[simula . research laboratory]

Random Testing

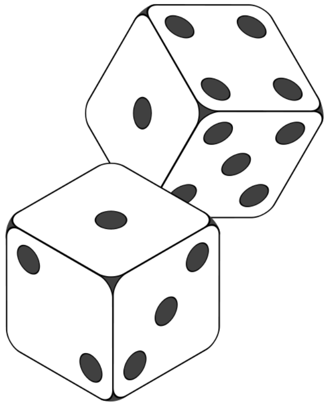
Dr. Andrea Arcuri
Simula Research Laboratory
arcuri@simula.no

Random Testing (RT)

RT: Testing strategy

Used to choose test cases

Main Message: Doing things at ***random*** can be ***good*** in software testing!



Random Testing: Part I

Some Definitions

SUT: software under test

S: set of all possible test cases for SUT

|S|: cardinality of S

void foo(int x, int y)

s in S is a pair (x,y), $|S|= 2^{32} * 2^{32} = 2^{64}$

F: subset of S, failing test cases

Failure Rate: $t=|F|/|S|$, probability that random test case fails if uniformly chosen

Example

```
int abs(int x){
if(x>0) return x;
   else return x; // should be -x
}
```

Failure rate: t~0.5

Ex.: `assertEquals(abs(-5),5);`

Test Cases and Inputs

assertEquals(r=SUT(i),e)

i: is the input for the SUT

r: obtained result, e.g. abs(-5)

e: expected result for r, e.g. 5 for abs(-5)

How to choose the inputs i???

Random is an option…

Random Testing (RT)

Choice of the input data at random

Cheap to implement

Easy to understand

Actually used in industry

Naïve? Useless?

We will see that it has *many positive sides*…

Random Generator

True random generators are expensive:

- e.g., atmospheric noise (see www.random.org)

In general, pseudo-random generators:

- Algorithms using for ex. linear congruential formula, as in java.util.Random

Randomness depends on sequence of sampled values and not on the source!

We will start to see again same pattern of values…

But pseudo-random is fine in most cases, because patterns are very long…

A close look at java.util.Random

```
public int nextInt() {  return next(32); }


synchronized protected int next(int bits) {
    seed = (seed * 0x5DEECE66DL + 0xBL) & ((1L
  << 48) - 1);
    return (int)(seed >>> (48 - bits));
}
```

Uniform Distribution

Each test case has same probability of being chosen, in particular P=1/|S|

Fine for numerical data: eg 0<x<100 , P=0.01

But some issues:

1. Complex Data Structure
2. Memory/Time Constraints
3. Test Case Length

Complex Data Structures

What if as input data we have trees, arrays, graphs, etc.???

Eg. “void foo(MyComplexGraph graph){…}”

One approach: consider binary representation

Eg. 00011100111101011100001111101110101…

Each bit at random

Problem: not all strings are valid input data!!!

Might end up to have valid data with very low probability!

Ad hoc generator

Difficult to guarantee uniform sampling, e.g. for graphs, trees, etc.

Memory/Time Constraints

A graph of 10 nodes takes less memory space than 10 billion nodes

Same concept for sequences of function calls

Need to put constraints on memory/time consumption of test data

If same probability of fault detection, better to run many quick test cases rather than only few slow ones!!! Because we would run more test cases

Uniform sampling: assuming same fault detection for each test case

Better to use other sampling distributions? Depends on assumptions we make on fault detection

ex.: “void foo(MyGraph graph)”

larger graphs lead to higher fault detection??? Maybe, maybe not…

Test Case Length

Different lengths to represent test data

Assume we put constraint L for length of binary string representation (length <= L)

We have 2^z sequence of length z!

2^z is more than the sum of all shorter sequences!!!

- Eg. $2^3 = 8 > 2^2 + 2^1 = 6$

Unwise to sample with uniform distribution!

- Most sampled string would be long 2^z, very unlikely to sample short sequences

One possible solution: first length at random, then random string of that length

Use of Random Testing

1. Sample test cases to trigger a failure

   Has the SUT any fault?

2. Sample test cases to cover testing targets

   E.g., code coverage

   Why? White and black box techniques are expensive

3. Reliability Estimation

   How reliable is the SUT?

   Discussed briefly, details in later lecture of the course

Finding Faults

All properties of RT derived from failure rate t.

Questions:

- How many test cases k to find a failure?
- Given k test cases, what is the probability of finding a failure?
- What is the variance of the results?
- Etc.

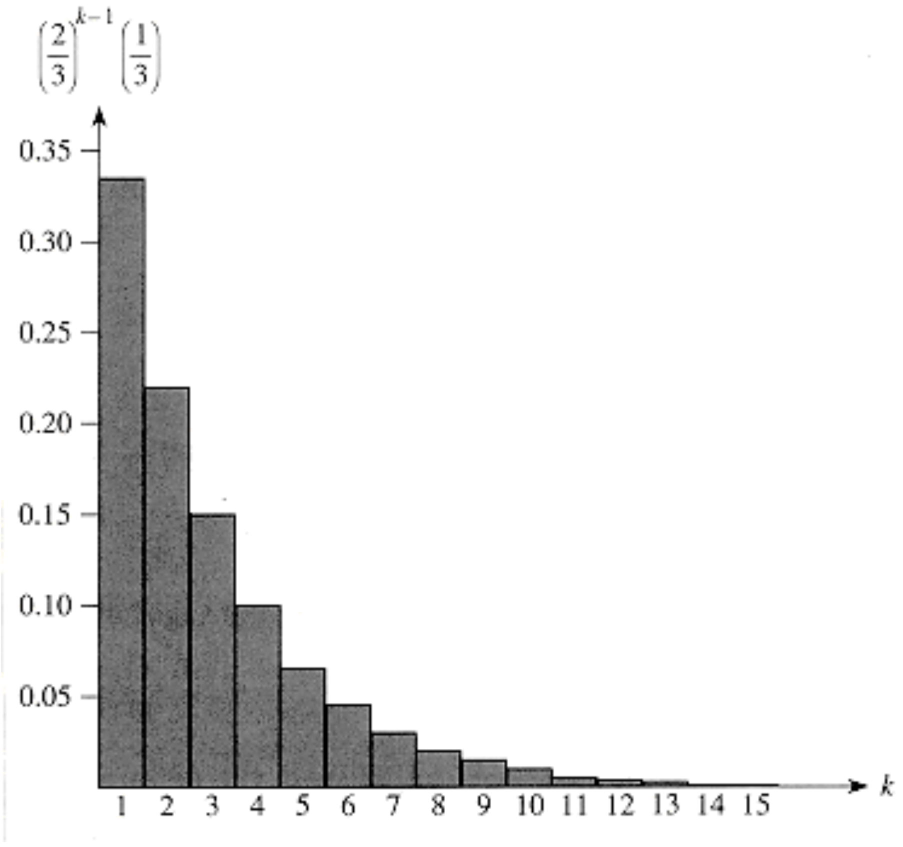
RT follows Geometric Distribution

P(RT=k), probability of triggering a failure after executing k test cases

P(RT=k) = $(1-t)^{(k-1)}$ t

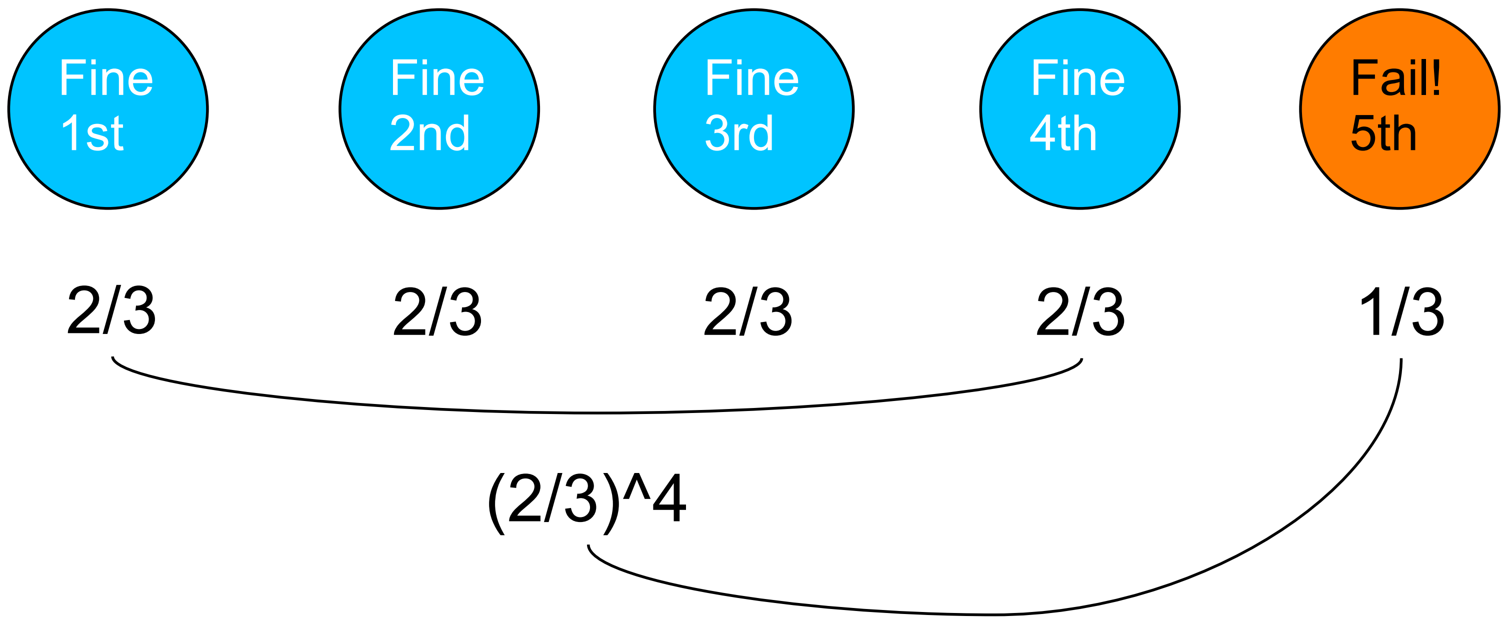
First (k-1) test cases should not fail, and that has probability (1-t) for each of them

Last test case fails with probability t



Example

k=5, t=1/3



$P(RT=5)=(2/3)^4 * (1/3)$

Some Other Properties

Mean: $1/t$

Variance: $(1-t)/(t^2)$

Median: $|-\log(2)/\log(1-t)|$

Example: $t=0.01$

Mean(0.01) = 100

Variance(0.01) = 9900

Median(0.01) = 69

Why bother?
It tells us exactly what to expect from RT

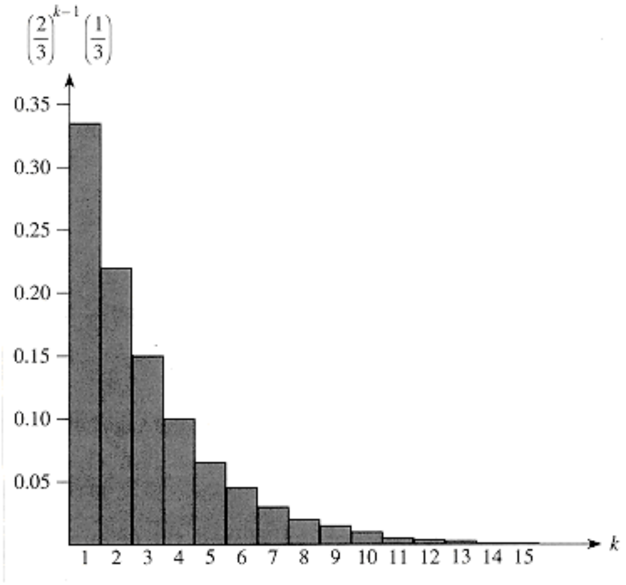
Some Consequences

Average value (mean) is 1/t

Median is in general ~70% of the mean

P of failure at the 1/t step is very low!!!

- $P(RT=1/t) = (1-t)^{(1/t-1)}t$
- With t=0.01, we have
- mean=100
- P(RT=100)=0.0036!!!
- ***k=1 has highest probability!!!***



Given k Test Cases

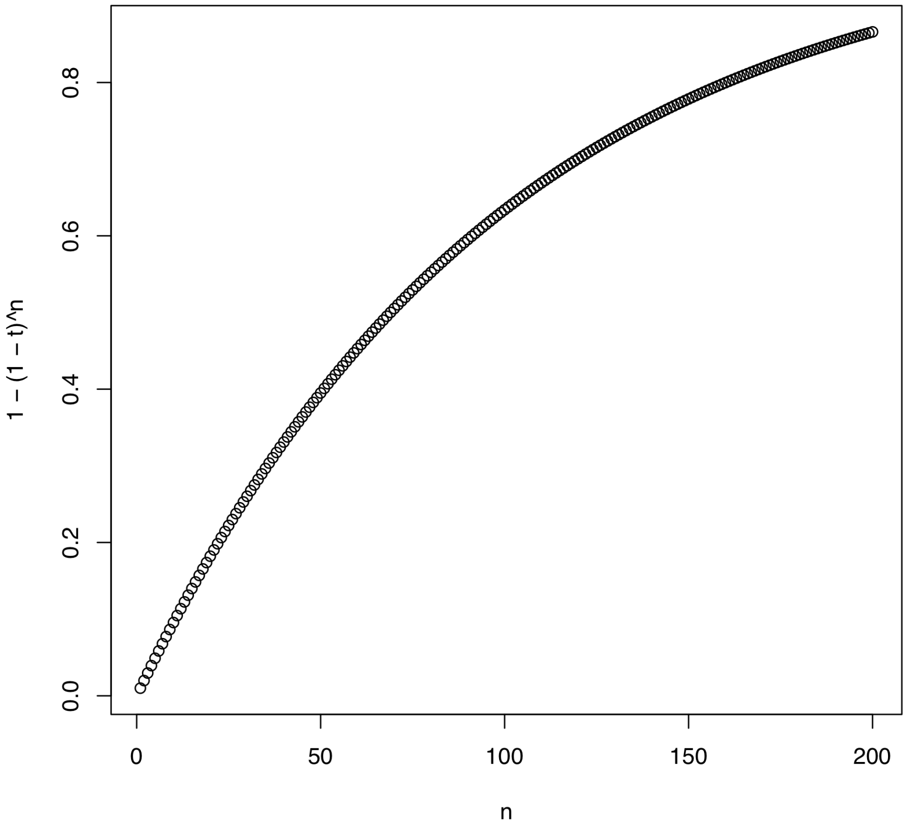
If there is a fault, I would like to have *at least one* test case that triggers a failure

No failure: P(no failure) = $(1-t)^k$

P of at least a failure?

P = 1 – P(no failure) =

$1 - (1-t)^k$



Some Comments

Usually failure rate t is unknown before testing…

But…it can be estimated…

- Previous projects
- Literature
- Type of software

…still important to understand RT dynamics

Estimations can be used to make choices such as:

- How many test cases to run and evaluate?
- When should we use RT?

RT for Coverage

Sample test cases at random

Execute them

Check the obtain coverage (ex., branches)

Keep sampling until desired coverage

Choose minimum subset of all sampled

- If no automated oracle
- If regression testing

Precise dynamics: *Coupon's Collector Problem*

- But quite complex math involved…

Some Notation

We have *n feasible* targets $\mathfrak{T}=\{\mathfrak{T}_1,\mathfrak{T}_2,\ldots,\mathfrak{T}_n\}$ and *m* unfeasible targets $\mathfrak{U}=\{\mathfrak{U}_1,\mathfrak{U}_2,\ldots,\mathfrak{U}_m\}$

Probability test case triggers $\mathfrak{T}_i$ is $t_i=|T_i|/|S|$

- T_i set of test cases that trigger $\mathfrak{T}_i$
- Note $|T_i|>0$ and $|U_i|=0$

Targets can be for example:

- Coverage elements (statements, branches, paths, etc)
- Assert statements we want to get failed (failure types), ex. for a list:
  - "assertTrue(list.size()==1); assertTrue(list.contains(5);)"

Main Assumption for Coverage

Targets $\mathfrak{T}$ are ***disjoint***

- A test case can trigger only one target $\mathfrak{T}_i$
- No problem for path coverage and failure type

In branch coverage targets are not disjoint

- Can still be mathematically analysed, but quite complex…

From Theory of Coupon's Collector

- If targets have some cardinality |Ti|, then we need k ~ n log(n) test cases on average to cover all targets
  - e.g., n=10 -> k = 10 * 2.3 = 23
- If targets do not have same probability:
  - (complex) equations exist…
  - Easiest scenario: all equal probabilities
- **IMPORTANT**: number *m* of unfeasible targets is ***irrelevant*** to k

RT vs. Partition Testing (PT)

PT: input domain divided into equivalent classes, choose at least one test case from each class, e.g. white and black box testing

Is it better to choose test cases at random or according to a coverage criterion such as branch coverage?

The answer is: *it depends…*

- Test cases for an equivalent class can be difficult to obtain
- Equivalent classes are not necessarily homogeneous (i.e., all failing or all passing)
- Waste of effort on unfeasible/empty classes
- Etc.

An Example

pre-condition: 0 <= x < 100

"void foo(int x){ if(x<80) … else … }"

Bug for x==0

PT: branch coverage

- half of sampled test cases in region [80,99]!

If 2 test cases, probability of triggering a failure?

- RT: 2 chances in range [0,99] , P=0.0199
- PT: 1 chance in range [0,79] , P=0.0125

RT full coverage?

- P= (0.8 * 0.2) + (0.2 * 0.8) = 0.32

When To Use RT

We have an automated oracle

Very complex problem:

- System Testing rather than Unit Testing

Rule of thumb:

- Always use first phase of RT and monitor coverage. Then you can apply other (more sophisticated) techniques if you really want /need to... ex., after fixing first bugs, if failure rate becomes too low to use RT

RT is not Perfect…

```
void foo(int x){
if(x==0){… /* faulty code here */}
}
```

Would need 4 billions test cases on average to trigger a failure…

PT would be better because:

- Fault in very small partition
- The partition is homogeneous

Real World Applications

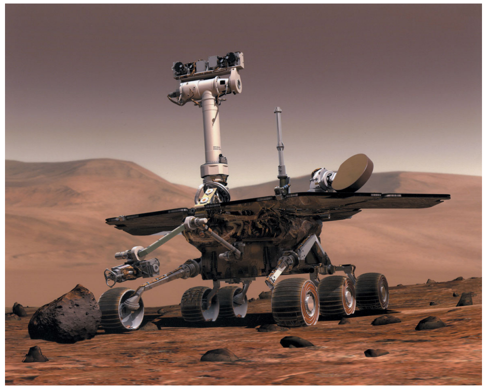
Several successful real-world applications

E.g., testing file system in software for space missions (Groce *et al.*, ICSE‘07)

Very complex system

Formal techniques failed

RT found several faults



Automation: reference file systems as oracle

Reliability Estimation

RT also used to estimate reliability

At the end, after testing for faults

RT's prob. distribution based on usage

- Not uniform!

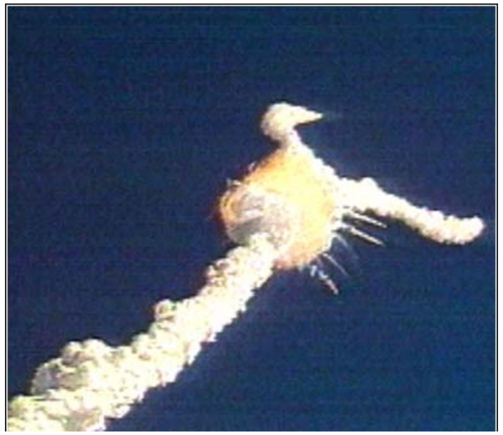
Operational profile (discussed later in the course)

Obviously, if found bug, should try to fix it

Several techniques using RT

Ariane 5

On June 4, 1996, the flight of the Ariane 5 launcher ended in a failure. It exploded after 40 seconds.



Likely Ariane 5 would have been saved by validation with actual trajectories based on operational profile

Random Testing: Part II

Can We Improve RT?

Let’s consider these 2 test suites of same size for “foo(int v)” :

X={1,2,3,4,5,6,7,8,9,10}

Y={-2345,12,342,-4443,2,3495437,-222223,24, 99343256,-524474}

Which is the best?

If we do not make any ***assumption***, the 2 test suites have ***same*** probability of finding failures!!!

What type of assumptions?

Some test cases might be more likely to reveal faults

Assumptions ***before*** execution

Assumption we analyse: **DIVERSITY**

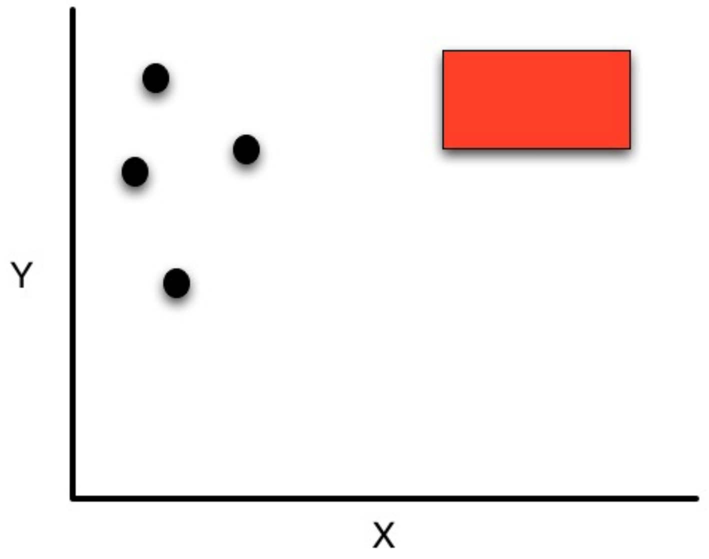
Given a set of test cases, higher probability of fault detection is obtained when the test cases are as diverse as possible among them

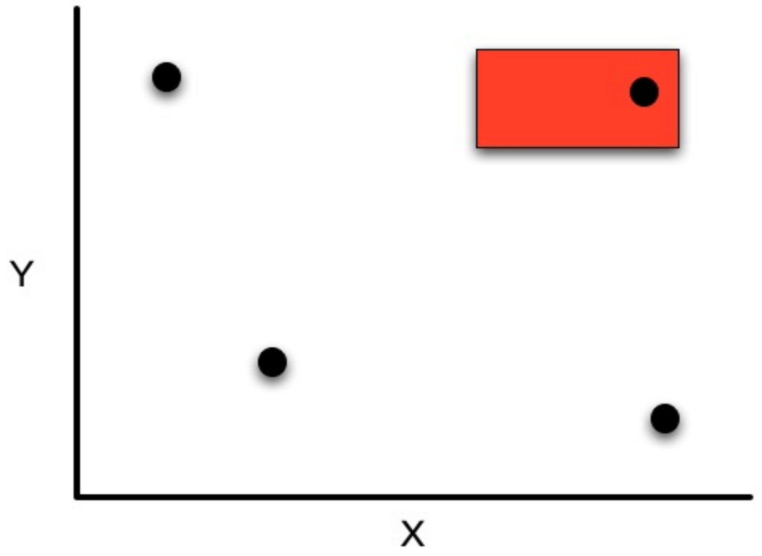
Other assumptions could be valid…

Overview

1. Motivation for *diversity* assumption
2. How to calculate diversity
3. How to exploit this knowledge

Contiguous Regions





Often failing test cases cluster in contiguous regions

“if(x>=0 && x<=10)”

Eg.: “void foo(int x, int y)”

More spread test cases: higher chances to hit faulty region(s)

Diversity depends on ***distance*** between test cases

Distance

d(s1,s2): distance between two test cases s1 and s2

If s1==s2, then d(s1,s2)=0

If input data is just integer for “void foo(int x)”:

- d(s1,s2) = |x1-x2| , e.g. d(-5,3)=8
- Where x is input data in test case s

But test data can be arbitrary complex!!!

- e.g., arrays, collections, sequence of function calls, etc.

Euclidean Distance

Numerical inputs

e.g., “void foo(int x, int y, int z)”

Test cases: s1=(x1,y1,z1) and s2=(x2,y2,z2)

$d(s1,s2) = \sqrt{(x1-x2)^2 + (y1-y2)^2 + (z1-z2)^2}$

In general:

$d(s1,s2) = \sqrt{\sum(v1,v2)^2}$, where v is data value in s

Sequence of Function Calls (SFC)

```
Container c = new Container();
c.insert(5);
c.insert(2);
c.remove(5);
assertEquals(c.size(),1);
```

What is the *distance* here?

From SFC to Symbols

First step: convert SFC in sequence of symbols (many possible ways to do it…)

Eg: name of function calls, input data, etc

SFC of previous ex.: (n,-,ins,5,ins,2,rem,5)

- mapped into 8 symbol string: (A,B,C,D,C,E,F,D)
- ins=C and 5=D are repeated twice

Any ***String Matching*** algorithm can be used

Hamming Distance

Count number of mismatches

s1 = (A,B,C,A,A)

s2 = (A,C,C,A,B)

d(s1,s2) = 2

But many other distances exist

Large literature in DNA analysis…

Diversity

Given subset G of test cases s from S

Given diversity function d

diversity(G) = $\sum$d(s1,s2)

for all pairs (s1,s2) in G

Adaptive Random Testing (ART)

Still choose test cases at random

One at the time

Add to current set G (initially empty)

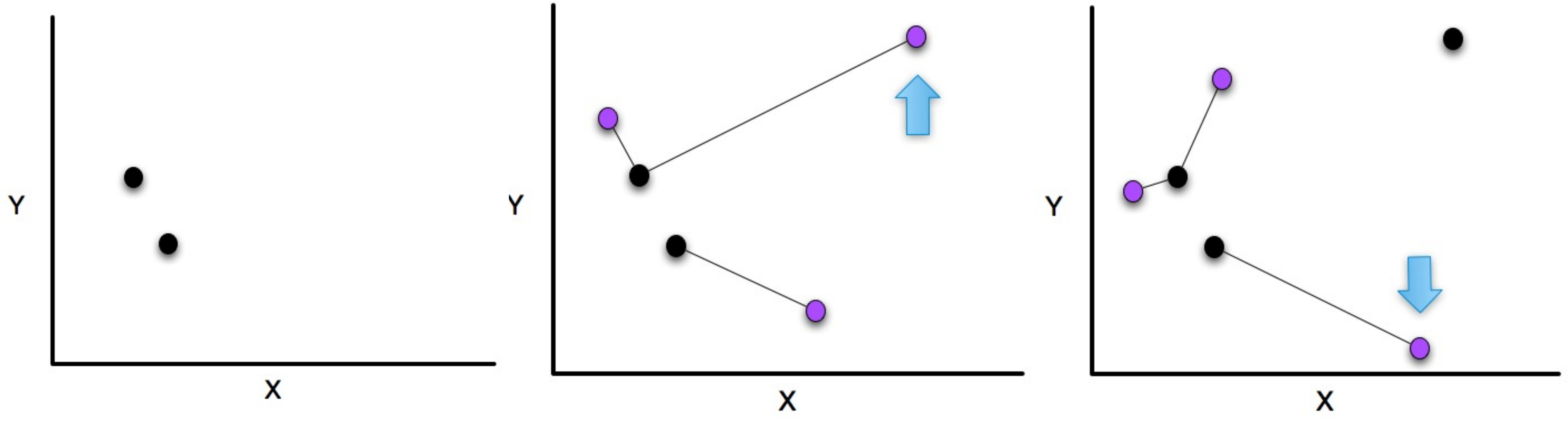
But when adding new one:

1. Sample Z test cases at random
2. Choose from Z the one that is most diverse (z) compared to current set G.
3. Add z to G

Most Diverse

Test case z in Z

Diversity of z from G: h(z)=min(d(z,g))



Complexity of ART

To choose k test cases, how many distance calculations?

Depending on execution time of test cases, distance calculation can be costly…

$N = 0 + |Z| + 2|Z| + \ldots + (k-1)|Z| = |Z|k(k-1)/2$

Given $|Z|$ as a constant: complexity $O(k^2)$

Variants of ART with fewer comparisons…

Comments on ART

Very easy to implement

Only challenge is definition of proper distance

Usually, ART is sensibly better than RT

There are several variants of ART

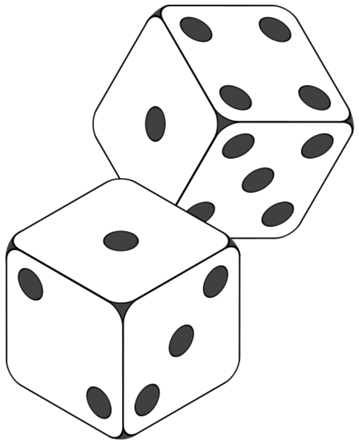
But…

ART is mainly ***academic***!

- Industries use RT but practically no knowledge of ART…

Conclusion

- Testing at random is good!
  - But do not play at random in a casino…
- (A)RT is effective and cheap to implement
- But need to know when and how to apply it
  - Good in many but not all testing problems…



References

W. Feller. An Introduction to Probability Theory and Its Applications, Vol. 1. Wiley, 3 edition, 1968.

J. W. Duran and S. C. Ntafos. An evaluation of random testing. IEEE Transactions on Software Engineering, 10(4):438–444, 1984.

S. C. Ntafos. On comparisons of random, partition, and proportional partition testing. IEEE Transactions on Software Engineering, 27(10):949–960, 2001.

A. Groce, G. Holzmann, and R. Joshi. Randomized differential testing as a prelude to formal verification. In IEEE International Conference on Software Engineering (ICSE), pages 621–631, 2007.

T. Y. Chen, F. Kuoa, R. G. Merkela, and T. Tseb. Adaptive random testing: The art of test case diversity. Journal of Systems and Software, 2009.