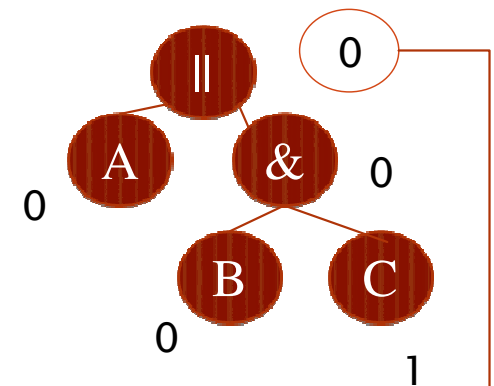
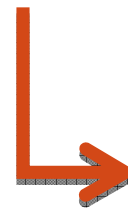
- A: (F,F → F)
(T,F → T)
- B: (F,T → T)
(F,F → F)
- A minimum of n+1 test cases for a decision with n inputs.

Generating MC/DC test cases

- We parse the decision structure into a tree
- Create a truth table TT
- Populate it by Evaluate graph nodes for each row in the truth table
- For each variable, search for the pair of rows where this variable affects alone the output of the decision.
- Record the tests sets

if (A || (B && C))

V= A, B, C
N=3



	A	B	C	
0	0	0	0	
1	0	0	1	0
2	0	1	0	
3	0	1	1	
4	1	0	0	
5	1	0	1	
6	1	1	0	
7	1	1	1	

The test sets:

A:

(0,0,0) -> 0 and (1,0,0) -> 1

(0,0,1) ->0 and (1,0,1) -> 1

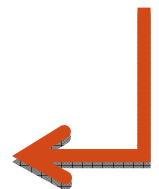
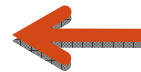
(0,1,0) -> 0 and (1,1,0) -> 1

B:

(0,0,1) 1-> 0 and (0,1,1) -> 1

C:

(0,1,0) -> 0 and (0,1,1) -> 1



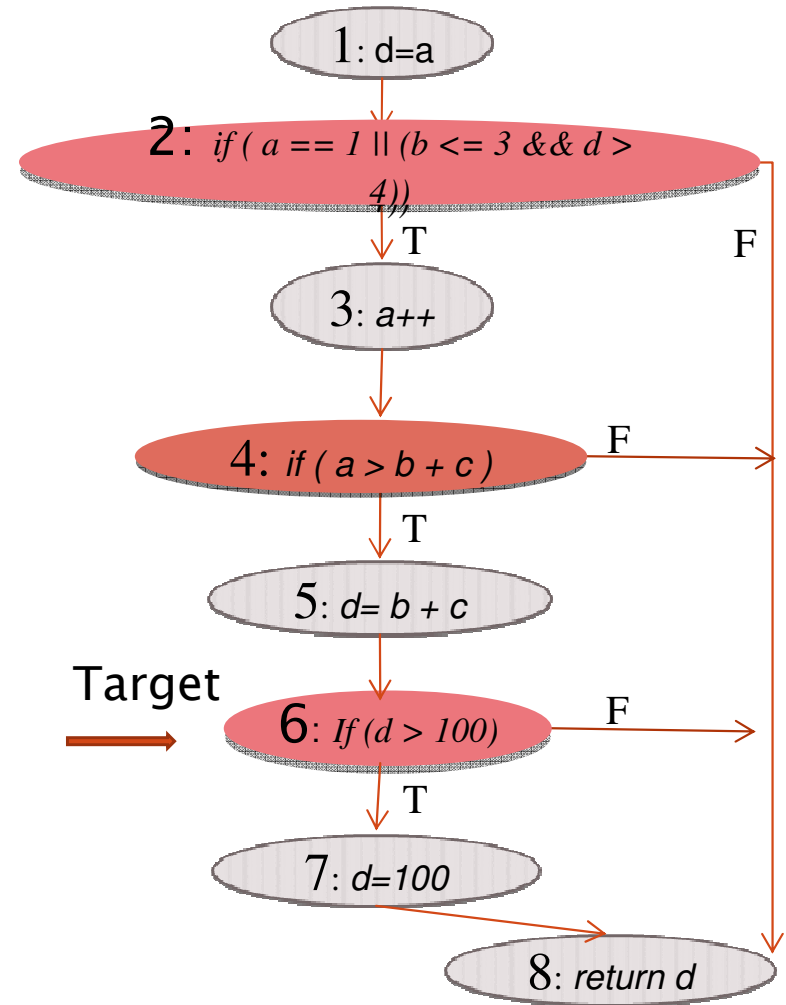
Control dependencies

To calculate control dependency fitness:

- Step 1: Extract the dependencies for each node.

Node	Depends on
3	2
4	2
5	4, 2
6	4, 2
7	6, 4, 2

- Step 2: Trace the execution of the code to check which nodes are traversed for each input test data.
- For each test input data, at run time:
 - 2, 2, 3 -> no decisions traversed -> control dep fit = 2-0 = 2
 - 1, 2, 3 -> { 2 } traversed -> Dep. fitness = 2-1 = 1



Data dependencies

- Control dependency fitness of target is 0 !
- We have no guidance how to generate input to make “shutdown” = true.
- The search becomes a total random search.
- Data dependency algorithm: look for all possible problem variables affecting the target and trace back recursively to collect all data and control dependency of all variables affecting it.

Pn = problem node ; pv = problem variable

last_def = last definition ; CD = control dependency

UV = used variables

Algo(pn, SET) {

Get pv at pn

For each last_def of pv

Save CD of last_def

NEW_SET = SET \cup CD

Save UV at last_def

For each CD and for each UV , new_pn

Algo(new_pn, NEW_SET)

next

Next

}

```
void Exit(b) {  
...  
1  if (b == Value)  
           r1 = true;  
2  else  
3           r1 = false;  
4  if (r1)  
5           error1 = true  
6  if (r2)  
7           error2 = true  
8  shutdown = error1 && error2  
9  if (shutdown) //target  
...  
}
```

Branching fitness function

- When a test input reaches the target, we need to know how close it is from achieving one of the tests.

```
int Exc(a,b,c) {
  if ( a == 1 || (b <= 3 && c > 4))
    a++;
}
```

Test data:

a= 3, b=2, c=-5



false

a= 6, b=7, c=1



false

- We extended the work of Leonardo Bottacci published in 2001 to generate the branching fitness function for the *If* and the *Else* branch of a decision.

Expression	Fitness if exp false	Fitness if exp. true	Fitness for IF then branch	Fitness for ELSE branch
a=b	F=abs(a-b)	F=0	(abs(a-b))	(a!=b ? k : 0)
a≠b	F=K	F=0	(a!=b ? 0 : k)	(a==b ? abs(a-b)) : 0
a<b	F=(a-b)+K	F=0	(a<b ? 0 : a-b + k)	(a<b ? a-b + k : 0)
a≤b	F=(a-b)	F=0	(a<=b ? 0 : a-b)	(a<=b ? a-b : 0)
a>b	F=(b-a)+K	F=0	(a > b ? 0 : a - b + k)	(a > b ? a - b + k : 0)
a≥b	F=(b-a)	F=0	(a >= b ? 0 : a -b)	(a >= b ? a -b : 0)
a b	F=min(f(a),f(b))	F=0	min(fa,fb)	fa + fb
a && b	F=f(a)+f(b)	F=0	fa + fb	min(fa,fb)

Generate test data using metaheuristic algorithms

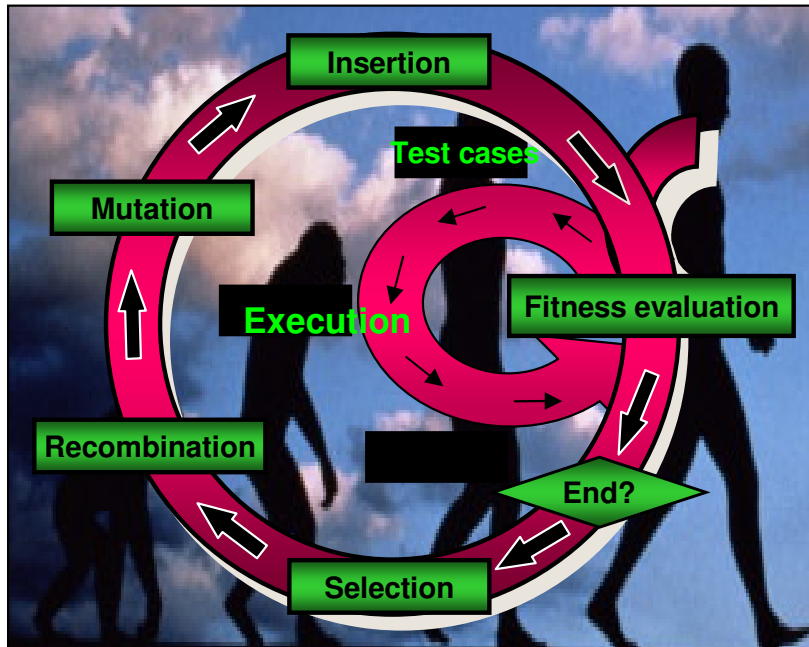
- Automatic generation of test input data requires a search of inputs in the input parameter space (real parameters, integer parameters...).
- We have then a search based software engineering problem, guided by a fitness function.

- For each generated input data, the fitness function:

$\text{Fitness} = \text{Control Dep. Fit.} + \text{Data Dep. Fit.} + \text{Branching Fit.}$

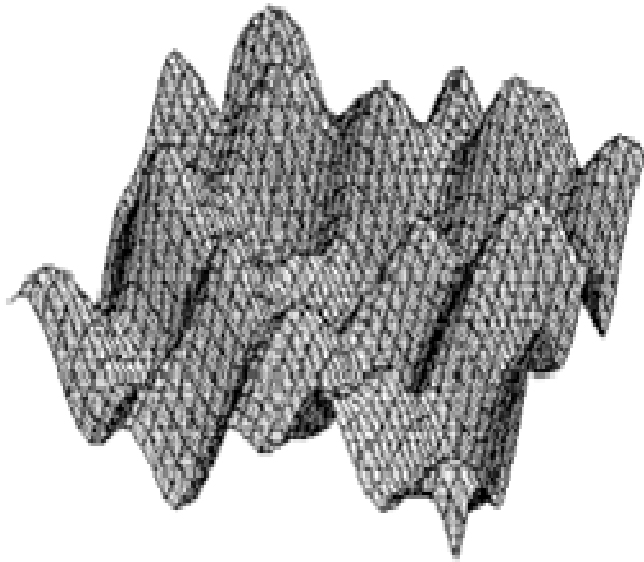
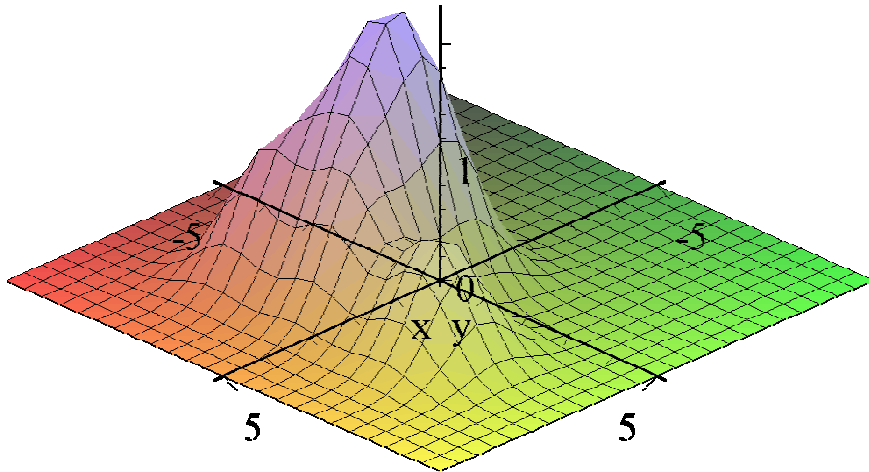
- We used:
 - Genetic algorithm
 - Hill climbing
 - Random generator

Genetic Algorithm



- One MC/DC set is selected
- Initially, we create randomly a population of n individuals.
- The algorithm evaluates the **fitness function** for each individual of the current generation.
- The algorithm selects m individuals with the best fitness function. They will be used as parents of the next generation.
- We apply crossover and mutation to generate the new population.
- After un maximum number of fitness evaluation, a second test case is selected.
- The algorithm loops until:
 - All tests cases are achieved.
 - We reached the maximum number of allowed fitness evaluation per test case.

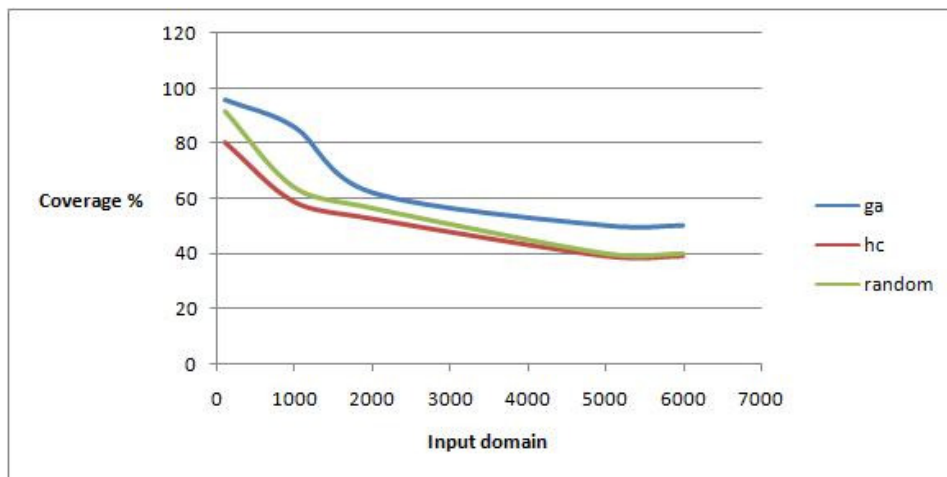
Hill Climbing



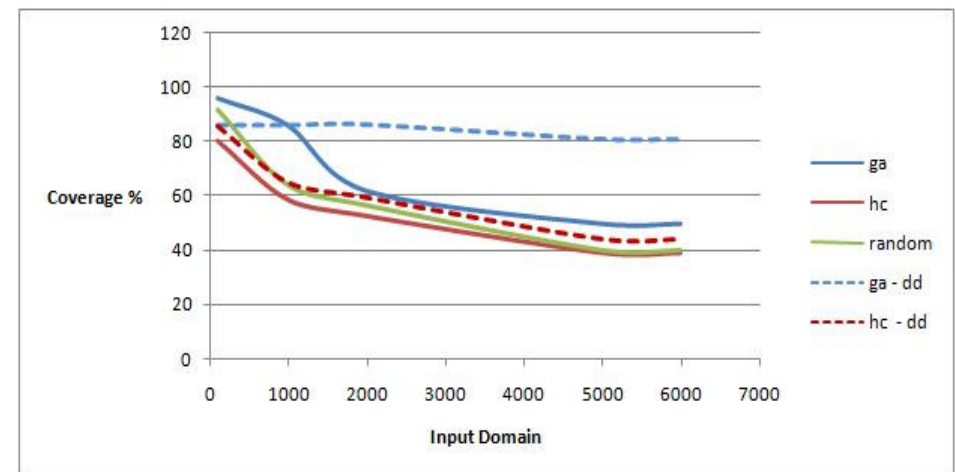
- One test set is selected
- Initially, a random point is selected.
- The fitness function is evaluated for this point.
- Then a neighbour is generated based on a step λ .
- The fitness function of the neighbour is evaluated, and if it improved, the neighbour becomes the new current solution.
- Random restart: After n neighbour generations, the algorithm selects randomly a new starting point and restart the neighbouring process for this new starting point.
- The algorithm loops for each of the test cases until:
 - All the tests cases are achieved.
 - The maximum number of allowed fitness evaluation per test case is reached.

Results (1)

No data dependency added –
Results for the *triangle* code



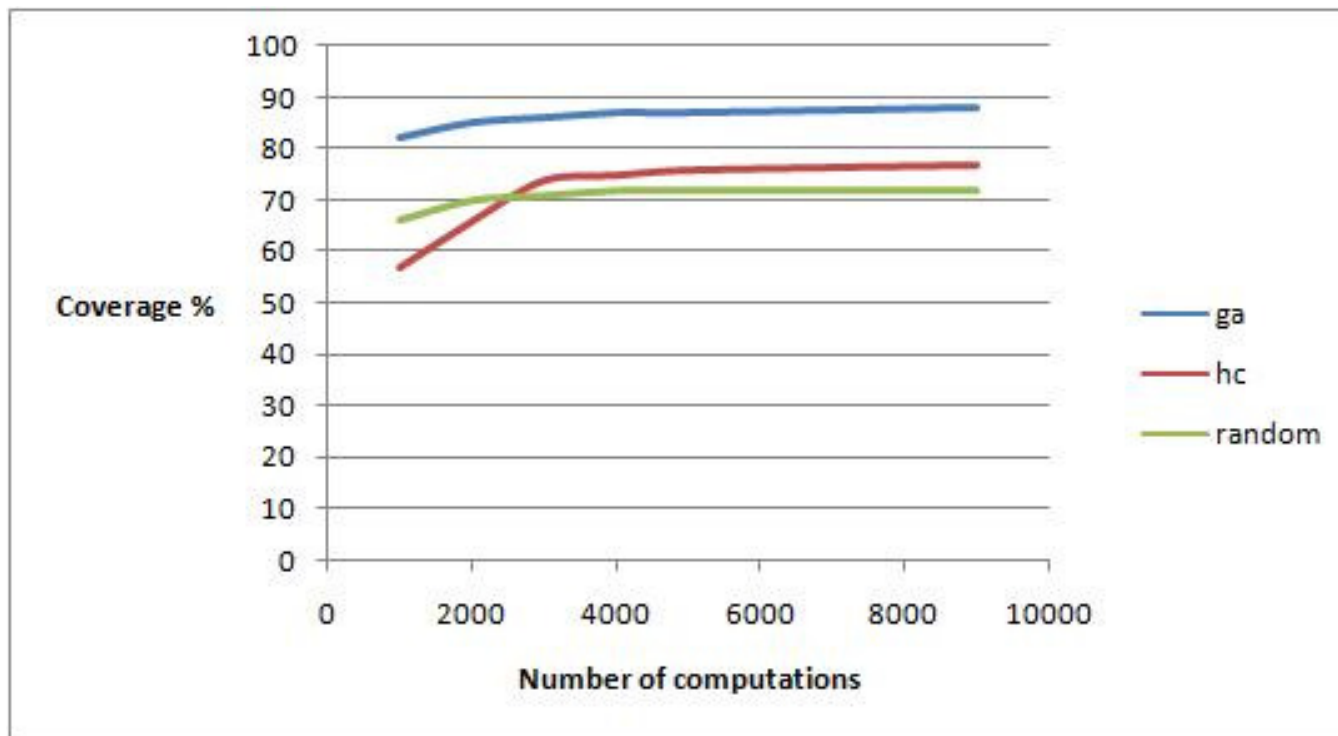
With Data dependency integrated –
Results for the *triangle* code



- We restricted the iterations to a maximum of 5000 fitness evaluations per test case.
- We run the tests 30 times for each decision in the code, for each algorithm.
- 30 run * 10 decisions for triangle * 3 average test cases per decision * 5000 fitness evaluation per test case = **4,500,000 fitness evaluation** per algorithm
- Integration the data dependencies sets, GA results improved dramatically.
- HC results didn't improve a lot. The reason is that first, HC requires much more generations, and second the neighbouring selection is not optimum.

Results (2)

- Results for the *NextDate* code.
- No data dependencies exist in the code.



Conclusion

- This is the first work to use metaheuristic for the MC/DC coverage.
- We were able to automatically generate test data for the MC/DC test criterion. In most of the companies such test data is written manually by expert testing engineers.
- The work can be generalised to cover other white box testing criteria.
- We extended McMinn (2006) Extending Chaining approach that generates data dependencies sets. The new sets generated include data and control dependencies to provide an improved search guidance.
- The experiments run show that integrating data dependencies improve the coverage percentage of the test cases.
- Future work: We will be fine tuning the parameters of the metaheuristic algorithm, and working on the neighbourhood selection of the Hill Climbing, to try to have even better results.

Thank you

Zeina Awedikian, Kamel Ayari, Giuliano Antoniol

Soccer Lab, Ecole Polytechnique de Montreal

zeina.awedikian@polymtl.ca