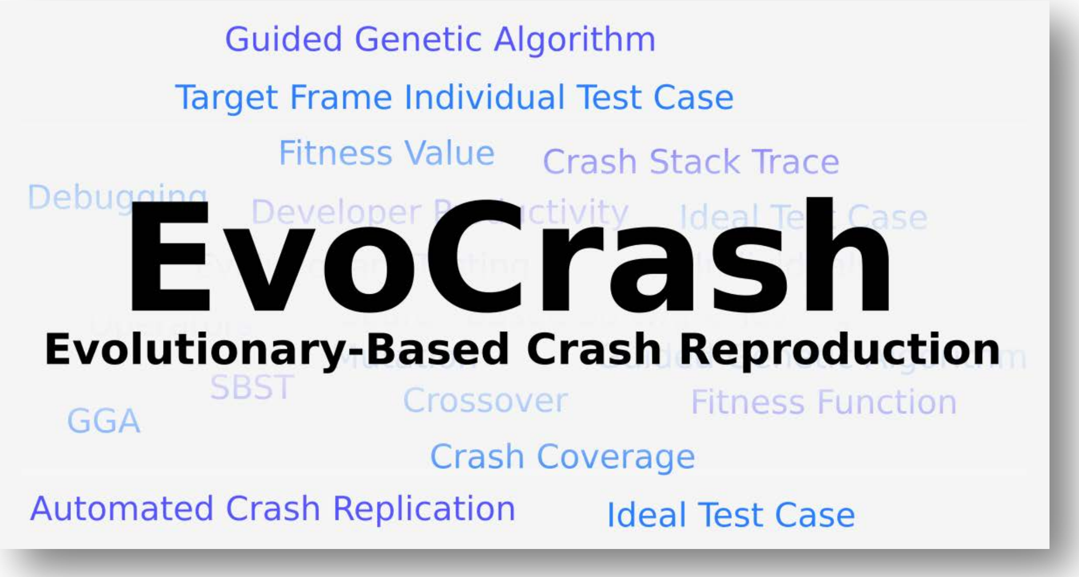


EvoCrash

Evolutionary-Based Crash Reproduction

Mozhan Soltani
PhD student
Software Engineering Research Group
Technical University of Delft

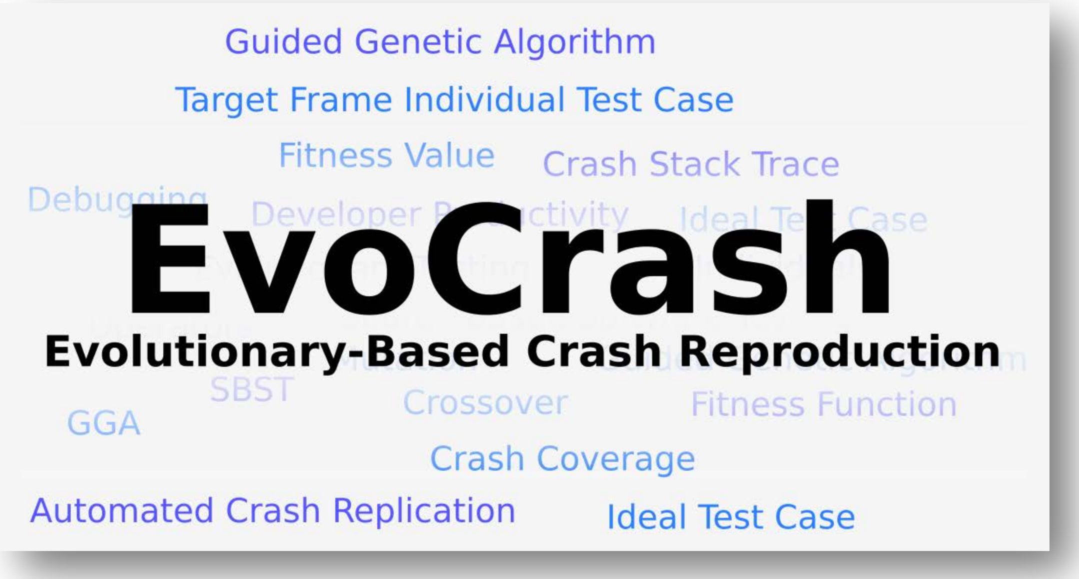
Let’s discuss the actions to take when debugging!

When debugging:

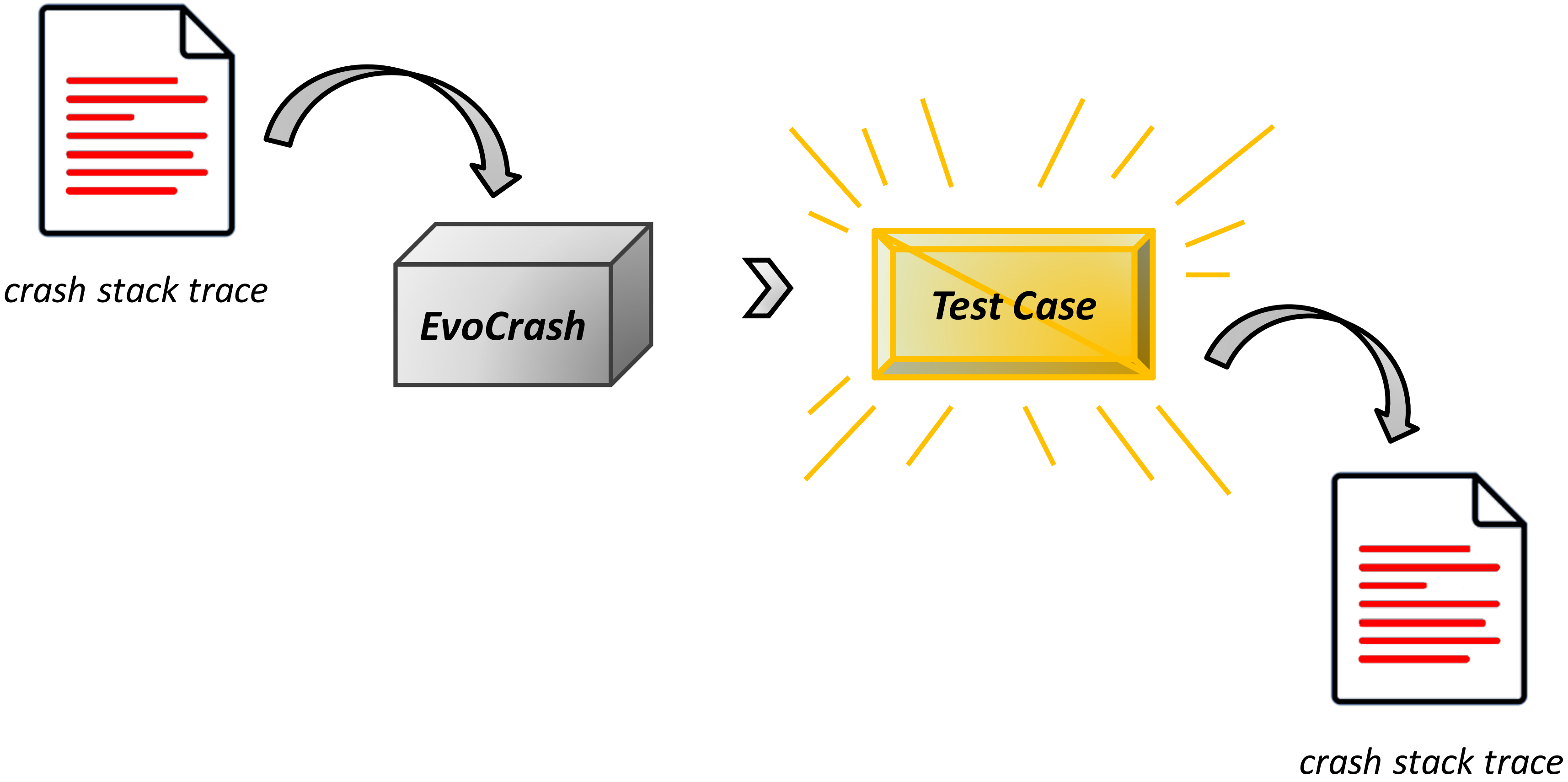
1. ***Confirm*** the crash actually happens (*reproduce it!*)
2. Identify the crash triggering conditions (*what is the defect?*)
3. Come up with a ***valid fix*** (*which does not introduce new defects!*)
4. Verify that the crash is *not reproducible* anymore

   (*add the crash reproducing test to the test suite!*)

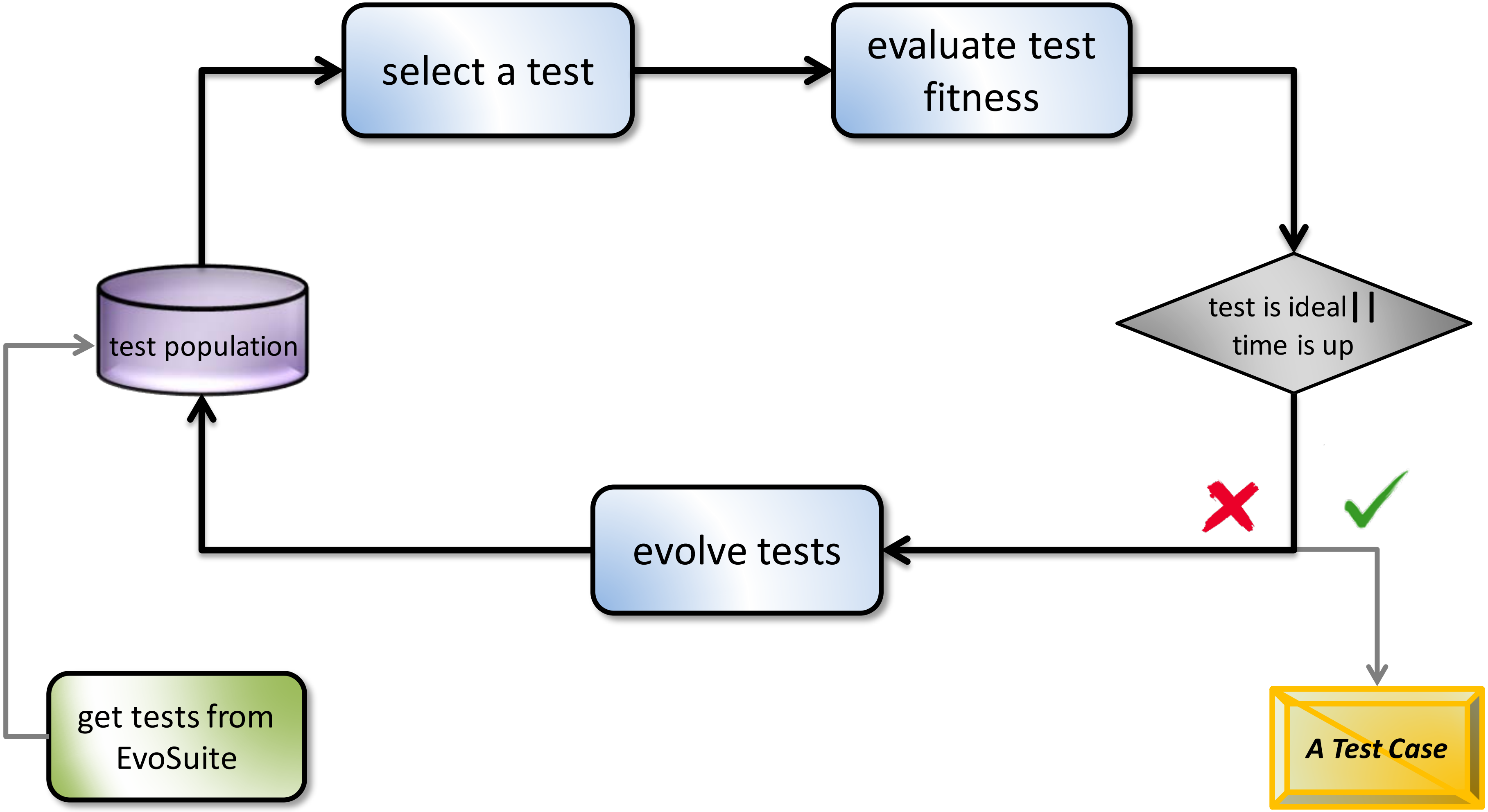
EvoCrash aims to support the first step (crash reproduction!)



How does EvoCrash do the magic?



EvoCrash is a *search-based* approach!



Search Based Software Engineering

Pool of possible solutions

Solutions shall address target criteria (*performance, code coverage, etc*)

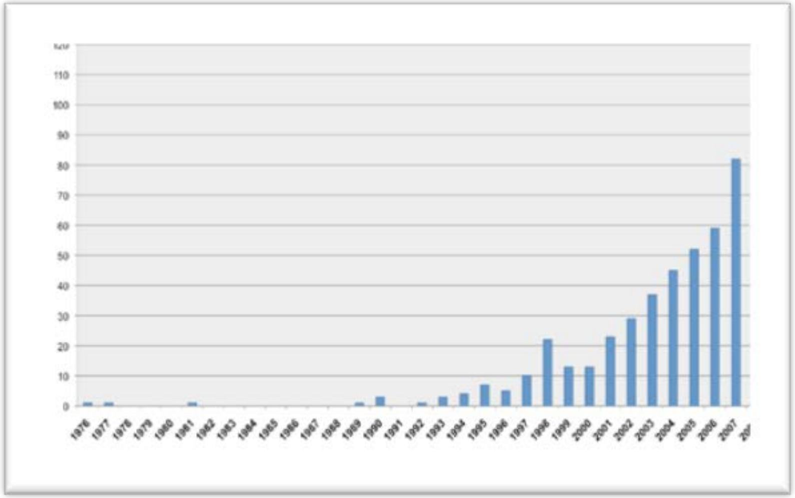
Search-**B**ased **S**oftware **E**ngineering is about:

1. Define the problem as a search problem
2. Define a ***fitness measure*** for comparing solutions

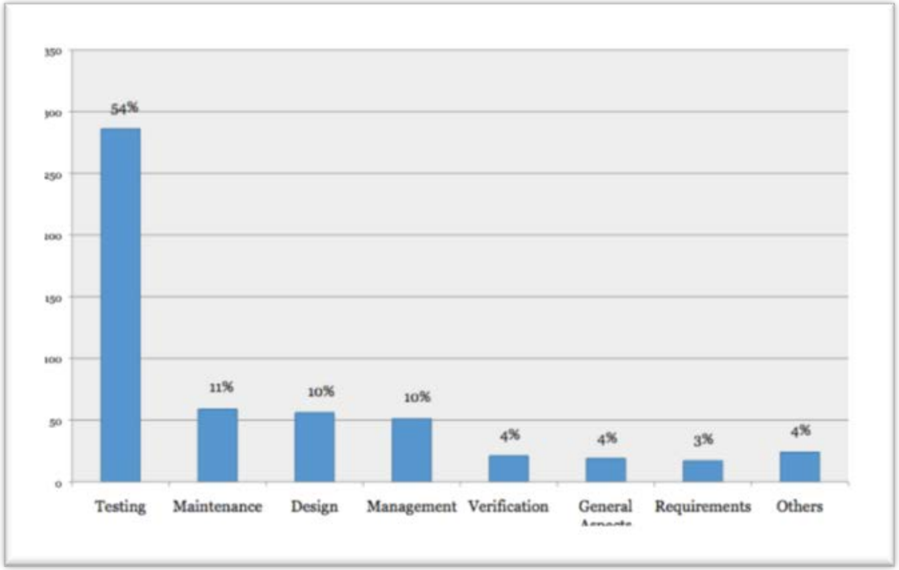
Automated search-based mechanisms make life much easier!

SBSE became popular!

Dramatic increase in the application of SBSE



Mostly applied in software testing!



Harman, M., Mansouri, S.A. and Zhang, Y., 2012. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)*, *45*(1), p.11.

Guidelines on applying SBSE

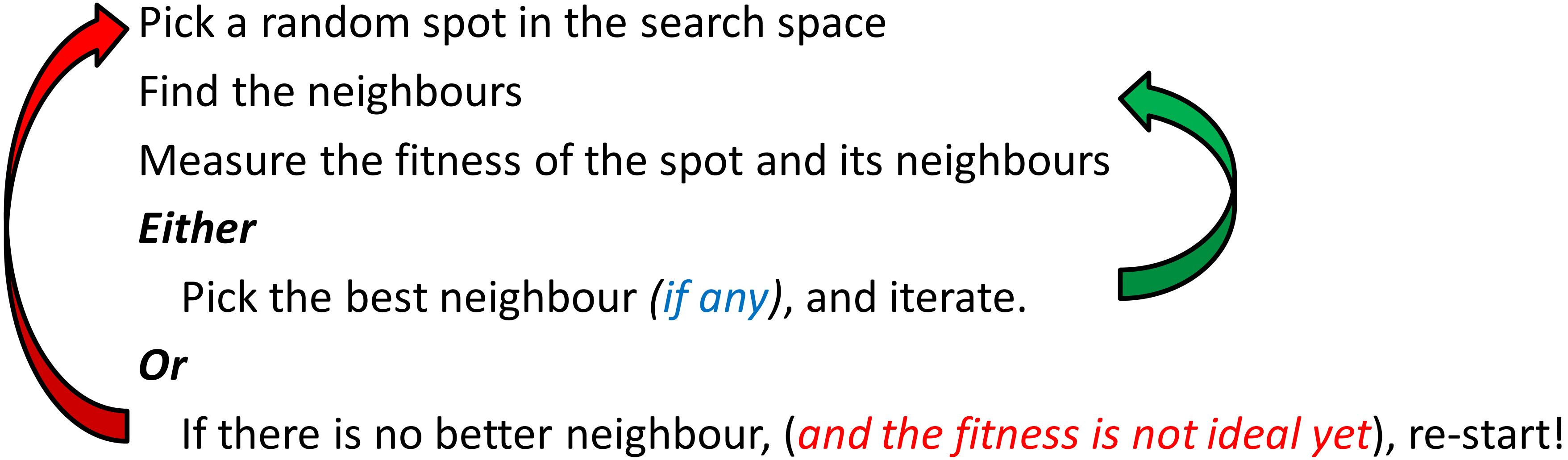
1. Choose a *representation* of the problem
   (*simple, yet accurate enough to show the difference among individuals!*)
2. Define the *fitness function*
   (*not too expensive to run since there will be many comparisons among individuals!*)
3. Implement a *random* search
   (*to evaluate the outcome of the fitness function for various random inputs!*)
4. Implement a *simple* search algorithm (*e.g. Hill climbing*)
   (*Maybe a simple heuristic search is sufficient for the search problem at hand!*)
5. If needed, try *other* search algorithms
   (*Maybe Genetic Algorithms to find the global optimum in a large search space*)
6. Analyze and compare the results
   (*Statistically compare effectiveness and efficiency of the implemented search-based approaches!*)

Harman, M., McMinn, P., De Souza, J.T. and Yoo, S., 2012. Search based software engineering: Techniques, taxonomy, tutorial. In *Empirical software engineering and verification* (pp. 1-59). Springer Berlin Heidelberg.

Search Algorithms

- **Exhaustive search**
  (*try all possibilities*)
- **Random search**
  (*try random solutions*)
- **Heuristic search**
  (*pick a random spot,*
  assess its fitness,
  and explore the search space accordingly)
  - Hill climbing
  - *[Simulated Annealing]*
  - GAs
  - …

Hill Climbing – *a local search*



Pick a random spot in the search space

Find the neighbours

Measure the fitness of the spot and its neighbours

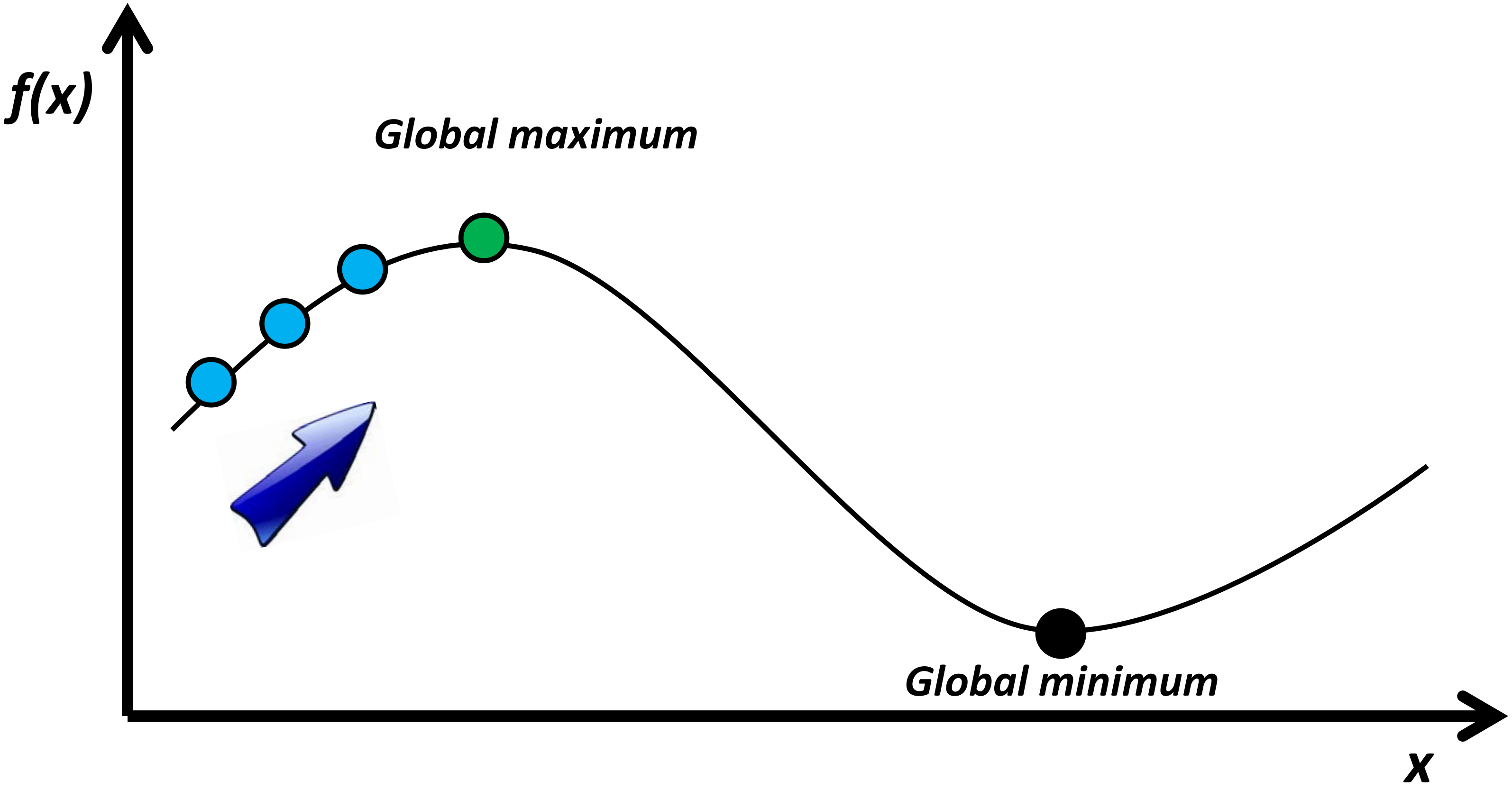
Either

Pick the best neighbour *(if any)*, and iterate.

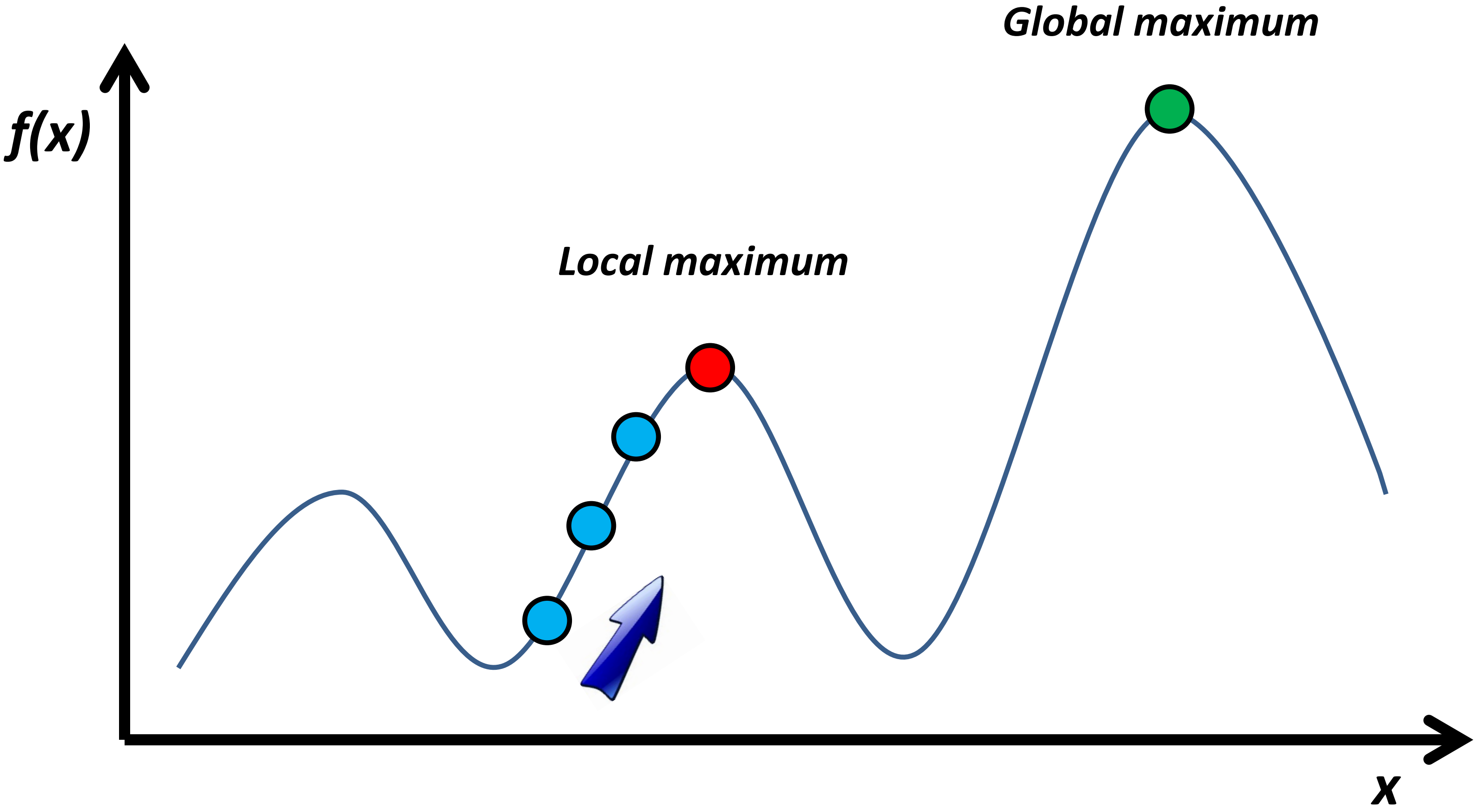
Or

If there is no better neighbour, (*and the fitness is not ideal yet*), re-start!

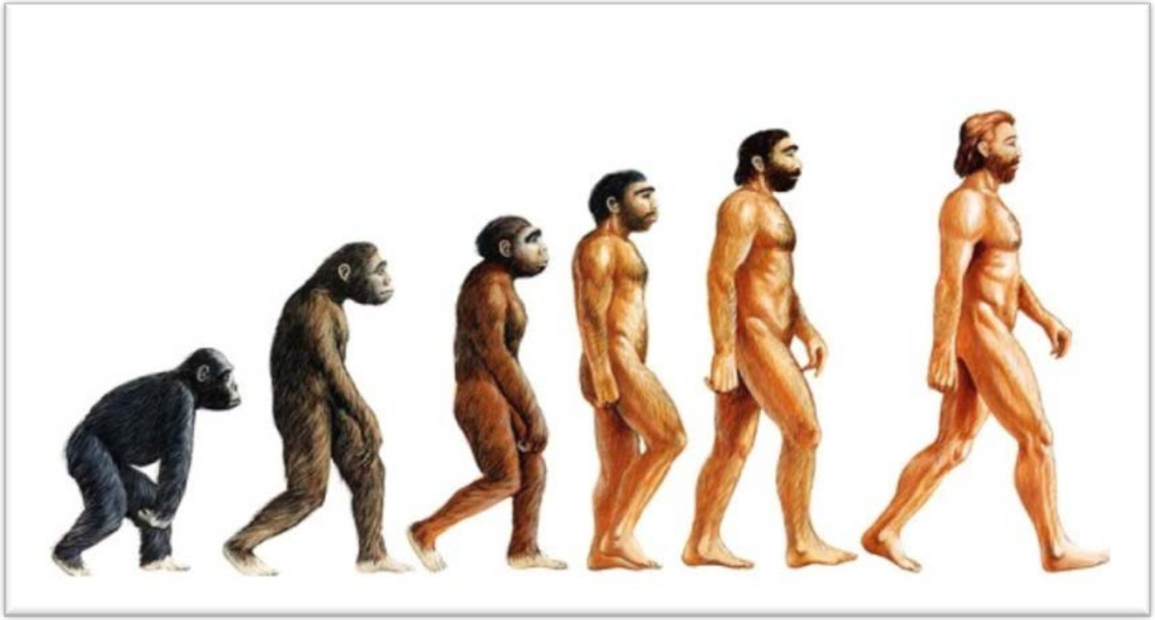
Hill climbing –
let's find the global maximum



Hill climbing –
let's find the global maximum



Genetic Algorithms – *global search*



Inspired by the *natural selection* in biology

Selection: fitting individuals are selected

Reproduction: They reproduce offsprings

Mutation: Offsprings inherit properties and evolve

Overview of GAs

1. Randomly generate initial population *P* (or seed some individuals)
2. Loop until search budget is consumed or **the one** is found!

    Evaluate fitness of each individual in *P*

    Select parents

    Generate offsprings (***crossover***)

    Make a *P'* from the parents and offsprings

    Mutate *P'*

    Insert *P'* to *P*

What is an individual?

Each individual is made up of components called "*chromosomes*", e.g.:

A vector of binaries: <1,0,0,0,1,0>

A vector of characters: <a,g,s,j,r>

Or even a **sequence of statements** in a test case!!

How to produce offsprings?

Parent-A: **<**1,1,1,1,0,0,1**>**
Parent-B: **<**0,0,1,1,0,1,1,0**>**

One-point crossover:
1) Pick a random position,
2) Swap the elements up to that position.

Example:
Let’s pick position 4!

Parent-A: **<**1,1,1,**1**,0,0,1**>** ⇨ **Child-A**: **<** 0,0,1,**1**,0,0,1**>**
Parent-B: **<**0,0,1,**1**,0,1,1,0**>** **Child-B**: **<** 1,1,1,**1**,0,1,1,0**>**

How to ***mutate*** an individual?

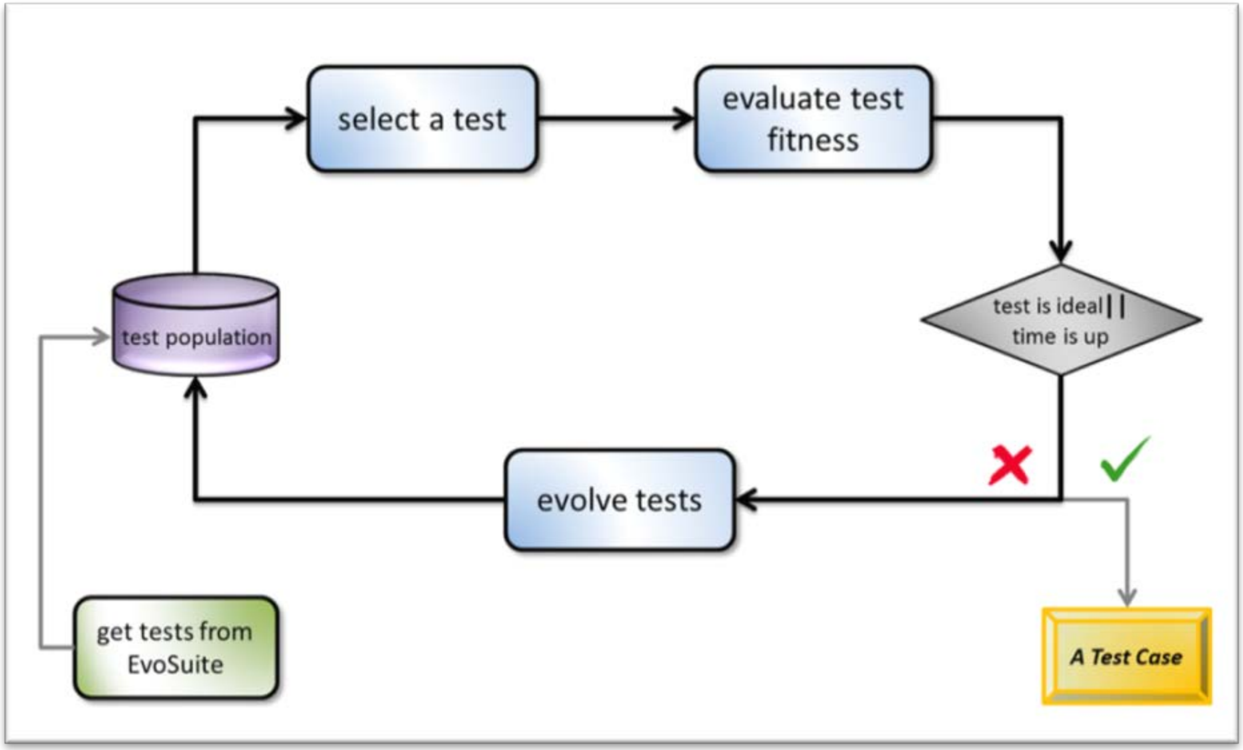
With the probability of 1/n (*n is the length of the individual*), modify some chromosomes!

A: <2,0,3,1,0,5,1,0> *add* ⇨ **A’:** <**9**,2,0,3,1,0,5,1,0>

A: <2,0,3,**1**,0,5,1,0> *change* ⇨ **A’:** <2,0,3,**8**,0,5,1,0>

A: <2,0,3,1,0,**5**,1,0> *remove* ⇨ **A’:** <2,0,3,1,0,1,0>

Let me hear your questions!



I hear you ask…

1. What tests are generated by EvoSuite?
2. How to evaluate the fitness of a test?
3. How is a test evolved?
4. What if the search time is over and there is no ideal test?

Whole test suite generation via ***EvoSuite***

"EvoSuite is a tool that automatically generates test cases with assertions for ***classes*** written in Java code."

In the whole test suite generation approach, the suite is optimized based on *multiple* coverage criteria at the same time!

So an individual here is a test suite, and the originally implemented optimization criteria: *branch coverage* & *test suite length*!

http://www.evosuite.org/evosuite/

Fraser, G. and Arcuri, A., 2013. Whole test suite generation. IEEE Transactions on Software Engineering, 39(2), pp.276-291.

So how are the test suites evolved?

Crossover: swaps test cases of two test suites

Mutation: a bit complicated!

1) Each test case is mutated with the probability of 1/n *(n is the size of the suite)*
2) A test case could be inserted, changed, or deleted.

N.B. when the test case is changed, mutation on the test is applied!

(for details on probabilities of these operations please refer to the paper!)

Fraser, G. and Arcuri, A., 2013. Whole test suite generation. IEEE Transactions on Software Engineering, 39(2), pp.276-291.

But for EvoCrash we need a single test case!

So how to apply EvoSuite?

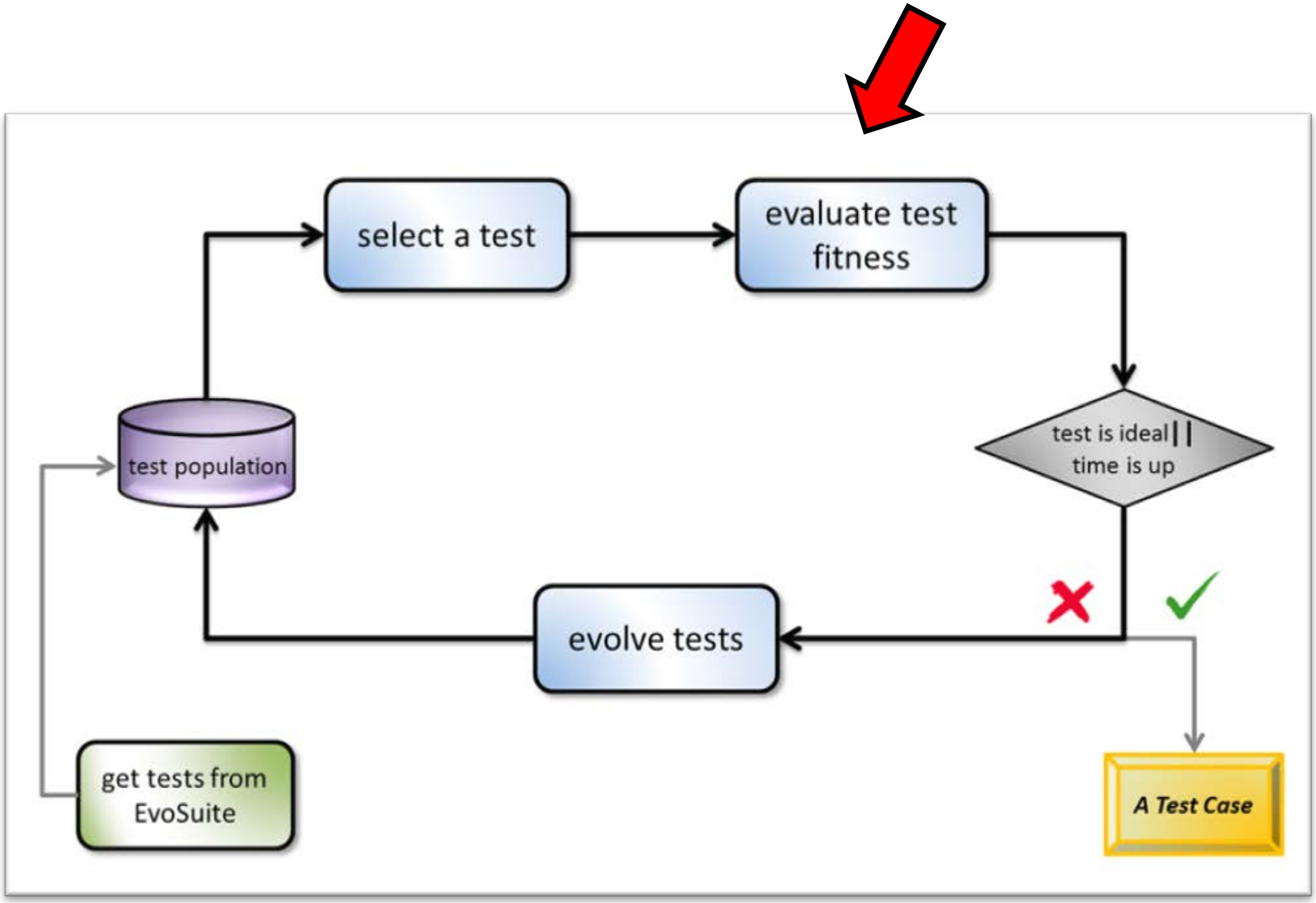
Well, as it is implemented now, we need to pick a **target class** for which the initial tests are produced by EvoSuite.

```
java.lang.ArrayIndexOutOfBoundsException:
   at org.apache.tools.ant.taskdefs.Concat$MultiReader.read(Concat.java:784)
   at org.apache.tools.ant.taskdefs.Concat.concatenate(Concat.java:513)
   at org.apache.tools.ant.taskdefs.Concat.cat(Concat.java:462)
   at org.apache.tools.ant.taskdefs.Concat.execute(Concat.java:371)
✓  at org.apache.tools.ant.UnknownElement.execute(UnknownElement.java:269)
```

So a typical unit test that looks like:

```
@Test(timeout = 4000)
  public void test0()  throws Throwable  {
      Object object0 = new Object();
      LinkedList<String> linkedList0 = new LinkedList<String>();
      Object[] objectArray0 = new Object[6];
      objectArray0[0] = object0;
      // Undeclared exception!
      CollectionUtils.collect((Collection) linkedList0, (Transformer)
      null, (Collection) linkedList0);
}
```

How do we evaluate the fitness?



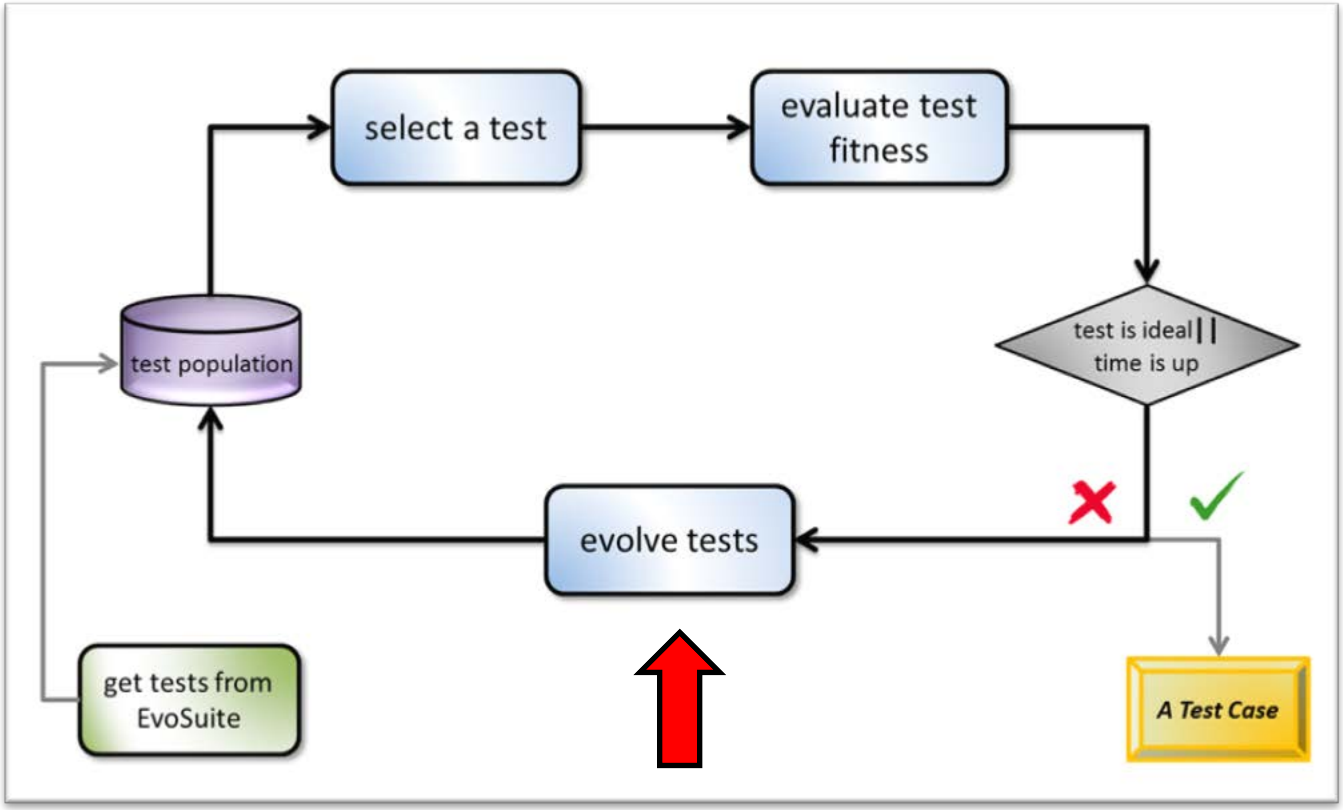
How to evaluate fitness of a test?

```
java.lang.ArrayIndexOutOfBoundsException:
    at org.apache.tools.ant.taskdefs.Concat$MultiReader.read(Concat.java:784)
    at org.apache.tools.ant.taskdefs.Concat.concatenate(Concat.java:513)
    at org.apache.tools.ant.taskdefs.Concat.cat(Concat.java:462)
    at org.apache.tools.ant.taskdefs.Concat.execute(Concat.java:371)
  ✓ at org.apache.tools.ant.UnknownElement.execute(UnknownElement.java:269)
```

When running a candidate test, we check:

1. Is the target line number covered?
2. Is the target exception thrown?
3. How similar is the generated stack trace to the target trace?

How do we evolve the test?

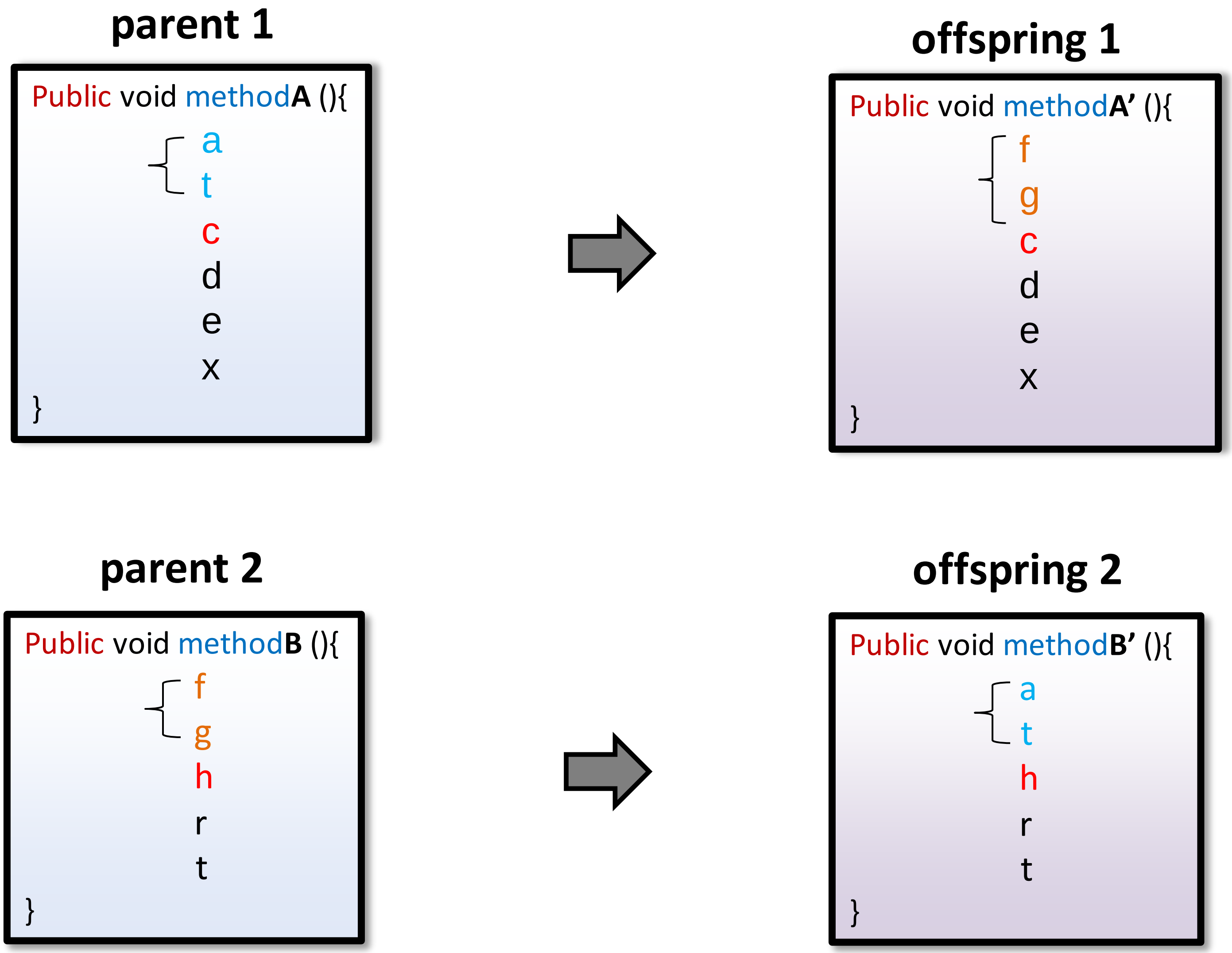


How is a test evolved?

We apply two standard ***GA*** operators:

- *Crossover*
- *Mutation*

How does *Crossover* work?



parent 1

```
Public void methodA (){
        a
        t
        c
        d
        e
        x
}
```

parent 2

```
Public void methodB (){
        f
        g
        h
        r
        t
}
```

offspring 1

```
Public void methodA’ (){
        f
        g
        c
        d
        e
        x
}
```

offspring 2

```
Public void methodB’ (){
        a
        t
        h
        r
        t
}
```

How does mutation work?

```
Public void methodA’ (){
            z (m)
            a (i)
            b (input2)
            c (j)
            d (p)
            t (q)
}
```

possible mutations on a test case

- A new method statement is added
- Input argument is altered
- A method statement is removed

What if the search time is over?

Well, the search was not successful, so an empty test is written to the target test directory!

Possible MSc thesis topics for you!

Problem:

How do we know which frame level to target from the crash stack at the beginning?

Proposal:

Let’s target them all, then analyze their relevance!

Direct the search process towards covering multiple target classes (*multi-target optimization*)

```
java.lang.ArrayIndexOutOfBoundsException:
   at org.apache.tools.ant.taskdefs.Concat$MultiReader.read(Concat.java:784)
✓  at org.apache.tools.ant.taskdefs.Concat.concatenate(Concat.java:513)
✓  at org.apache.tools.ant.taskdefs.Concat.cat(Concat.java:462)
✓  at org.apache.tools.ant.taskdefs.Concat.execute(Concat.java:371)
✓  at org.apache.tools.ant.UnknownElement.execute(UnknownElement.java:269)
```

EvoCrash to the rescue!

Question:

How helpful is EvoCrash to developers in practice?

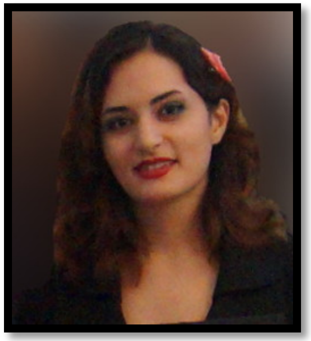
Answer:

We shall investigate the degree to which EvoCrash aids debugging in practice, so that we can explore and identify the directions towards further improvements.

Let’s seek industrial opportunities!

Interested in EvoCrash?

Let’s meet and discuss ideas for your MSc thesis!



m.soltani@tudelft.nl

Your assignment!

You are asked to debug 2 programs with and without using EvoCrash tests.

You will write a report based on the two cases you practiced with.

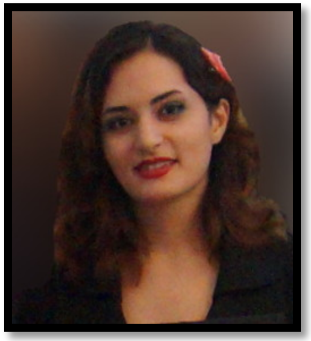
- You shall do the assignments individually!
- However, after the assignment is done, you may share your results and experiences with your group-mates to discuss and produce the reports.

Your assignment!

Please note:

- We collect data to assess if using the tests had any impact on your debugging practice.
- We will use the data **anonymously** and only share among the **EvoCrash team**.
- Your performance during the assignment will have **no impact** on your course grades! (you are graded based on the group reports!)
- The assignment may take up to **2 hours**, so please be prepared for that!

The EvoCrash team appreciates your participation in advance!



Mozhan Soltani
PhD student
Software Engineering
Research Group
TU Delft

Dr. Annibale Panichella
SnT center
University of Luxemburg

Prof. dr. Arie van Deursen
Software Engineering
Research Group
TU Delft

Dr. Mauricio Aniche
Software Engineering
Research Group
TU Delft