

Interactivity in the generation of test cases with evolutionary computation



UNIVERSIDAD
DE CÓRDOBA



UCA

Universidad
de Cádiz

Aurora Ramírez (Univ. Córdoba, Spain)

Pedro Delgado-Pérez (Univ. Cádiz, Spain)

Kevin J. Valle-Gómez (Univ. Cádiz, Spain)

Inmaculada Medina-Bulo (Univ. Cádiz, Spain)

José Raúl Romero (Univ. Córdoba, Spain)



2021 IEEE

CONGRESS ON EVOLUTIONARY COMPUTATION
28.06-1.07.2021 • Kraków • POLAND

Outline

1. Introduction and problem analysis

- *Motivation*
- *Requirements*

2. Interactive optimization for the test generation problem

3. A proof of concept

- *Interactive options in EvoSuite*
- *Illustrative example*

4. Conclusions and future work

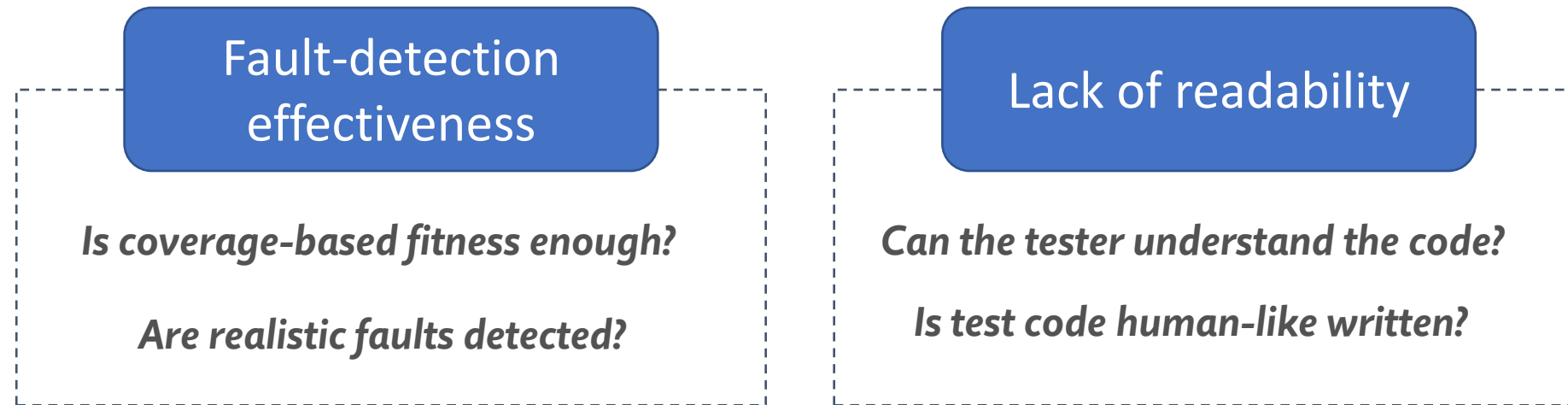
Introduction and problem analysis

Motivation

Automated **test case/suite** generation

- Efficient alternative to a costly manual process
- Frequently solved with fully automated evolutionary algorithms (SBST)

Two main **limitations** remain:



Introduction and problem analysis

Requirements

Requirement 1:

White-box testing requires a **broad knowledge of the source code** of the classes under test.

Requirement 2:

Testers should be able to **incorporate their knowledge and preferences** to the test generation process.

Requirement 3:

Search-based test generation tools should be able to **cope with two limitations: *detection power* and *readability***.



Interactive optimization for the test generation problem

- 1 Type of interactive algorithm**
 - Human-based evaluation → Oriented to **readability** improvement (*subjective*)
 - Human-guided search → Oriented to better **fault-detection** capability
- 2 Type of human actions**
 - *Evaluation*
 - **Scores** for detection capability / readability
 - **Reward/penalize** test code sequences
 - **Weight objectives** (*multi-objective approach*)
 - *Modification*
 - Edit **arguments** in method calls
 - Add **complex objects** or data structures
 - Specify **methods** that should be combined
 - Add complex **assertions**
 - *Selection*
 - Choose the best solution among those detecting the same mutant
 - Choose the best solution according to its readability



A. Ramírez, J.R. Romero, C. Simons. *A Systematic Review of Interaction in Search-Based Software Engineering*. IEEE Trans. Software Engineering, 45(8):760-781. 2019.

Interactive optimization for the test generation problem

3

Interaction mechanism

- *Time and frequency*
- *Solution selection*
- *Level of detail*

- **Adaptive**: n trials without improvement, coverage threshold, detected mutant
- **Fixed**: every n iterations
- **On demand**: the tester pauses the search
- **Best solution**(s) according to the fitness value
- **All solutions** detecting the same target mutant
- **Specific criterion**, e.g., number of detected mutants
- **Complete** solution
- **Test case** together with the **lines affected by the mutation**

4

Feedback integration

- *Information lifetime*
- *Information validity*

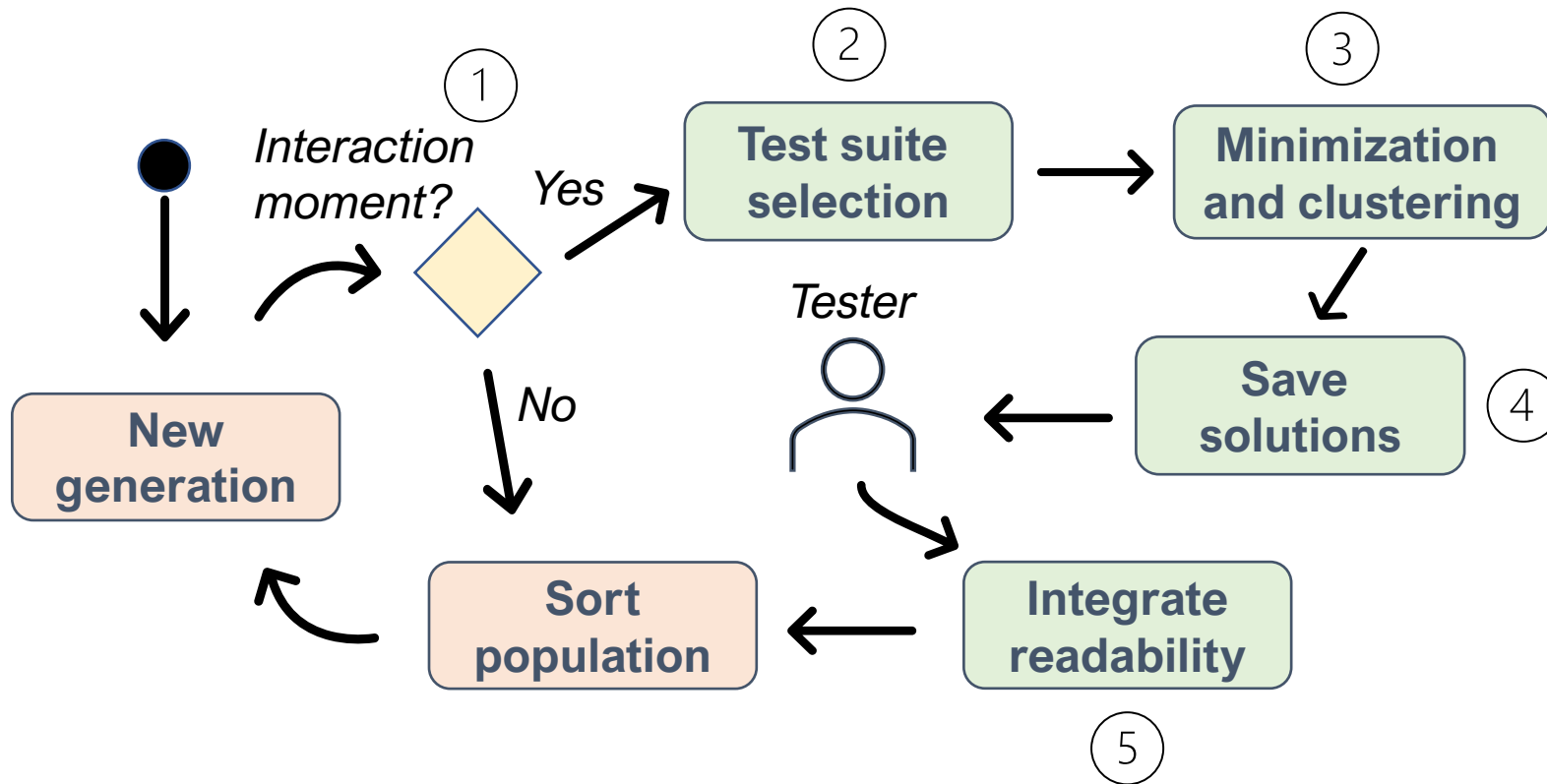
- **Mutation-based**: applied to a *test case – mutant* pair
- **Short-term**: evaluation and modifications are transferred to other solutions
- **Long-term**: tester's preferences are saved for other executions
- **Permanent**: tester's feedback remains unaltered
- **Flexible**: tester's feedback can be revisited



A. Ramírez, J.R. Romero, C. Simons. *A Systematic Review of Interaction in Search-Based Software Engineering*. IEEE Trans. Software Engineering, 45(8):760-781. 2019.

A proof of concept

Interactive options in EvoSuite



- ① **Interactions happen when:**
 1. Best coverage > threshold (1st)
 2. Every N generations (after 1st)
 3. Max. Num. Interactions not reached
- ② **Interaction to break ties:** choose the solutions with best fitness.
- ③ Apply EvoSuite **minimization procedure** and keep different test suites only.
- ④ **Save the candidate test suites** for subjective evaluation.
- ⑤ **Assign tester's readability score** as secondary objective.

A proof of concept

Illustrative example

(1) Search until first interaction

When the coverage of the best candidate is greater or equal to *Revise_after_percentage_of_coverage* (88%), the **secondary objective is enabled** (15th generation).

The screenshot shows the Eclipse IDE with the following components:

- Editor:** Displays the `updateSecondaryObjectiveCoverage` method in `GeneticAlgorithm.java`. The code is as follows:

```
319 }
320
321
322
323 private void updateSecondaryObjectiveCoverage(TestSuiteChromosome t) {
324     double coverage = t.getCoverage() * 100;
325
326     if(coverage > Properties.REVISE_AFTER_PERCENTAGE_OF_COVERAGE) {
327         enableFirstSecondaryCriterion();
328
329         //After this, REVISE_AFTER_PERCENTAGE_OF_COVERAGE is not used anymore.
330         //We transfer the control to ENABLE_SECONDARY_OBJECTIVE_AFTER
331         Properties.REVISE_AFTER_PERCENTAGE_OF_COVERAGE = 0;
332
333         interactionAlreadyEnabled = true;
334
335         double progress = this.progress() * 100.0;
336         updateSecondaryObjectiveInteractivity(progress);
337     }
338 }
```
- Variables View:** Shows the state of variables at the current execution point:

Name	Value
"coverage"	88.05555555555556
"currentIteration"	15
"population"	(id=80)
"population.get(0)"	(id=89)
"minimizedTestsAndRepresentative"	<error(s)_during_the_evaluation>

SUT: ATM class (EvoSuite tutorial)

<https://www.uco.es/SEBASENet/CEC2021>

A proof of concept

Illustrative example

(2) Selection of candidate solutions

26/30 candidate test suites have the same (best) fitness value, but only **3 will be presented to the tester for revision** (*Percentage_to_revise=10%*)

The screenshot shows the Eclipse IDE interface. The main editor displays a Java file named `InteractiveEvaluationSecondaryObjective.java`. The code includes a comment: `// Note: It may happen that the minimization fails. This is something to take into account in case`. A line of code is highlighted: `if(population.size() == 1 || (!sortInteraction && population.size() > 1 && minimizedTestsAndRepresentative.size() > 1) | (sortInteraction && ((breakTie && minimizedTestsAndRepresentative.size() >= 1) || (population.size() > 1 && minimizedTestsAndRepresentative.size() > 1)))`. Below this, there is a comment: `//2) PASS THE INFORMATION TO EXTERNAL FILES FOR THEIR REVISION`. The code continues with a `boolean` variable `fenotypesToValue` set to `false`, a `for` loop over `Map.Entry` objects, and an `if` statement checking for `valuedFenotypes`. The `break` statement is used to exit the loop. The `if` statement is followed by a `Properties` object check: `if(fenotypesToValue || Properties.REVISIT_CANDIDATES) {`.

The right-hand side of the IDE shows the 'Expressions' view. It displays a table of variables and their values:

Name	Value
"coverage"	<error(s)_during_the_evaluation>
"currentIteration"	<error(s)_during_the_evaluation>
"population"	(id=100)
[0]	TestSuiteChromosome (id=101)
[1]	TestSuiteChromosome (id=119)
[2]	TestSuiteChromosome (id=120)
"population.get(0)"	(id=101)
"minimizedTestsAndRepresentative"	(id=99)

A proof of concept

Illustrative example

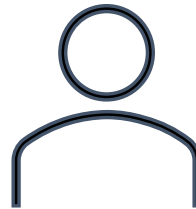
- (3) Minimization
- (4) Tester's evaluation

Only the different test suites after **minimization** are evaluated (2/3)

```
62 public void test4() throws Throwable {
63     Bank bank0 = new Bank();
64     ATM aTM0 = new ATM(bank0);
65     Person person0 = new Person("IouF+u0.!zQ", "%[QR]SqhXNQmvm");
66     int[] intArray0 = new int[3];
67     CurrentAccount currentAccount0 = new CurrentAccount(person0, (-1563));
68     Company company0 = new Company("");
69     bank0.addAccount(currentAccount0);
70     int[] intArray1 = new int[2];
71     ATMCard aTMCard0 = new ATMCard(company0, currentAccount0, intArray1);
72     aTM0.insertCard(aTMCard0);
73     aTM0.enterPin(intArray0);
74 }
75
76 @Test(timeout = 4000)
77 public void test5() throws Throwable {
78     Bank bank0 = new Bank();
79     ATM aTM0 = new ATM(bank0);
80     Company company0 = new Company("");
81     CurrentAccount currentAccount0 = new CurrentAccount(company0, 0);
82     bank0.addAccount(currentAccount0);
83     int[] intArray0 = new int[1];
84     ATMCard aTMCard0 = new ATMCard(company0, currentAccount0, intArray0);
85     aTM0.insertCard(aTMCard0);
86     aTM0.enterPin(intArray0);
87 }
88
89 @Test(timeout = 4000)
90 public void test6() throws Throwable {
91     Bank bank0 = new Bank();
92     ATM aTM0 = new ATM(bank0);
93     aTM0.deposit((-827));
94 }
95
96 @Test(timeout = 4000)
97 public void test7() throws Throwable {
98     Bank bank0 = new Bank();
99     ATM aTM0 = new ATM(bank0);
100     aTM0.withdraw((-827));
101 }
```

7 test cases,
readability=6

Tester



6 test cases,
readability=8

```
49     aTM0.insertCard(aTMCard0);
50 }
51
52 @Test(timeout = 4000)
53 public void test3() throws Throwable {
54     Bank bank0 = new Bank();
55     ATM aTM0 = new ATM(bank0);
56     int[] intArray0 = new int[0];
57     aTM0.enterPin(intArray0);
58 }
59
60 @Test(timeout = 4000)
61 public void test4() throws Throwable {
62     Bank bank0 = new Bank();
63     ATM aTM0 = new ATM(bank0);
64     Company company0 = new Company("");
65     CurrentAccount currentAccount0 = new CurrentAccount(company0, 0);
66     bank0.addAccount(currentAccount0);
67     int[] intArray0 = new int[19];
68     ATMCard aTMCard0 = new ATMCard(company0, currentAccount0, intArray0);
69     aTM0.insertCard(aTMCard0);
70     aTM0.enterPin(intArray0);
71 }
72
73 @Test(timeout = 4000)
74 public void test5() throws Throwable {
75     Bank bank0 = new Bank();
76     ATM aTM0 = new ATM(bank0);
77     aTM0.deposit((-827));
78 }
79
80 @Test(timeout = 4000)
81 public void test6() throws Throwable {
82     Bank bank0 = new Bank();
83     ATM aTM0 = new ATM(bank0);
84     aTM0.withdraw((-827));
85 }
86 }
```

A proof of concept

Illustrative example

(5) Integrate feedback
(6) Search continues...

Readability scores are assigned to the test suites,
and **the population is sorted**

New interactions might be required **in next generations**
(*Revise_frequency=10, Max_times_sort=3*)

Name	Value
"coverage"	<error(s)_during_the_evaluation>
"currentiteration"	16
"population"	(id=237)
[0]	TestSuiteChromosome (id=119)
[1]	TestSuiteChromosome (id=120)
[2]	TestSuiteChromosome (id=120)
[3]	TestSuiteChromosome (id=101)
[4]	TestSuiteChromosome (id=101)
[5]	TestSuiteChromosome (id=242)
[6]	TestSuiteChromosome (id=243)
[7]	TestSuiteChromosome (id=244)
[8]	TestSuiteChromosome (id=245)
[9]	TestSuiteChromosome (id=244)

IT IS TIME TO INTERACT! Go to the folder EvoSuite/CEC/InteractionSort-1.
After that, revise and provide a readability value for each test suite.

Readability for test suite 0:
5
Readability for test suite 1:
7
Readability for test suite 2:
9

TARGET_CLASS: com.examples.with.different.packagename.interactive.ATM
criterion: LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH
Coverage: 0.9498263888888889
Total_Goals: 122
Covered_Goals: 111
TestSuite: 12

Conclusions and future work

- ✓ Complete definition of **interactive options** for the test generation problem
 - ✓ Design of an interactive algorithm to **improve readability** of test suites
 - ✓ A **running example** using EvoSuite with parameters to control the interaction
-

- ❖ More experiments (SUT, parameters), including studies with testers
- ❖ Implementation of interaction options oriented to fault-detection
- ❖ Other test generation problems (GUI, integration testing)

Interactivity in the generation of test cases with evolutionary computation



Aurora Ramírez (aramirez@uco.es)

Pedro Delgado-Pérez (pedro.delgado@uca.es)

Kevin J. Valle-Gómez (kevin.valle@uca.es)

Inmaculada Medina-Bulo (inmaculada.medina@uca.es)

José Raúl Romero (jrromero@uco.es)

