SENG3320/6320: Software Verification and Validation

School of Electrical Engineering and Computing

Semester 1, 2020

Mutation Testing

Mutation Testing

- Program mutation:
  - Create artificial bugs by injecting changes to statements of programs.
  - Simulate subtle bugs in real programs
- Mutation testing
  - A software testing technique based on program mutation.
  - Can be used to evaluate test effectiveness and enhance test suite.

An Example

Original Program

```
begin
    int x,y;
    input(x,y);
    if(x<y)
      output(x+y);
    else
      output(x*y);
end
```

Mutation

x≤y

Mutant

```
begin
    int x,y;
    input(x,y);
    if(x≤y)
      output(x+y);
    else
      output(x*y);
end
```

Summary of Mutation Testing

- A fault-based methodology for evaluating tests [Hamlet1977, DeMillo+1978]
  - **Step-1:** Applies artificial changes based on ***mutation operators*** to generate mutants (**each mutant with only one artificial bug**)
  - **Step-2:** Runs the test suite against each mutant
    - If any test fails (i.e., mutant's output != original output), the mutant is killed
    - If all tests pass, the mutant survives
  - **Step-3:** Computes the mutation score: the higher, the better

Does Mutation Testing work?

- Following example is100% for all control-flow and data-flow coverage
- How about mutation testing?

Test: sum(1,0)==1?

```
public int sum(int x, int y){
        return x-y; //should be x+y
    }
```

```
public int sum(int x, int y){
        return x*y;
    }
```

Mutation can be stronger than control/data-flow coverage

```
public int sum(int x, int y){
        return x-0;
    }
```

mutants

...

Kill a Mutant: example

```
read x
if (x>0)
  x=x+1;
if (x<4)
  x=1;
return x;
```

P

Value Replacement

```
read x
if (x>0)
  x=1+1;
if (x<4)
  x=1;
return x;
```

M

Can this test suite kill the mutant?

{t1: x=0; t2: x=1; t3: x=2}

Mutation Operators

- **Mutation operators** are language dependent
  - For Fortran a total of 22 operators were proposed
  - For Java around 40 operators were proposed
  - For C a total of 108 operators were proposed

Mutation Operator	Description
AAR	array reference for array reference replacement
ABS	absolute value insertion
ACR	array reference for constant replacement
AOR	arithmetic operator replacement
ASR	array reference for scalar variable replacement
CAR	constant for array reference replacement
CNR	comparable array name replacement
CRP	constant replacement
CSR	constant for scalar variable replacement
DER	DO statement alterations
DSA	DATA statement alterations
GLR	GOTO label replacement
LCR	logical connector replacement
ROR	relational operator replacement
RSR	RETURN statement replacement
SAN	statement analysis
SAR	scalar variable for array reference replacement
SCR	scalar for constant replacement
SDL	statement deletion
SRC	source constant replacement
SVR	scalar variable replacement
UOI	unary operator insertion

Mutation Operators

More Mutation Operators

Operator	Description
AOR	Arithmetic Operator Replacement
AOI	Arithmetic Operator Insertion
AOD	Arithmetic Operator Deletion
ROR	Relational Operator Replacement
COR	Conditional Operator Replacement
COI	Conditional Operator Insertion
COD	Conditional Operator Deletion
SOR	Shift Operator Replacement
LOR	Logical Operator Replacement
LOI	Logical Operator Insertion
LOD	Logical Operator Deletion
ASR	Assignment Operator Replacement

Language Feature	Operator	Description
Encapsulation	AMC	Access modifier change
Inheritance	IHD	Hiding variable deletion
	IHI	Hiding variable insertion
	IOD	Overriding method deletion
	IOP	Overriding method calling position change
	IOR	Overriding method rename
	ISI	super keyword insertion
	ISD	super keyword deletion
	IPC	Explicit call to a parent's constructor deletion
Polymorphism	PNC	new method call with child class type
	PMD	Member variable declaration with parent class type
	PPD	Parameter variable declaration with child class type
	PCI	Type cast operator insertion
	PCC	Cast type change
	PCD	Type cast operator deletion
	PRV	Reference assignment with other comparable variable
	OMR	Overloading method contents replace
	OMD	Overloading method deletion
	OAC	Arguments of overloading method call change
Java-Specific Features	JTI	this keyword insertion
	JTD	this keyword deletion
	JSI	static modifier insertion
	JSD	static modifier deletion
	JID	Member variable initialization deletion
	JDC	Java-supported default constructor deletion
	EOA	Reference assignment and content assignment replacement
	EOC	Reference comparison and content comparison replacement
	EAM	Accessor method change
	EMM	Modifier method change

ABS - Absolute Value Insertion

```
int gcd(int x, int y) {
  int tmp;
  while(y != 0) {
    tmp = x % y;
    x = y;
    y = tmp;
  }
  return x;
}
```

P

```
int gcd(int x, int y) {
  int tmp;
  while(y != 0) {
    tmp = x % y;
    x = abs(y);
    y = tmp;
  }
  return x;
}
```

M

ABS - Absolute Value Insertion

```
int gcd(int x, int y) {
  int tmp;
  while(y != 0) {
    tmp = x % y;
    x = y;
    y = tmp;
  }
  return x;
}
```

P

```
int gcd(int x, int y) {
  int tmp;
  while(y != 0) {
    tmp = x % y;
    x = y;
    y = tmp;
  }
  return abs(x);
}
```

M

AOR - Arithmetic Operator Replacement

```
int gcd(int x, int y) {
  int tmp;
  while(y != 0) {
    tmp = x % y;
    x = y;
    y = tmp;
  }
  return x;
}
```

P

+, -, *, /

AOR - Arithmetic Operator Replacement

```
int gcd(int x, int y) {
  int tmp;
  while(y != 0) {
    tmp = x * y;
    x = y;
    y = tmp;
  }
  return x;
}
```

P

+, -, %, /

ROR - Relational Operator Replacement

```
int gcd(int x, int y) {
  int tmp;
  while(y != 0) {
    tmp = x % y;
    x = y;
    y = tmp;
  }
  return x;
}
```

P

>, >=, <, <=, ==, false , true

COR - Conditional Operator Replacement

```
if(a && b)
```

```
if(a || b)
if(false)
if(true)
if(a)
if(b)
```

SOR - Shift Operator Replacement

x = m>>a

x = m<<a

x = m>>>a

x = m

LOR - Logical Operator Replacement

x = m&n

x = m|n

x = m^n

x = m

x = n

UOI - Unary Operator Insertion

```
int gcd(int x, int y) {
  int tmp;
  while(y != 0) {
    tmp = x % y;
    x = y;
    y = tmp;
  }
  return x;
}
```

P

```
tmp = x % (-y);
tmp = x % (+y);
tmp = x % (--y);
tmp = x % (++y);
...
```

mutants

SVR - Scalar Variable Replacement

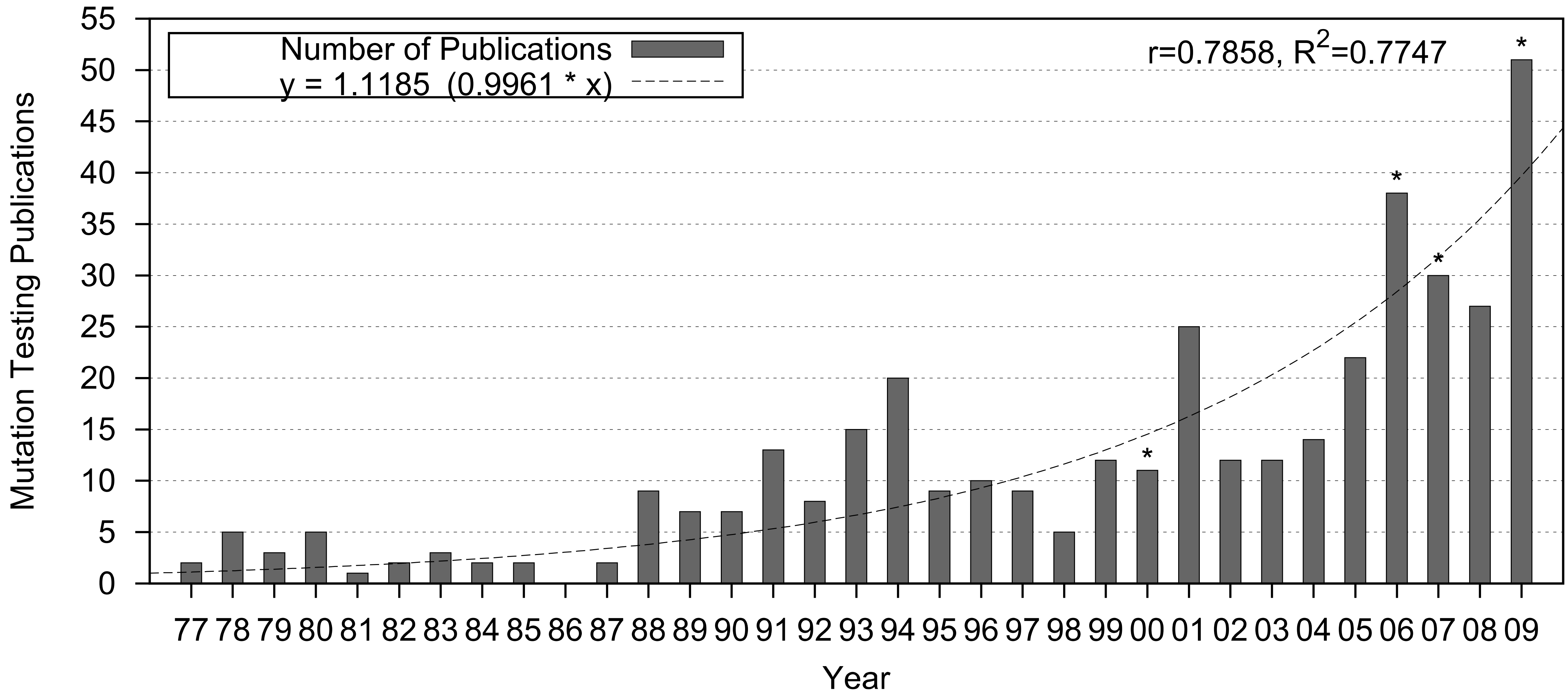
```
int gcd(int x, int y) {
  int tmp;
  while(y != 0) {
    tmp = x % y;
    x = y;
    y = tmp;
  }
  return x;
}
```

P

```
    tmp = x % x
    tmp = y % y
    x = x % y
    y = y % x
    tmp = tmp % y
tmp = x % tmp
    ...
```

mutants

Development of Mutation Testing



From: Yue Jia and Mark Harman, An analysis and survey of the development of mutation testing, TSE, 2010

Does Mutation Testing work?

“Mutation testing is more powerful than statement or branch coverage.”

Walsh, PhD thesis, State University of NY at Binghampton, 1985

“Mutation testing is superior to data-flow coverage criteria.”

Frankl, Weiss, Hu, Journal of Systems and Software, 1997

“Generated mutants are similar to real faults.”

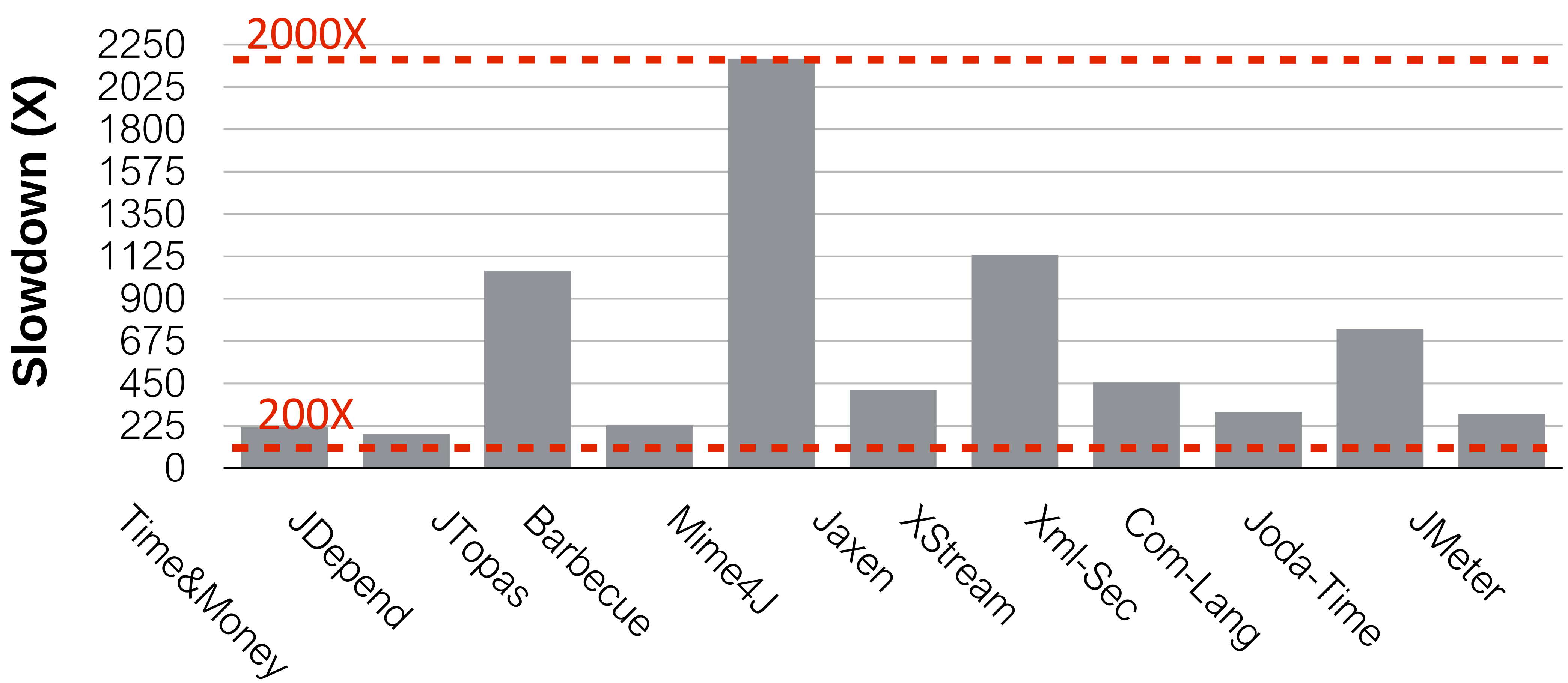
Andrews, Briand, Labiche, ICSE 2005

“Generated mutants can substitute real faults in software testing experimentation.”

Just, Jalali, Inozemtseva, Ernst, Holmes, and Fraser, FSE 2014

Limitation of Mutation Testing

- Mutation testing is extremely costly, since we need to run the test suite against each mutant
  - State-of-the-art PIT mutation tool (with many optimizations) on 11 OSS:



Mutation testing tools

- Java
  - **PIT: http://pitest.org/**
  - MAJOR: http://mutation-testing.org/
  - Javalanche: https://github.com/david-schuler/javalanche/
  - MuJava: http://cs.gmu.edu/~offutt/mujava/
- C
  - MILU: http://www0.cs.ucl.ac.uk/staff/y.jia/Milu/
- Python
  - MutPy: https://pypi.python.org/pypi/MutPy/0.4.0
- C#
  - NinjaTurtles: http://ninjaturtles.codeplex.com/

Further Reading

- R. G. Hamlet, “Testing programs with the aid of a compiler,” TSE, vol. 3, 1977.
- R. A. DeMillo, R. J. Lipton, and F. G. Sayward, “Hints on test data selection: Help for the practicing programmer,” Computer, vol. 11, 1978.
- A. J. Offutt, A. Lee, G. Rothermel, R. Untch, and C. Zapf, “An experimental determination of sufficient mutation operators,” ACM TOSEM, vol. 5, no. 2, pp. 99–118, 1996.
- James C. King, Symbolic execution and program testing, Communications of the ACM, volume 19, number 7, 1976, 385—394.
- Cadar, Cristian; Dunbar, Daniel; Engler, Dawson, "KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs". Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation. OSDI'08. Berkeley, CA, USA, USENIX Association, pp. 209–224.

Random/Fuzz Testing

Exhaustive Testing is Hard

```
int max(int x, int y)
{
  if (x > y)
    return x;
  else
    return x;
}
```

- Number of possible test cases (assuming 32 bit integers)
  - $2^{32} \times 2^{32} = 2^{64}$
- Do bigger test sets help?
  - Test set

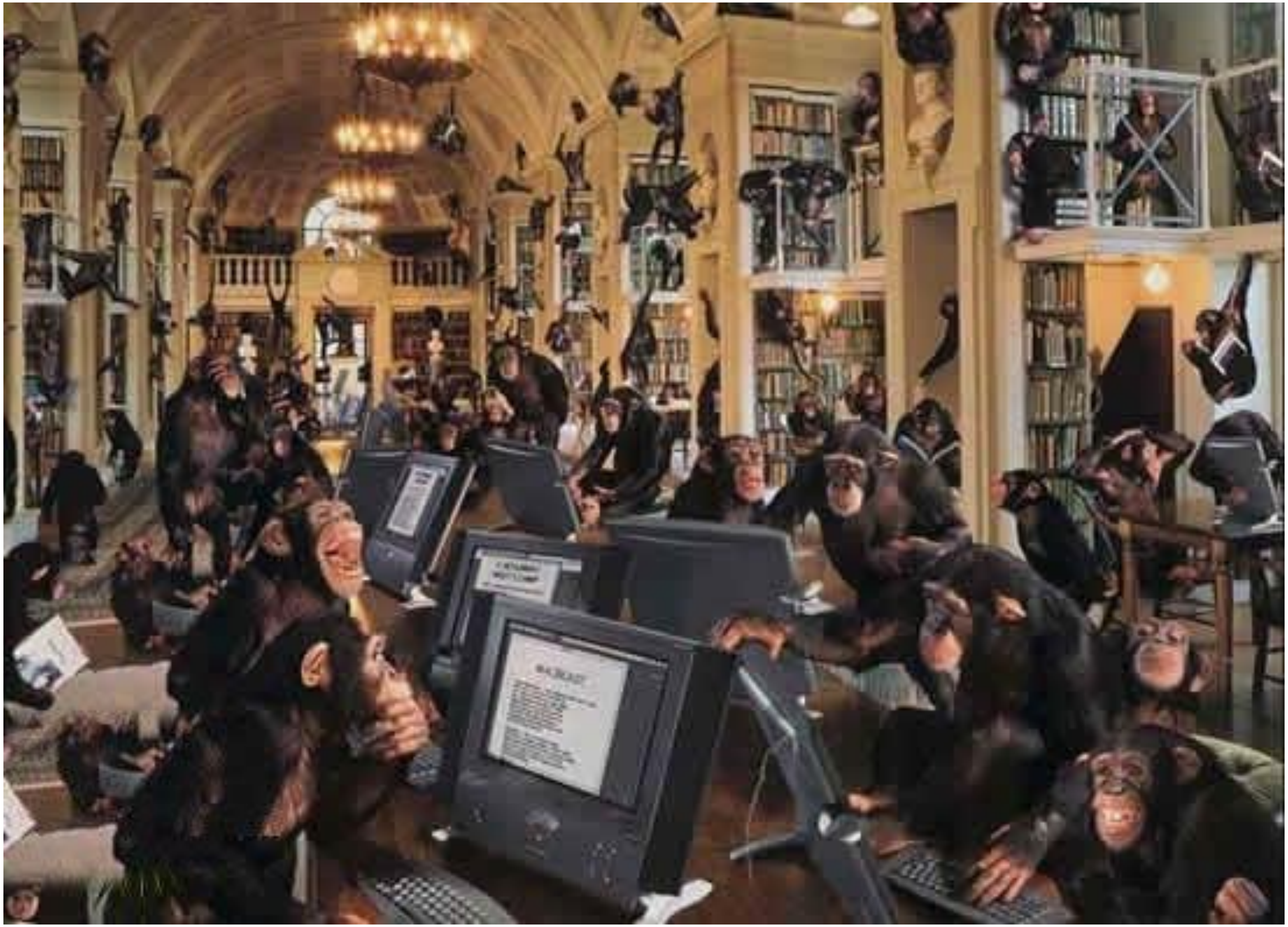
  {(x=3,y=2), (x=2,y=3)}

  will detect the error
  - Test set

  {(x=3,y=2),(x=4,y=3),(x=5,y=1)}

  will not detect the error although it has more test cases
- The power of the test set is not determined by the number of test cases

How about this approach?



Random testing

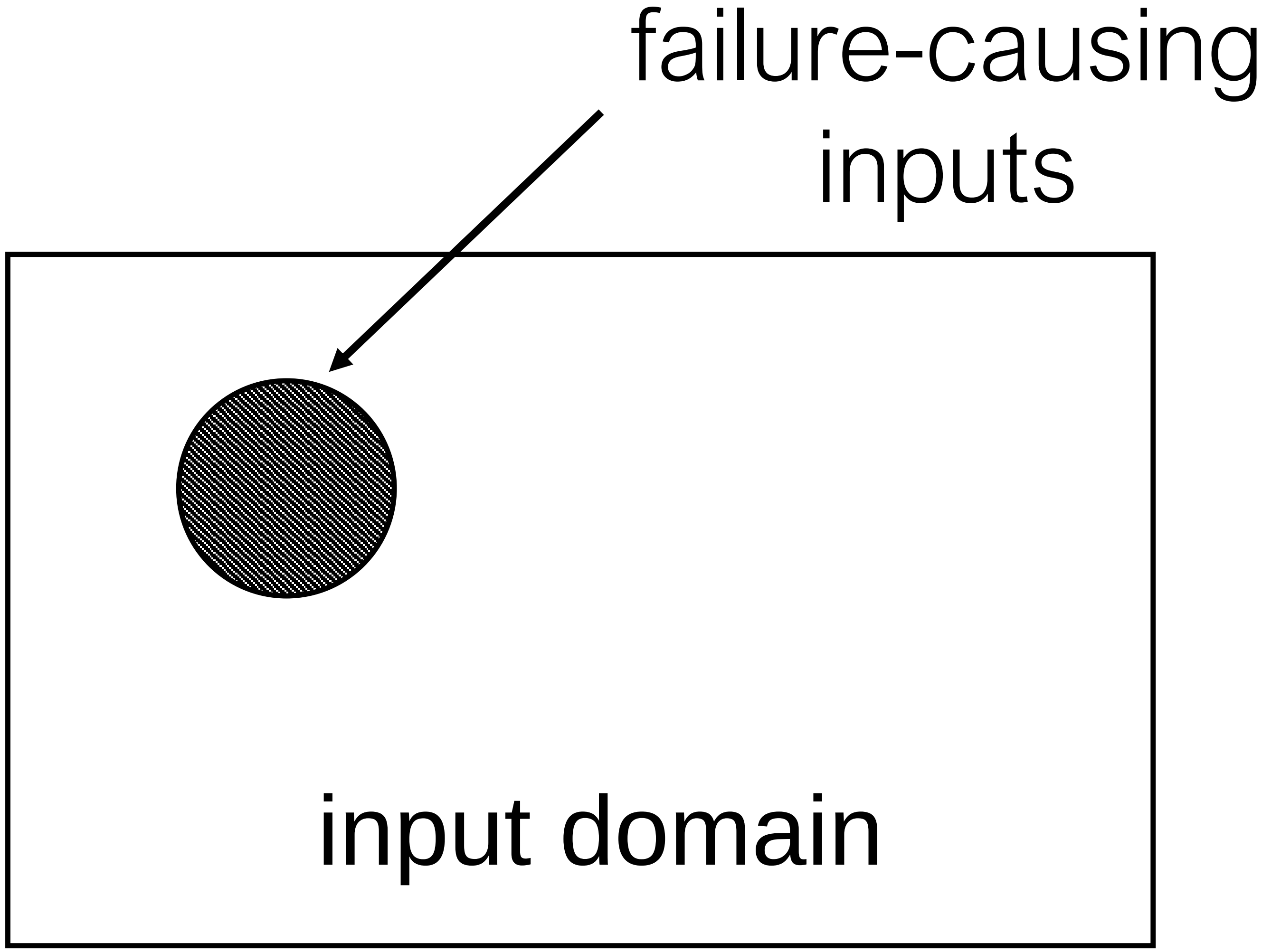
- Use a random number generator (e.g., monkeys) to generate test cases
- Also called Fuzz Testing, Monkey Testing

```
int myAbs(int x)
{
  if (x > 0)
    return x;
  else
    return x; //bug: should be '-x'
}
```

The random tests for this function could be: {123, 36, -35, 48, 0}

Random testing

- Selects tests from the entire input domain randomly and independently
  - Input domain – set of all possible inputs
  - Failure-causing inputs – inputs that exhibit failure



A sample test program

```
void testAbs(int n) {
    for (int i=0; i<n; i++) {
        int x = getRandomInput();
        int result = myAbs(x);
        assert(result>=0);
    }
}
```

Generating random inputs

- Suppose that a program accepts one parameter below:

  integer X

- Consider applying random testing to this range of X values:

  -10 ≤ X < 10

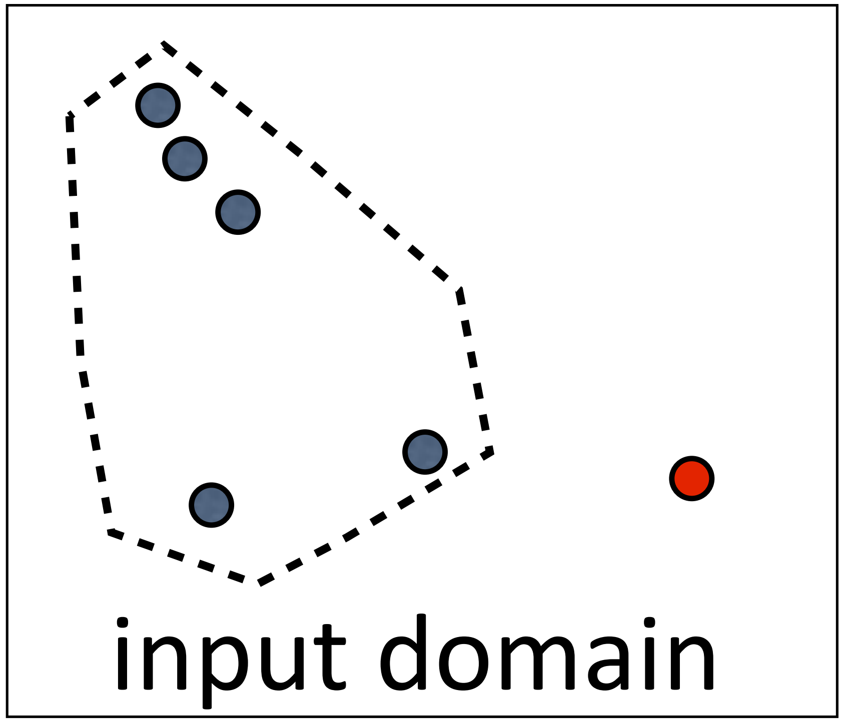
Can you write a program to implement RT with a uniform distribution?

Adaptive Random Testing

- Random testing - no guide towards failure-causing inputs (inputs that exhibit failure)

- Adaptive random testing: to achieve "even spread of test cases" in random testing.

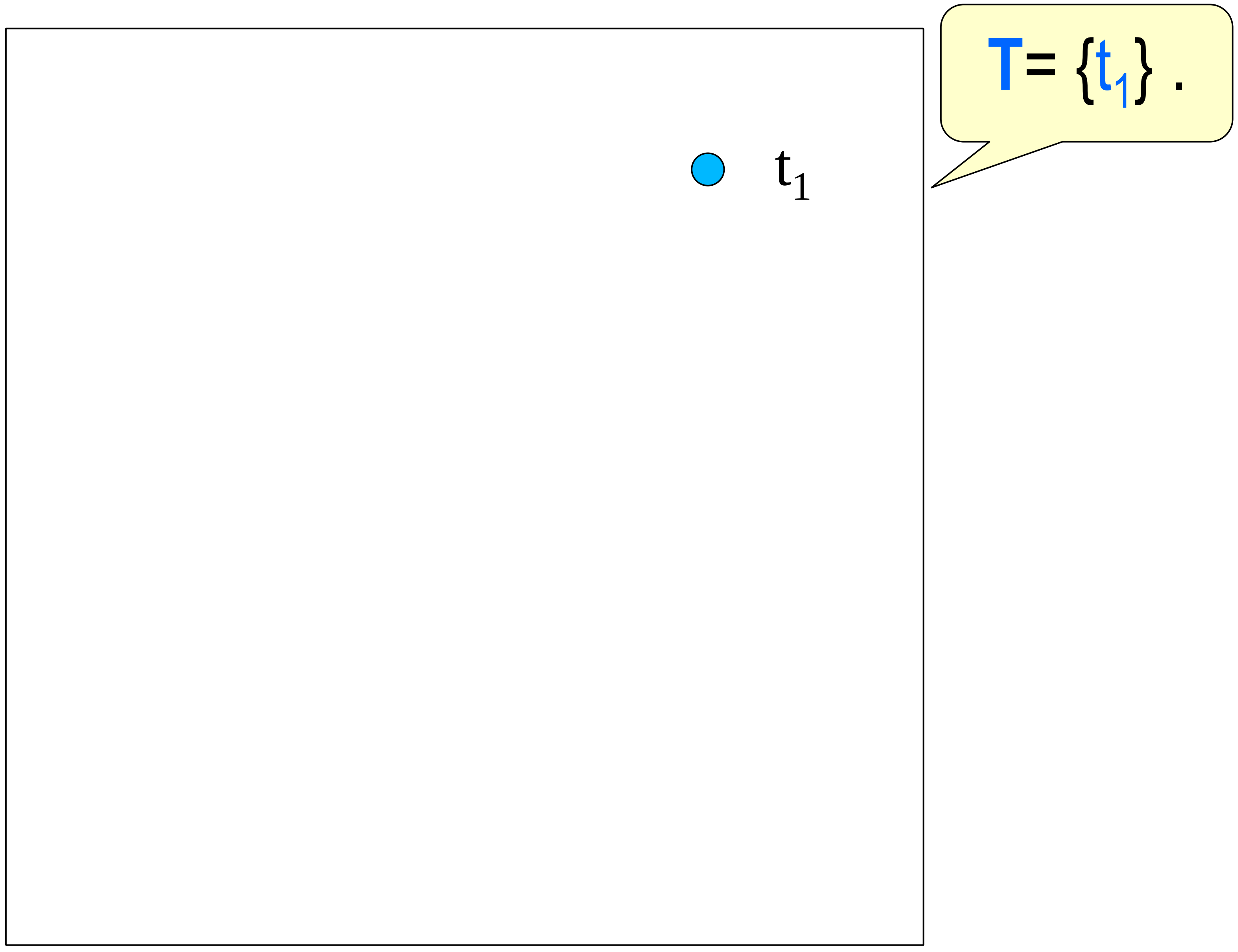
- One algorithm is **Fixed Size Candidate Set ART**



T. Y. Chen, H. Leung, and I. K. Mak. Adaptive random testing. In Proceedings of the 9th Asian Computing Science Conference, pages 320–329, 2004.

Fixed Size Candidate Set ART

Step 1. Randomly select the first input, namely t_1, and store it in a list (called **T**)

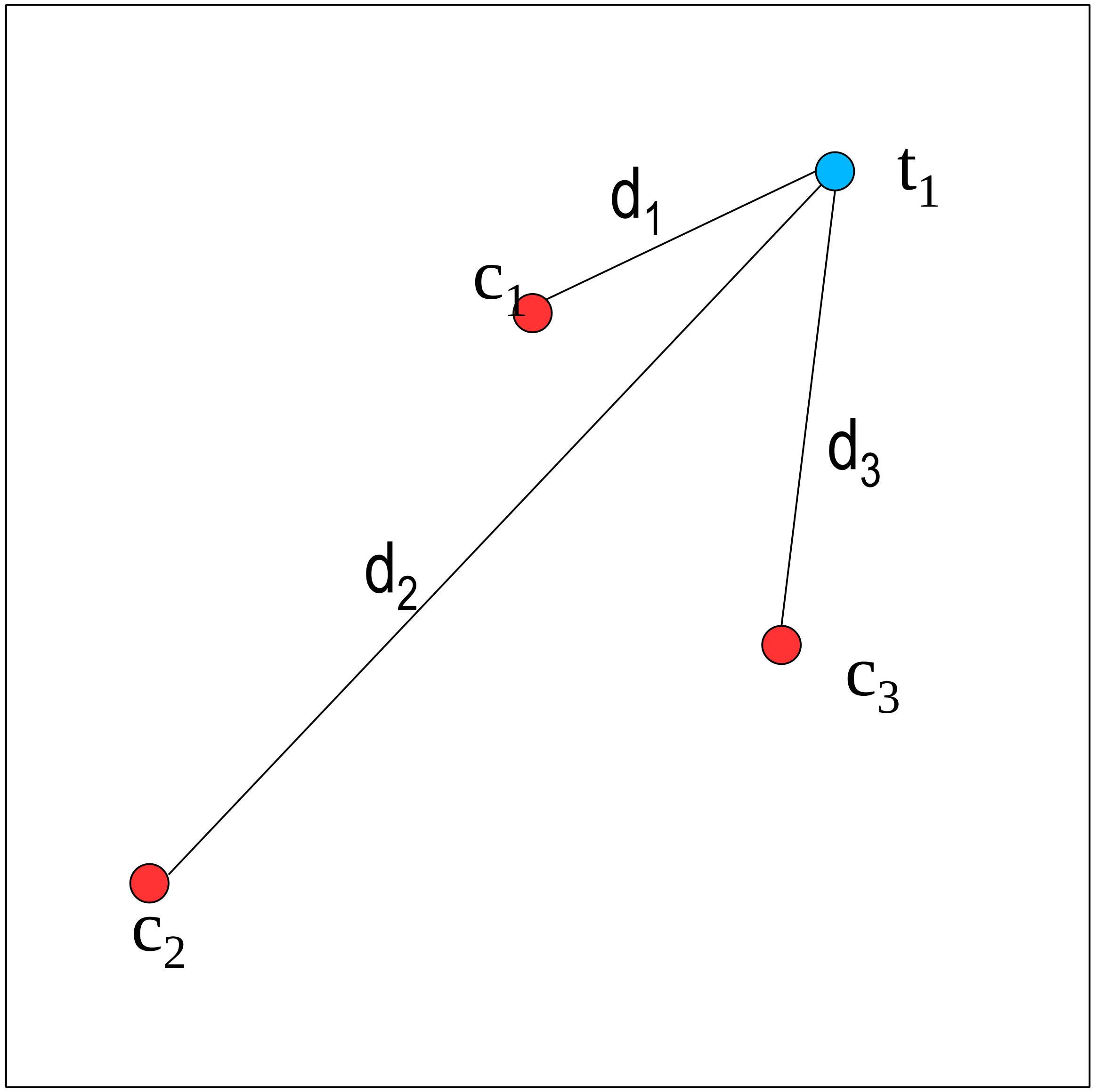


Fixed Size Candidate Set ART (cont.)

Step 2. Construct ***k*** random inputs to form a **candidate set** **C** = {c_1, c_2, …, c_k}, and measure their distance to t_1.

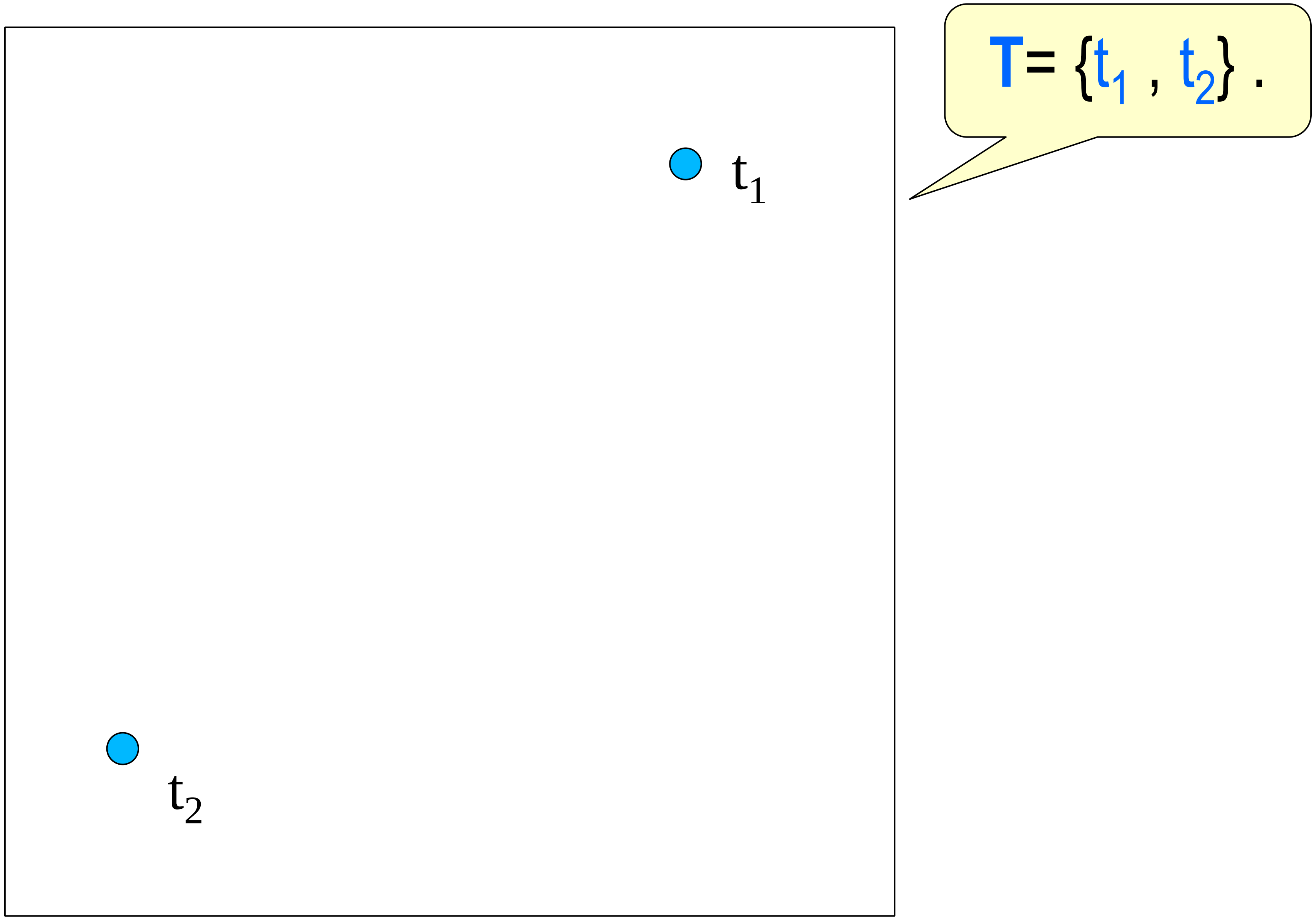
In this example, let set ***k*** = 3.

C= {c_1, c_2, c_3} .



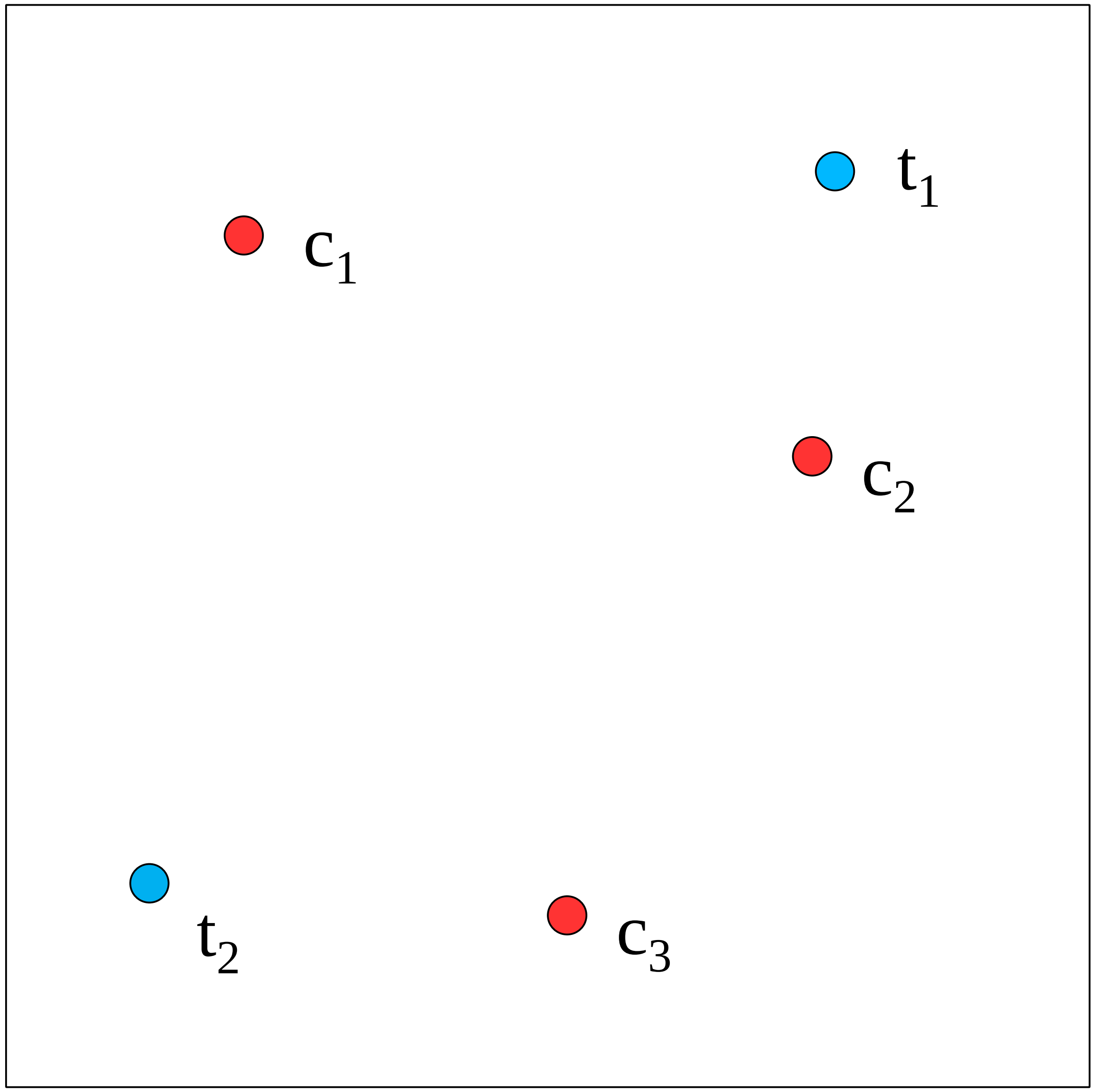
Fixed Size Candidate Set ART (cont.)

Step 3. Select the candidate which is the farthest away from t_1 to be the next test case. We name it t_2 and store it in **T**.



Fixed Size Candidate Set ART (cont.)

Step 4. Re-construct another **candidate set** **C** with ***k*** random inputs.

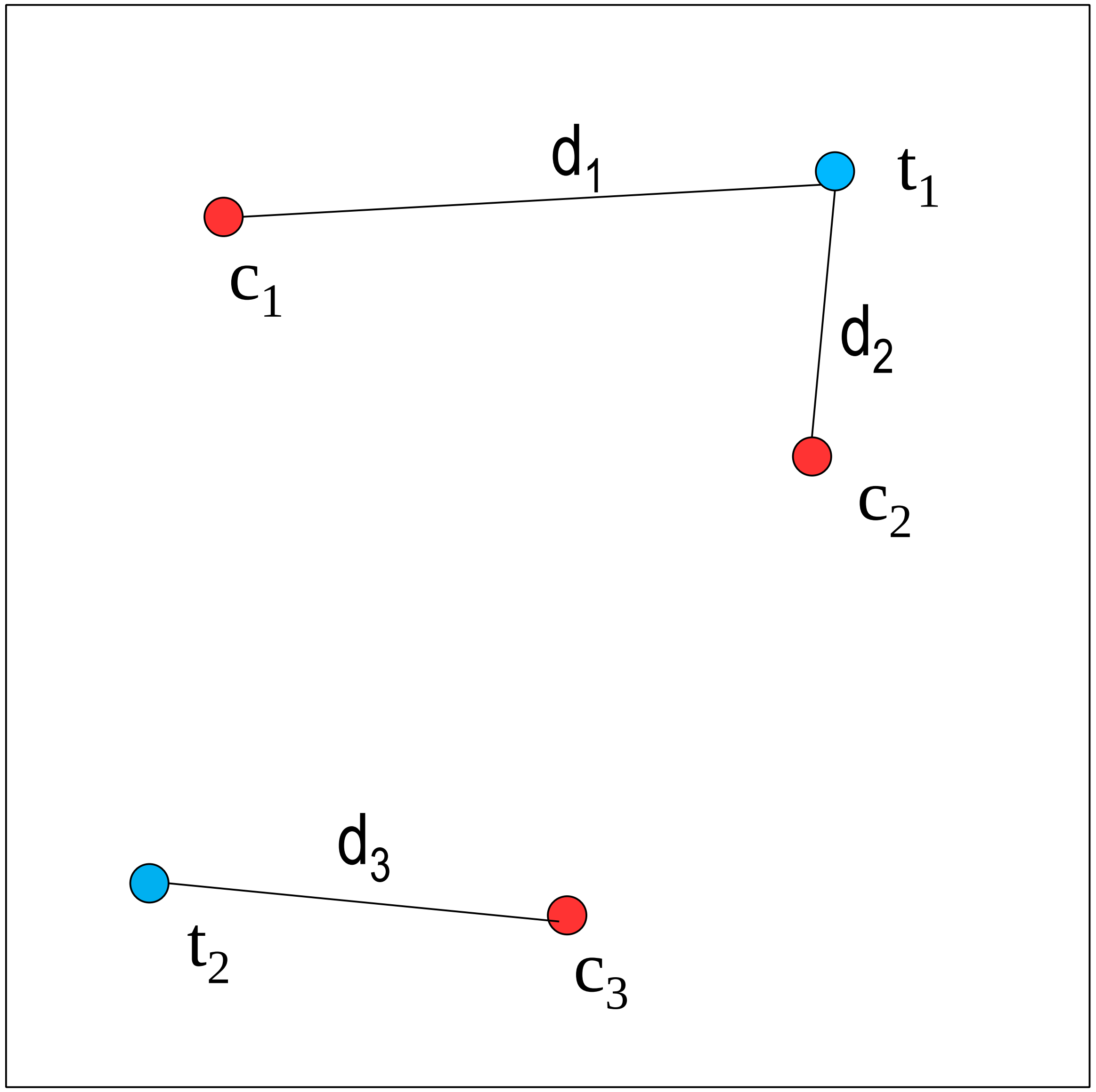


k = 3. Another **C**= {c_1, c_2, c_3} will be constructed

Fixed Size Candidate Set ART (cont.)

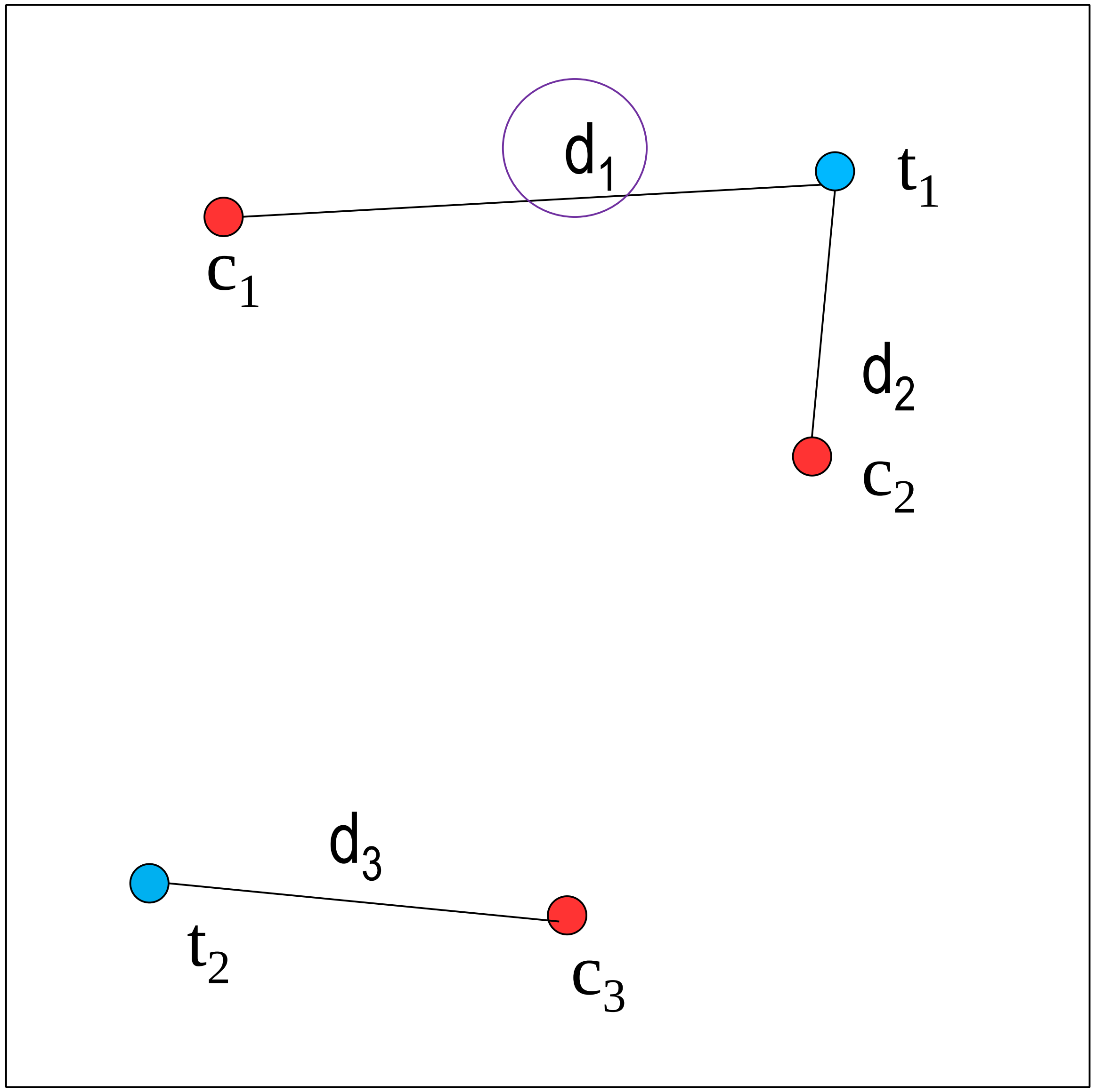
Step 5. For each candidate c_j in **C**, do the following

1. find which test case in **T** is the nearest neighbour of c_j
2. calculate the distance between c_j and its nearest neighbour.



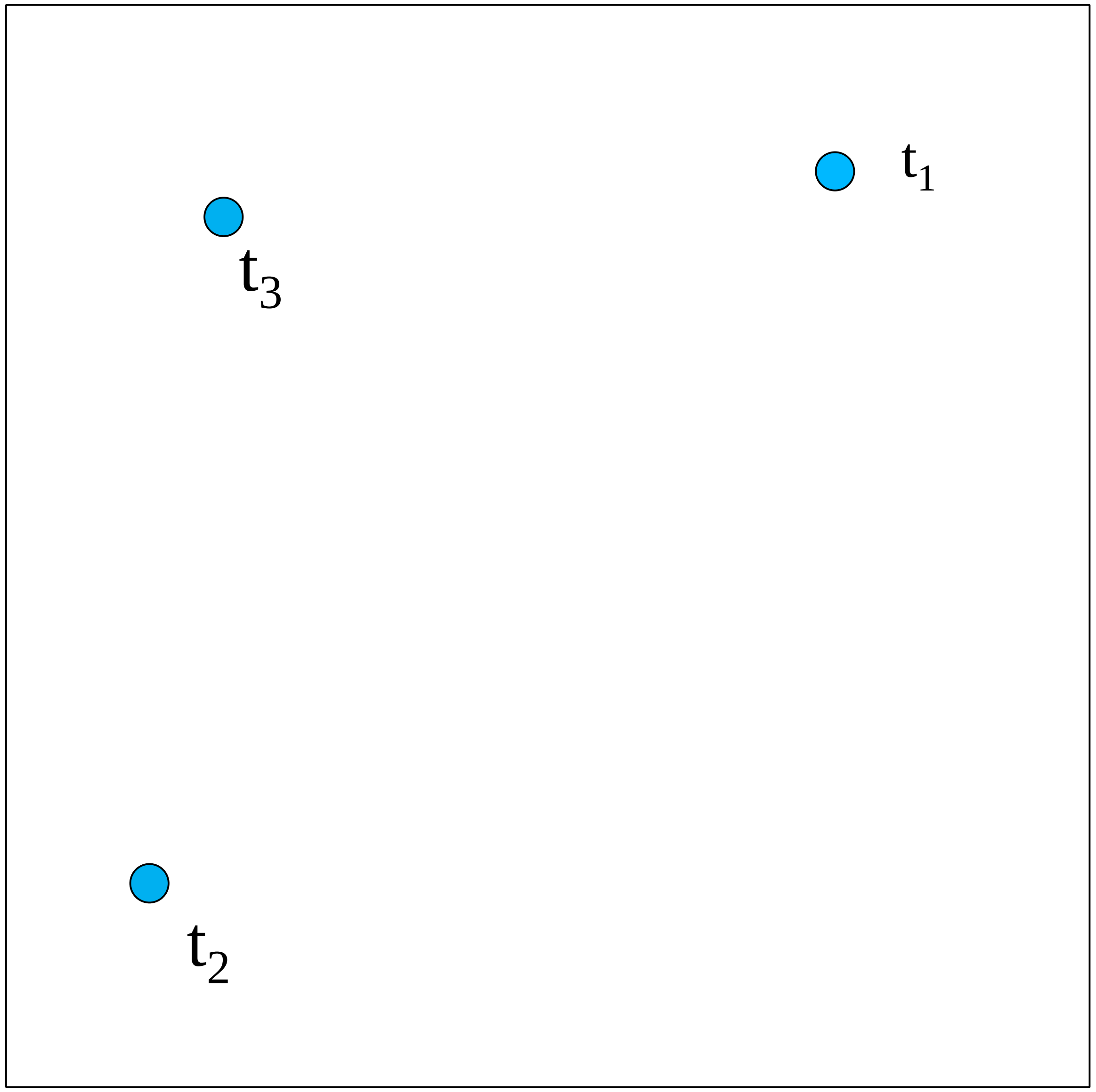
Fixed Size Candidate Set ART (cont.)

Step 6. Select the candidate with the longest distance to its nearest neighbour.



Fixed Size Candidate Set ART (cont.)

Step 7. Store the selected candidate in T



Fixed Size Candidate Set ART (cont.)

Repeat Steps 4-7 until the **stop** condition is satisfied.

The criterion for selecting the best candidate is:

$$\mathbf{Max(\ min(c_1, T), min(c_2, T), \ldots, min(c_k, T)\ )}$$

Distance Metric in ART

- Euclidean distance metric can be used to calculate the distance between a candidate and executed test case

If $\mathbf{p} = (p_1, p_2,..., p_n)$ and $\mathbf{q} = (q_1, q_2,..., q_n)$ are two points in n-dimensional space

$$\mathrm{d}(\mathbf{p}, \mathbf{q}) = \mathrm{d}(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

[Source: https://en.wikipedia.org/wiki/Euclidean_distance]

ART Algorithm /1

Algorithm 1:

```
/*
selected_set := { test data already selected };
candidate_set := {};
total_number_of_candidates := 10;
*/
function Select_The_Best_Test_Data(selected_set, candidate_set,
                    total_number_of_candidates);
    best_distance := −1.0;
    for i := 1 to total_number_of_candidates do
        candidate := randomly generate one test data from the program
                     input domain, the test data cannot be in
                     candidate_set nor in selected_set;
        candidate_set := candidate_set + { candidate };
        min_candidate_distance := Max_Integer;
        foreach j in selected_set do
            min_candidate_distance := Minimum(min_candidate_distance,
                     Euclidean_Distance(j, candidate));
        end_foreach
        if (best_distance < min_candidate_distance) then
            best_data := candidate;
            best_distance := min_candidate_distance;
        end_if
    end_for
    return best_data;
end_function
```


T. Y. Chen, H. Leung, and I. K. Mak. Adaptive random testing. In Proceedings of the 9th Asian Computing Science Conference, pages 320–329, 2004

Algorithm 2:

```
initial_test_data := randomly generate a test data from the input domain;
selected_set := { initial_test_data };
counter := 1;
total_number_of_candidates := 10;
use initial_test_data to test the program;
if (program output is incorrect) then
        reveal_failure := true;
else
        reveal_failure := false;
end_if
while (not reveal_failure) do
        candidate_set := {};
        test_data := Select_The_Best_Test_Data(selected_set, candidate_set,
                                   total_number_of_candidates);
        use test_data to test the program;
        if (program output is incorrect) then
                reveal_failure := true;
        else
                selected_set := selected_set + { test_data };
                counter := counter + 1;
        end_if
end_while
output counter;
```

ART Algorithm /2

T. Y. Chen, H. Leung, and I. K. Mak. Adaptive random testing. In Proceedings of the 9th Asian Computing Science Conference, pages 320–329, 2004

How to do random white-box testing?

Generate random method invocations

Program under test

```
public class HashSet extends Set{
	public boolean add(Object o)
	{…}
	public boolean remove(Object
o){…}
	public boolean isEmpty()
	{…}
	public boolean equals(Object
o){…}
	…
  }
```

Generate random parameters

test1:

```
Set s = new HashSet();
s.add("hi");
```

test2:

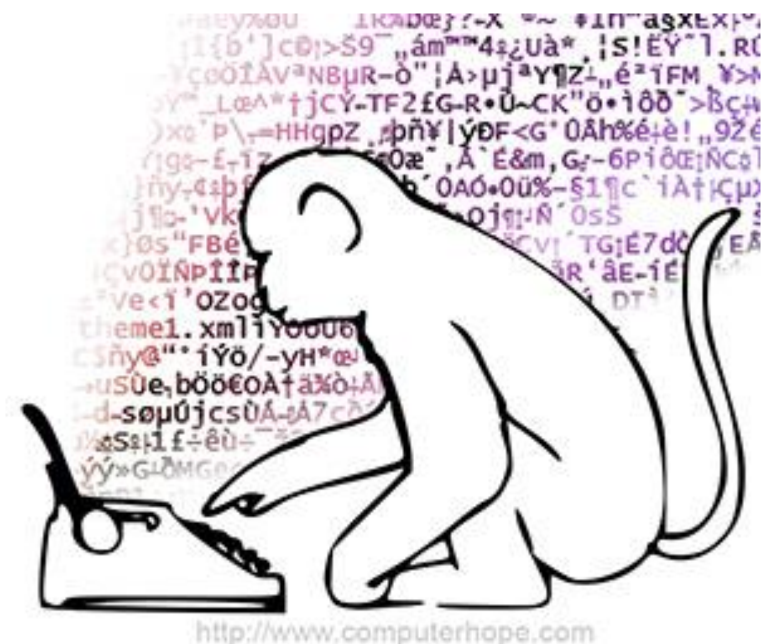
```
Set s = new HashSet();
s.add("hi");
s.remove(null);
```

test3:

```
Set s = new HashSet();
s.isEmpty();
s.remove("no");
s.isEmpty();
s.add("no");
s.isEmpty();
s.isEmpty();
…
```

Fuzz Testing

- Fuzz Testing (also called Fuzzing Testing):
  - A random testing technique that involves providing invalid, unexpected, or random data as inputs to a program.
  - Commonly used to discover coding errors and unknown vulnerabilities in software, operating systems or networks by inputting massive amounts of random data, called fuzz, to the system in an attempt to make it crash.
  - A cost-effective alternative to more systematic testing techniques.



Fuzz Testing Example /1

- Standard HTTP GET request
  - GET /index.html HTTP/1.1

Anomalous requests:

GET ///////index.html HTTP/1.1
GET %n%n%n%n%n%n.html HTTP/1.1
GET /AAAAAAAAAAAAA.html HTTP/1.1
GET /index.html HTTTTTTTTTTTTTP/1.1
GET /index.html HTTP/1.1.1.1.1.1.1.1

Different Ways To Generate Inputs

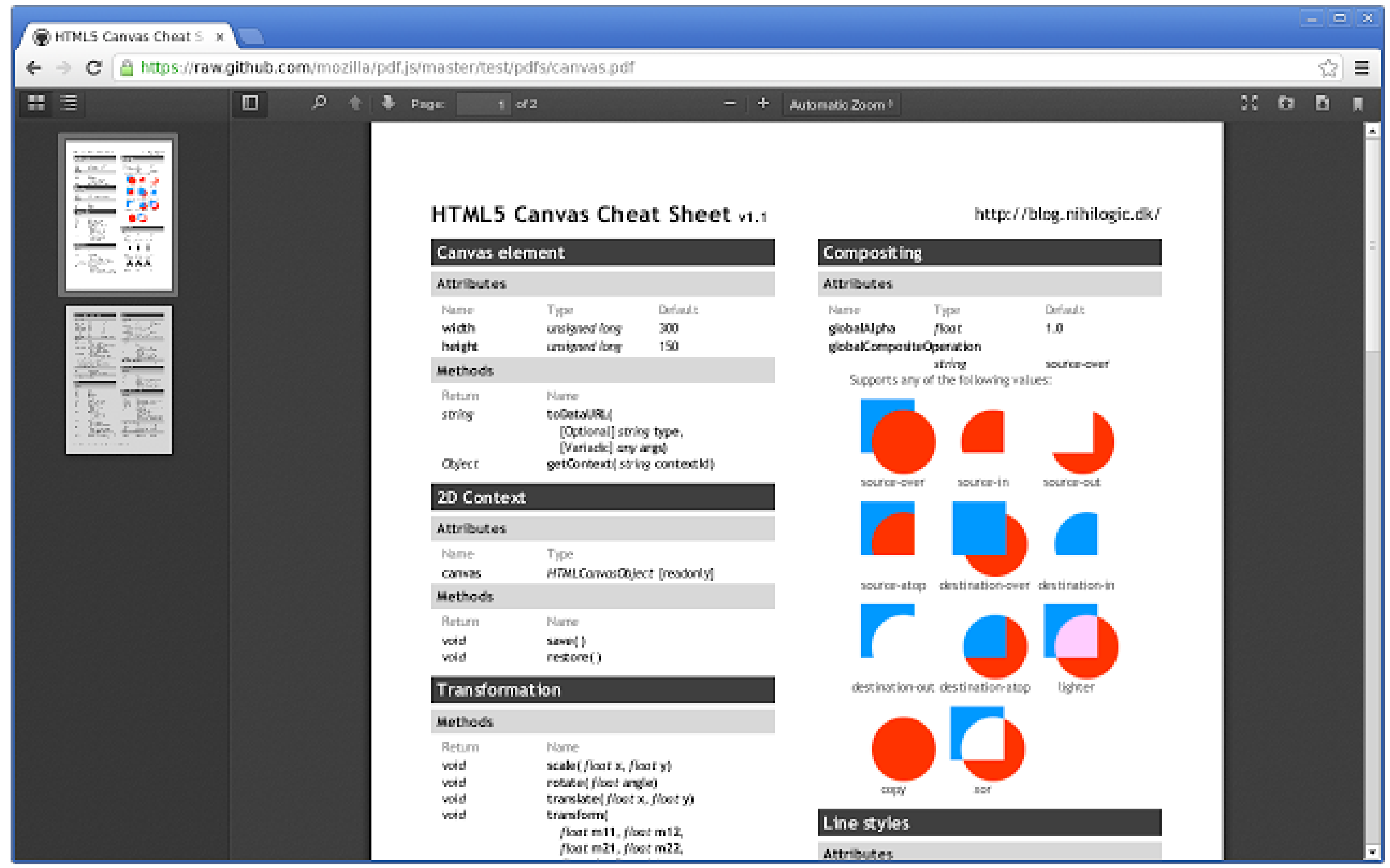
- Mutation Based - “Dumb Fuzzing”
- Generation Based - “Smart Fuzzing”

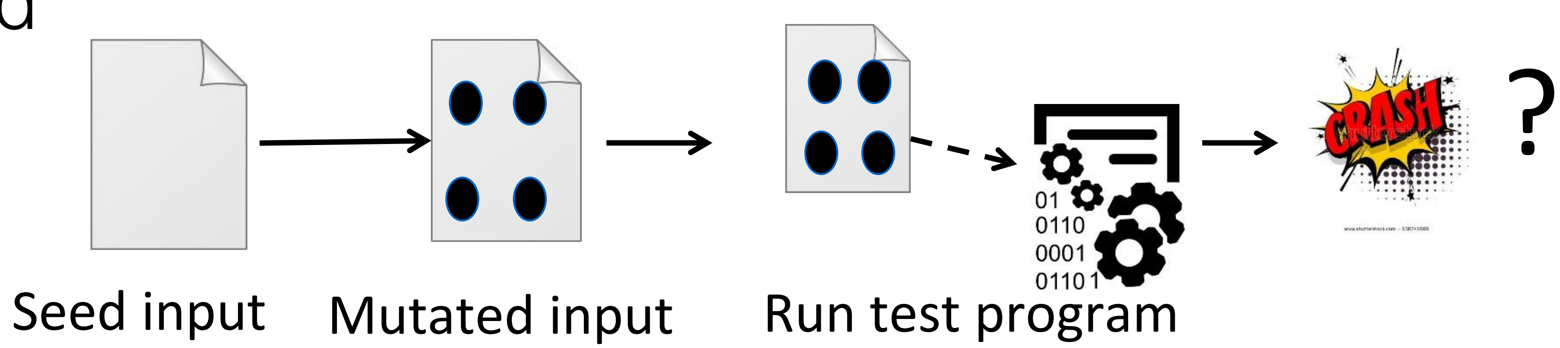
Mutation Based Fuzzing

- Mutation Based - “Dumb Fuzzing”: little or no knowledge of the structure of the inputs is assumed
- Anomalies are added to existing valid inputs
- Anomalies may be completely random or follow some heuristics (e.g. remove NUL, shift character forward)
- Examples:
    - Taof, GPF, ProxyFuzz, FileFuzz, Filep, etc.

Example: fuzzing a pdf viewer

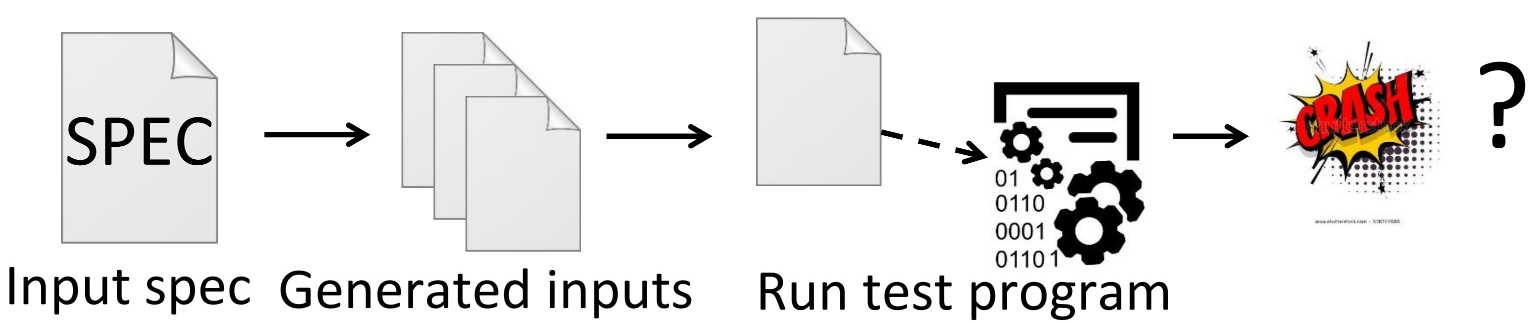
- Google for .pdf (about 1 billion results)
- Crawl pages to build a corpus
- Use fuzzing tool (or script to)
  1. Grab a file
  2. Mutate that file (optional)
  3. Feed it to the program
  4. Record if it crashed (and the input that crashed it)





Generation Based Fuzzing

- Generation Based - “Smart Fuzzing”: test cases are generated from some description of the format: RFC, documentation, etc.
- Anomalies are added to each possible spot in the inputs
- Knowledge of protocol should give better results than random fuzzing



Example: Protocol Description

```
//png.spk
//author: Charlie Miller

// Header - fixed.
s_binary("89504E470D0A1A0A");

// IHDRChunk
s_binary_block_size_word_bigendian("IHDR"); //size of data field
s_block_start("IHDRcrc");
        s_string("IHDR");  // type
        s_block_start("IHDR");
// The following becomes s_int_variable for variable stuff
// 1=BINARYBIGENDIAN, 3=ONEBYE
                s_push_int(0x1a, 1);    // Width
                s_push_int(0x14, 1);    // Height
                s_push_int(0x8, 3);     // Bit Depth - should be 1,2,4,8,16, based
   on colortype
                s_push_int(0x3, 3);     // ColorType - should be 0,2,3,4,6
                s_binary("00 00");      // Compression || Filter - shall be 00 00
                s_push_int(0x0, 3);     // Interlace - should be 0,1
        s_block_end("IHDR");
s_binary_block_crc_word_littleendian("IHDRcrc"); // crc of type and data
s_block_end("IHDRcrc");
...
```

Fuzzing Rules of Thumb

- Protocol specific knowledge very helpful
  - Generational tends to beat mutation-based, better spec's make better fuzzers
- More fuzzers is better
  - Each implementation will vary, different fuzzers find different bugs
- The longer you run, the more bugs you find
- Best results come from guiding the process
  - Notice where your getting stuck, use profiling!
- Code coverage can be very useful for guiding the process

Tool Support and Resources

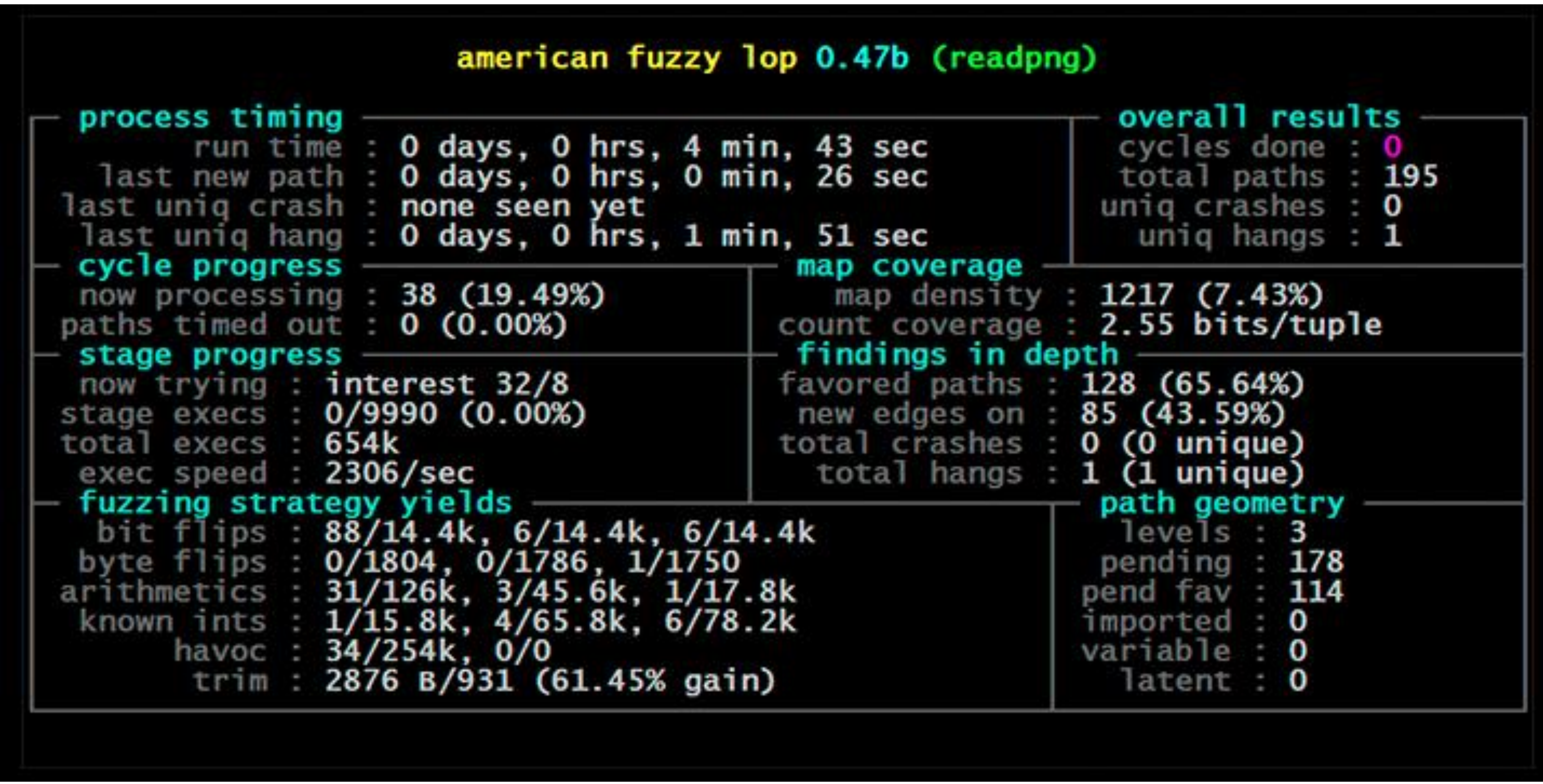
• American fuzzy lop is a fuzzer that employs genetic algorithms in order to efficiently increase code coverage of the test cases: http://lcamtuf.coredump.cx/afl/

• Randoop - a unit test generator for Java. It automatically creates randomized unit tests for your classes and creates JUnit tests.

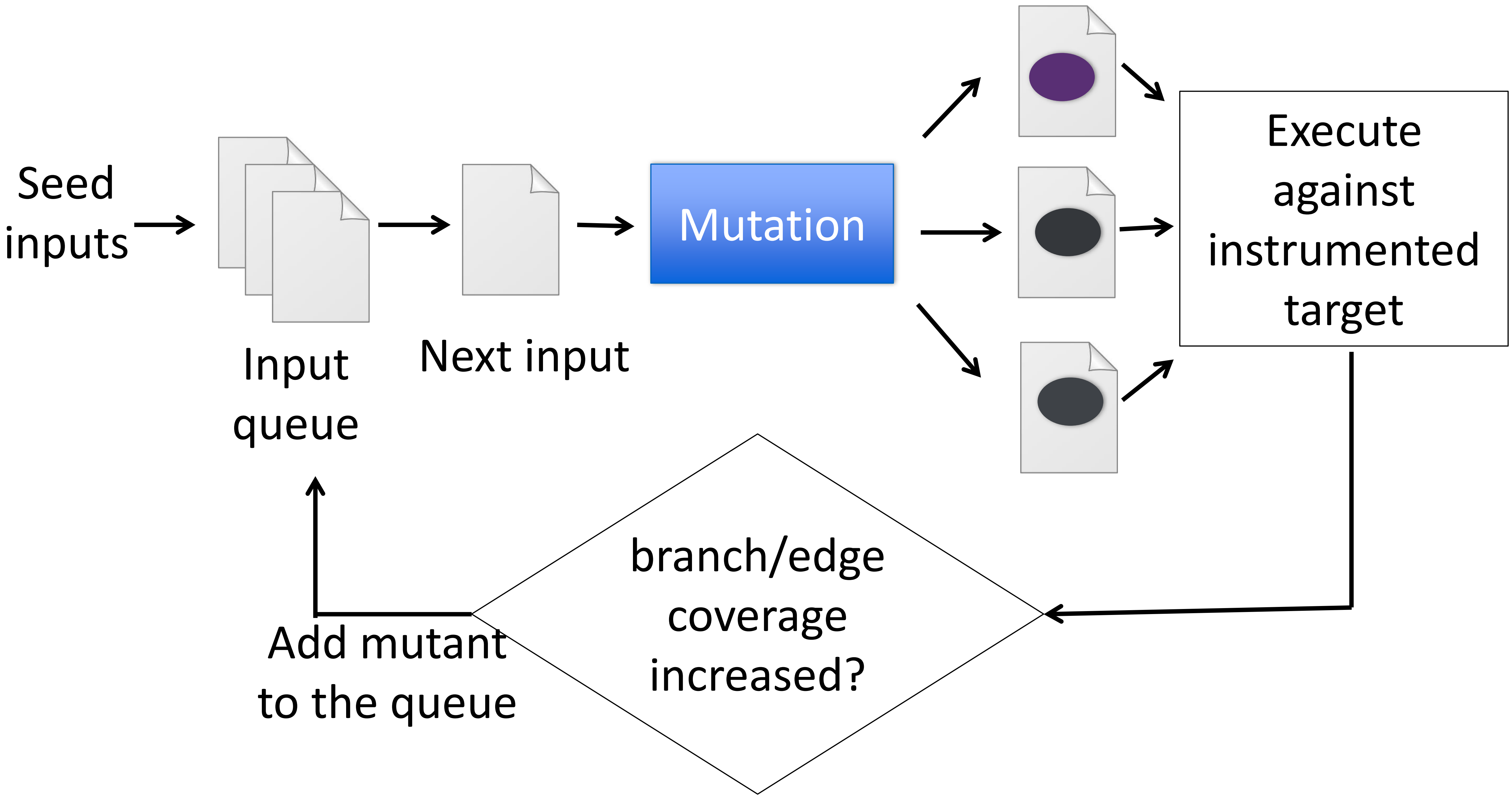
• Awesome-Fuzzing, https://github.com/secfigo/Awesome-Fuzzing#tutorials-and-blogs

American Fuzzy Lop (AFL)

http://lcamtuf.coredump.cx/afl/



American Fuzzy Lop (AFL)



Randoop

- Randoop - a unit test generator for Java. It automatically creates randomized unit tests for your classes and creates JUnit tests.

Randoop: Automatic u

https://randoop.github.io/randoop/

Randoop

Automatic unit test generation for Java

Download the **Latest Release**

View the **GitHub Project**

What is Randoop?

Randoop is a unit test generator for Java. It automatically creates unit tests for your classes, in JUnit format.

The Randoop manual tells you how to install and run Randoop.

How does Randoop work?

Randoop generates unit tests using feedback-directed random test generation. This technique randomly, but smartly, generates sequences of method/constructor invocations for the classes under test. Randoop executes the sequences it creates, using the results of the execution to create assertions that capture the behavior or your program. Randoop creates tests from the code sequences and assertions.

Randoop can be used for two purposes: to find bugs in your program, and to create regression tests to warn you if you change your program's behavior in the future.

Randoop's combination of randomized test generation and test execution results in a highly effective test generation technique. Randoop has revealed previously-unknown errors even in widely-used libraries including Sun's and IBM's JDKs and a core .NET component. Randoop continues to be used in industry, for example at ABB corporation.

Success Stories

- In August 2016, the Defense Advanced Research Projects Agency (DARPA) held the finals of the first Cyber Grand Challenge. The objective was to develop automatic defense systems that can discover software flaws in real-time. Fuzzing was used as an effective offense strategy to discover flaws in the software of the opponents. It showed tremendous potential in the automation of vulnerability detection. The winner was a system called "Mayhem" developed by the team ForAllSecure led by David Brumley.
- In Sep 2016, Microsoft announced Project Springfield, a cloud-based fuzz testing service for finding security critical bugs in software.
- In December 2016, Google announced OSS-Fuzz which allows for continuous fuzzing of several security-critical open-source projects.

https://en.wikipedia.org/wiki/Fuzzing

Random/Fuzz testing

- Advantages:
  - Intuitively simple
- Disadvantages:
  - The oracle problem: a test case is an input, an expected output, and a mechanism for determining if the observed output is consistent with the expect output.
  - Corner faults might escape detection.
  - Root cause analysis: debugging with a randomly generated input is challenging.

Thanks!


Testing
Software
development
design
articles
management
test
quality
code
web
use
risk
good
usability
Agile
People
Project
information
article
system
check
time
team
bug
Ruby
Tools
life
testers