Search-based Testing of Procedural Programs: Iterative Single-Target or Multi-Target Approach?

ssbse 2016

Simone Scalabrino

Giovanni Grano

Dario Di Nucci

Rocco Oliveto

Andrea De Lucia

“

The overall cost of testing has been estimated at being at least half of the entire development cost, if not more.

Boris Beizer

Beizer. Software testing techniques. 2003. Dreamtech Press.

“

Software developers only spend a quarter of their work time engineering tests, whereas they think they test half of their time.

Beller et al.

Beller et al. When, how, and why developers (do not) test in their IDEs. ESEC/FSE 2015

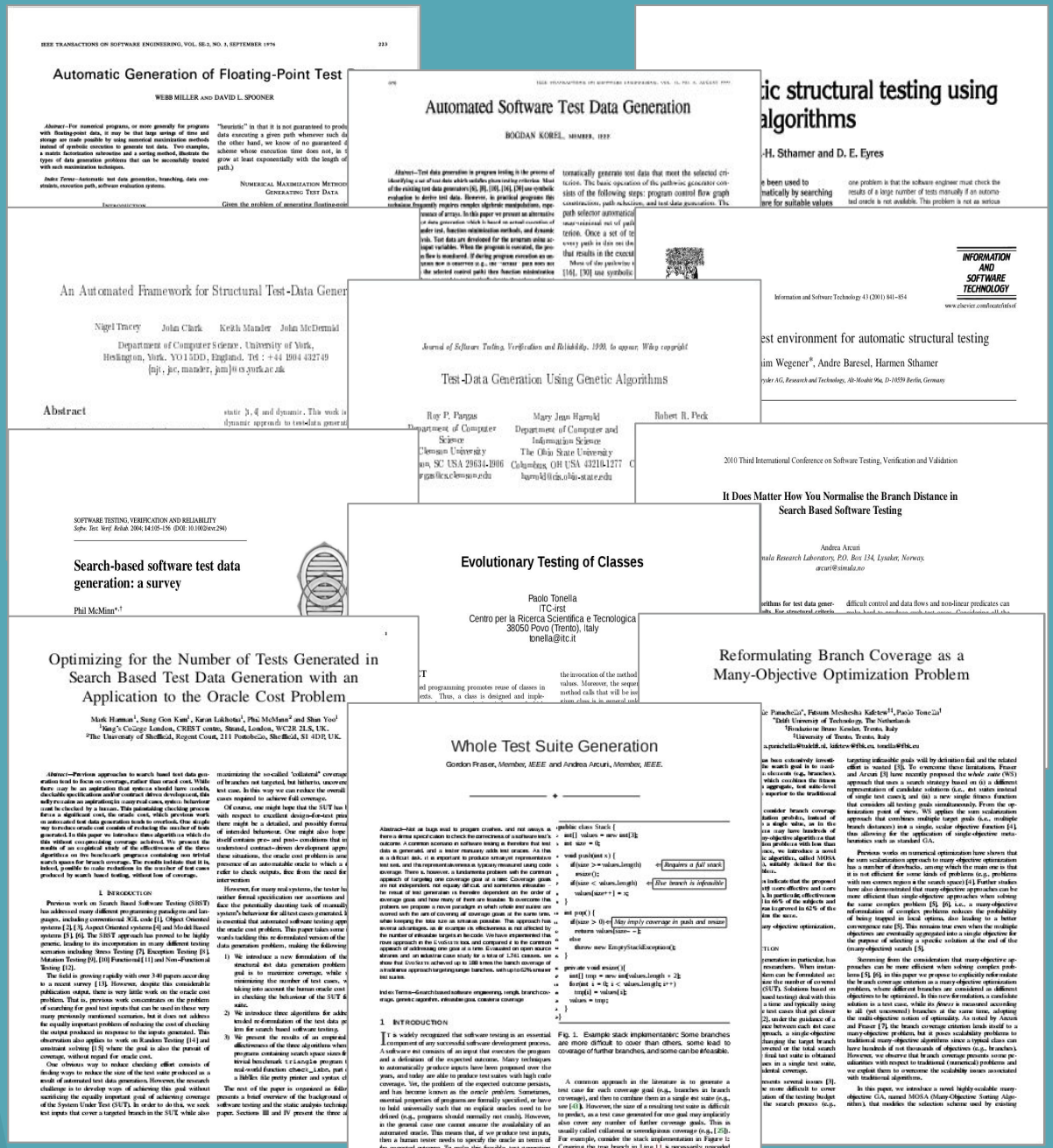
Test case generation

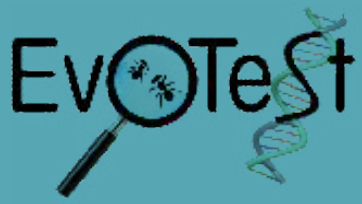
```
700 gdouble
701 gradient_calc_bilinear_factor (gdouble  dist,
702                                gdouble *vec,
703                                gdouble  offset,
704                                gdouble  x,
705                                gdouble  y)
706 {
707   if (dist == 0.0)
708     {
709       return 0.0;
710     }
711   else
712     {
713       gdouble r;
714       gdouble rat;
715
716       /* Calculate linear offset from the start line outward */
717
718       offset = offset / 100.0;
719
720       r   = vec[0] * x + vec[1] * y;
721       rat = r / dist;
722
723       if (fabs (rat) < offset)
724         return 0.0;
725       else if (offset == 1.0)
726         return (rat == 1.0) ? 1.0 : 0.0;
727       else
728         return (fabs (rat) - offset) / (1.0 - offset);
729     }
730 }
```

```
 1 #include <stdlib.h>
 2 #include <check.h>
 3 START_TEST(ocelot_testcase1)
 4 {
 5 double __val0 = 7883.606068766041;
 6 double __val1 = 7472.893282147905;
 7 double __val2 = 8109.302460798321;
 8 double __val3 = -5018.78129027117;
 9
10 double __array0[2] = {-6866.205482964684, -350.3045266881127};
11
12 void* __ptr0 = &__array0;
13
14 gdouble __arg0 = __val0;
15 gdouble __arg2 = __val1;
16 gdouble __arg3 = __val2;
17 gdouble __arg4 = __val3;
18 gradient_calc_bilinear_factor(__arg0,__ptr0,__arg2,__arg3,__arg4);
19
20 /* REPLACE THE ASSERTION BELOW */
21 ck_assert_str_eq("OK", "OK");
22 }
23 END_TEST
24
```

Test case generation

SBST Methodologies





EVOSUITE

AUSTIN

IGUANA

Input Generation Using Automated Novel Algorithms

Test case generation

SBST Tools

eToc - evolutionary Testing of classes

OCELOT

Optimal Coverage sEarch-based tooL for sOftware Testing

OCELOT

Features

- Test Case Generation for C
- Fully Implemented in Java and C (through JNI)
- Based on JMetal Framework
- Structs and Pointers Handling
- Check Unit Testing Framework

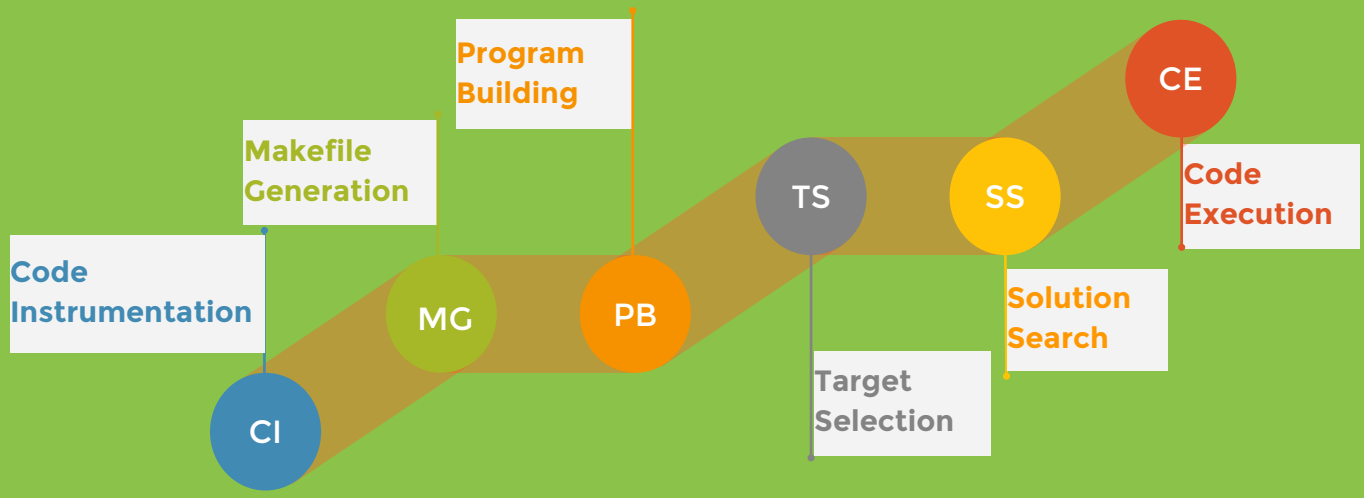
2015	Change	Programming Language	Ratings	Change
		Java	18.236%	-1.33%
		C	10.955%	-4.67%
		C++	6.657%	-0.13%
		C#	5.493%	+0.58%
		Python	4.302%	+0.64%
	↑	JavaScript	2.929%	+0.59%
	↓	PHP	2.847%	+0.32%
	↑	Assembly language	2.417%	+0.61%
	↓	Visual Basic .NET	2.343%	+0.28%
	↓	Perl	2.333%	+0.43%
	↑	Delphi/Object Pascal	2.169%	+0.42%
		Ruby	1.965%	+0.18%
	↑	Swift	1.930%	+0.74%
	⇓	Objective-C	1.849%	+0.03%
	↑	MATLAB	1.826%	+0.65%
		Groovy	1.818%	+1.31%
		Visual Basic	1.761%	+0.23%
		R	1.684%	+0.64%
		Go	1.625%	+1.37%

OCELOT

Why C?

OCELOT

Process Overview



MOSA

Multi-Objective Sorting Algorithm*

Panichella et al.

LIPS

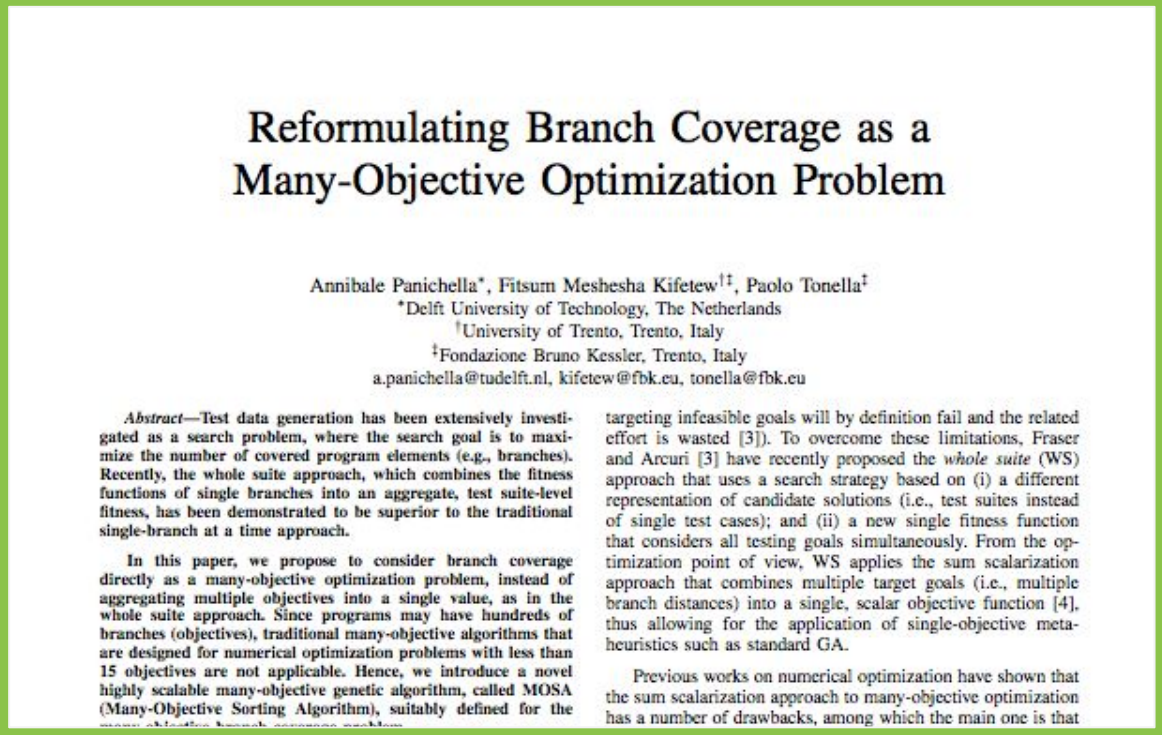
Linearly Independent Path based Search

Scalabrino et al.

OCELOT

Target Selection Algorithms

Many-Objective Sorting Algorithm



Reformulating Branch Coverage as a Many-Objective Optimization Problem

Annibale Panichella*, Fitsum Meshesha Kifetew†‡, Paolo Tonella‡
*Delft University of Technology, The Netherlands
†University of Trento, Trento, Italy
‡Fondazione Bruno Kessler, Trento, Italy
a.panichella@tudelft.nl, kifetew@fbk.eu, tonella@fbk.eu

Abstract—Test data generation has been extensively investigated as a search problem, where the search goal is to maximize the number of covered program elements (e.g., branches). Recently, the whole suite approach, which combines the fitness functions of single branches into an aggregate, test suite-level fitness, has been demonstrated to be superior to the traditional single-branch at a time approach.

In this paper, we propose to consider branch coverage directly as a many-objective optimization problem, instead of aggregating multiple objectives into a single value, as in the whole suite approach. Since programs may have hundreds of branches (objectives), traditional many-objective algorithms that are designed for numerical optimization problems with less than 15 objectives are not applicable. Hence, we introduce a novel highly scalable many-objective genetic algorithm, called MOSA (Many-Objective Sorting Algorithm), suitably defined for the

targeting infeasible goals will by definition fail and the related effort is wasted [3]). To overcome these limitations, Fraser and Arcuri [3] have recently proposed the *whole suite* (WS) approach that uses a search strategy based on (i) a different representation of candidate solutions (i.e., test suites instead of single test cases); and (ii) a new single fitness function that considers all testing goals simultaneously. From the optimization point of view, WS applies the sum scalarization approach that combines multiple target goals (i.e., multiple branch distances) into a single, scalar objective function [4], thus allowing for the application of single-objective meta-heuristics such as standard GA.

Previous works on numerical optimization have shown that the sum scalarization approach to many-objective optimization has a number of drawbacks, among which the main one is that

Fitness function reformulation

Find a set of test cases that optimizes the branch coverage of each branch

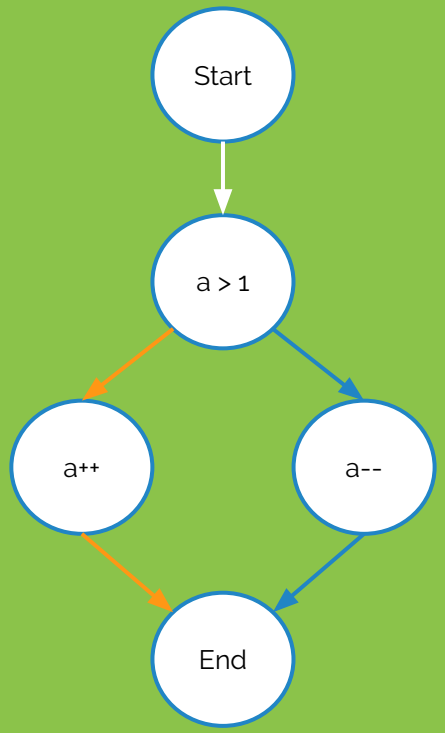
Dominance and Pareto optimality

Each solution is evaluated in terms of Pareto dominance and optimality,

Preference Sorting

A new ranking algorithm for sorting the solutions

Linearly Independent Path based Search



Single target selection algorithm

Inspired by McCabe baseline method and Dynamic Symbolic Execution

- Starts with random test data (t_0)
- Selects a new target at each iteration

Seeding

Final population of the previous iteration is reused

Collateral coverage

Considers the targets serendipitously achieved

Search budget optimization

Allocates SB_i/n_i evaluations for each target

Independent of the search algorithm

Current implementation based on GA

EMPIRICAL EVALUATION

Research Questions

MOSA VS LIPS

RQ1
Effectiveness

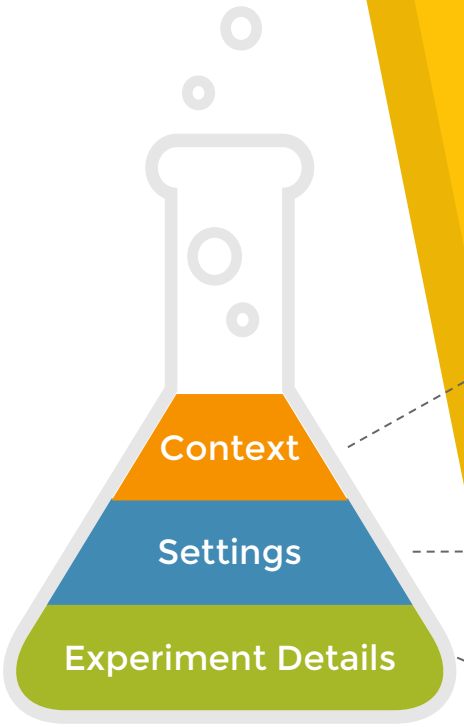
Branch Coverage

RQ2
Efficiency

Execution Time

RQ3
Oracle Coast

Test Suite Size



Design

35 C functions
605 branches

Gimp: Gnu Image Manipulation Program
GSL: Gnu Scientific Library
SGLIB: a generic library for C
spice: analogue circuit simulator

Population Size	*100*
Crossover Rate	*0.90*
Mutation Rate	*1/#variables*
Search Budget	*200.000*

30 runs
Average Values
Wilcoxon's Test (p-value 0.05)
Vargha-Delaney Test

Results RQ1: Effectiveness

	MOSA	LIPS
Overall Branch Coverage	84,73%	86.29%
Cases in which is better	2[1]	10[2]

[1] 1 case with large effect size
[2] 8 cases with medium/large effect size

Results RQ2: Efficiency

	MOSA	LIPS
Average Execution Time	14.80s	5.03s
Cases in which is better	0	35[1]

[1] with large effect size

Results RQ2: Efficiency

	MOSA	LIPS
Average Execution Time	14.80s	5.03s
Cases in which is better	0	35[1]

Too much time for ranking the Pareto Fronts!

[1] with large effect size

Results RQ3: Oracle Cost

	MOSA	LIPS
Average # Test Cases	4.4	6.1
Cases in which is better	35[1]	0

[1] 33 cases with large effect size

Results RQ3: Oracle Cost

	MOSA	LIPS
Average # Test Cases	4.4	6.1
Cases in which is better	35[1]	0

LIPS does not directly handle the oracle problem!

[1] 33 cases with large effect size

Worse than LIPS in terms of branch coverage
Worse than LIPS in terms of execution time[1]

	MOSA	LIPS*
Average # Test Cases	4.4	4.25
Cases in which is better	6	7

LIPS*

no collateral coverage

[1] Although better than MOSA

LIPS + minimization

	MOSA[1]	LIPS
Average # Test Cases	3.61	3.66
Cases in which is better	6[2]	2[3]

[1] For a fair comparison the minimization was applied also to MOSA suites
[2] 4 cases with medium/large effect size
[3] 1 case with large effect size

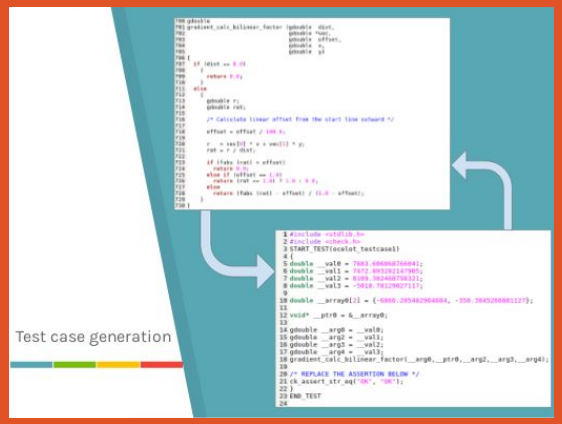
LIPS + minimization

	MOSA[1]	LIPS
Average # Test Cases	3.61	3.66
Cases in which is better	6[2]	2[3]

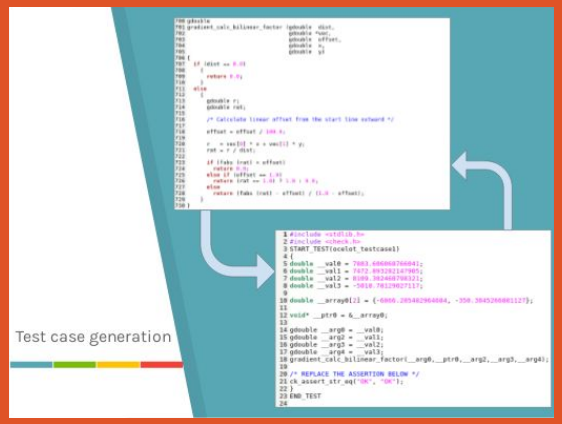
Minimization execution time < 1s

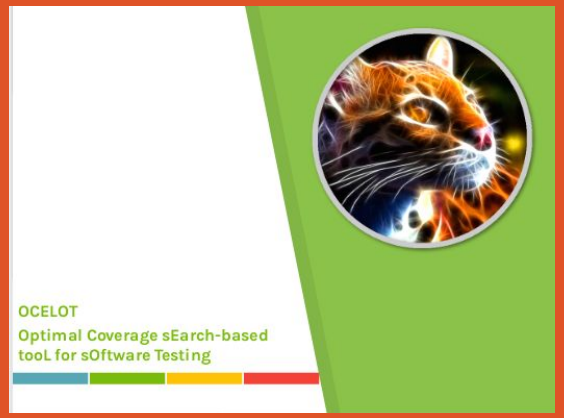
[1] For a fair comparison the minimization was applied also to MOSA suites
[2] 4 cases with medium/large effect size
[3] 1 case with large effect size

Conclusions

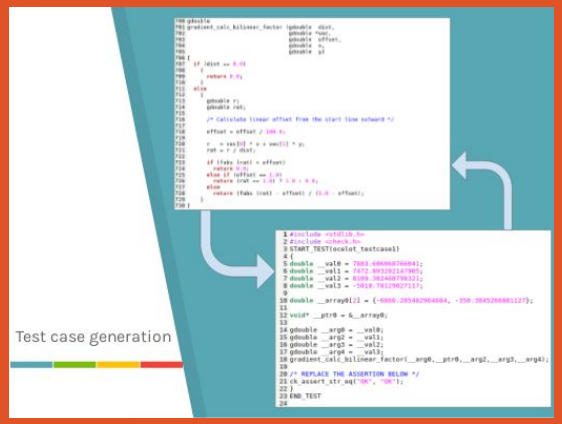


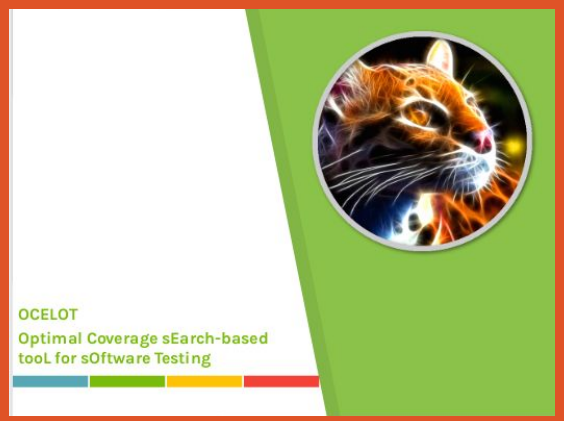
Conclusions

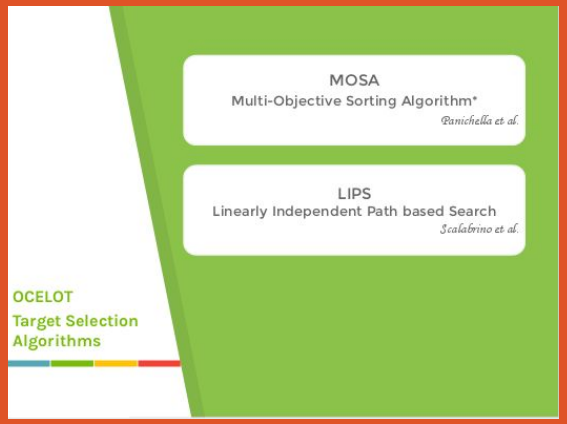




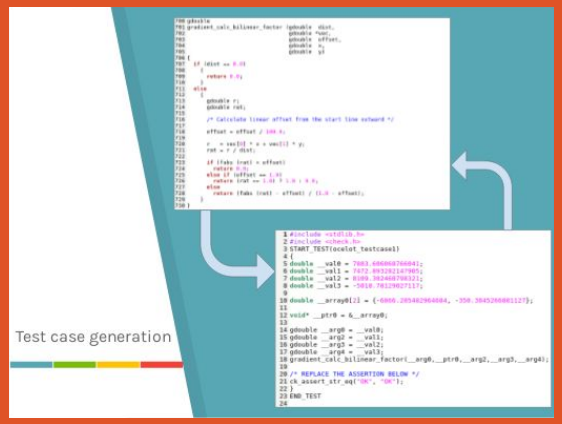
Conclusions

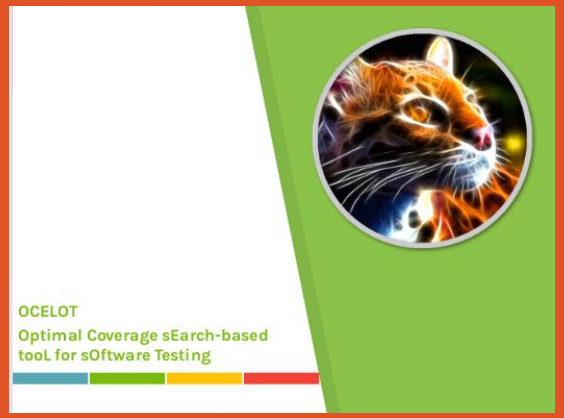


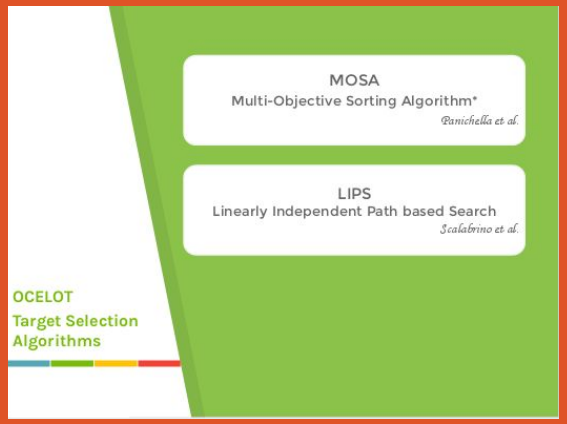




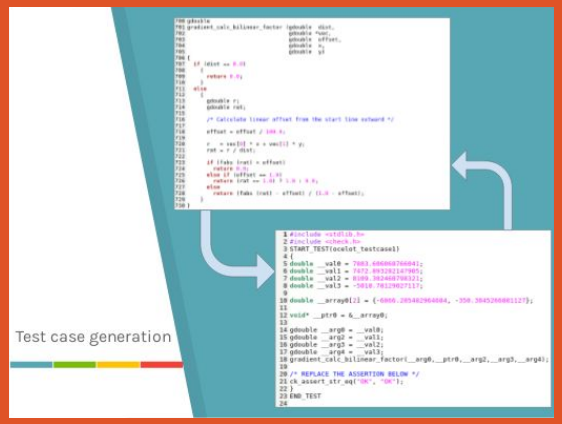
Conclusions

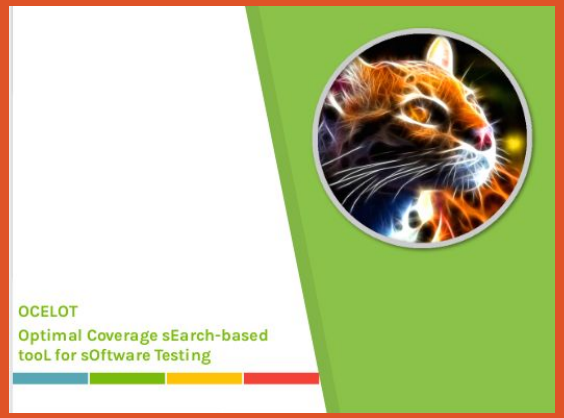


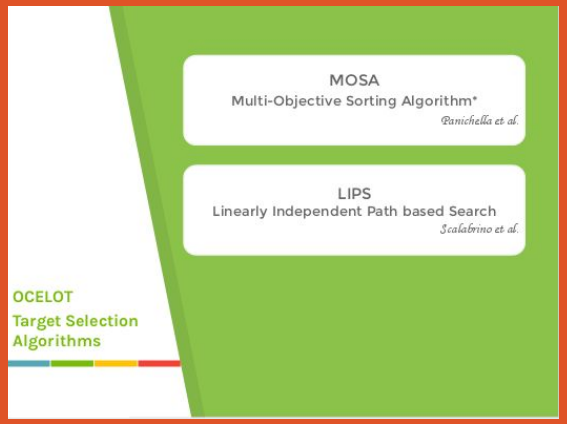




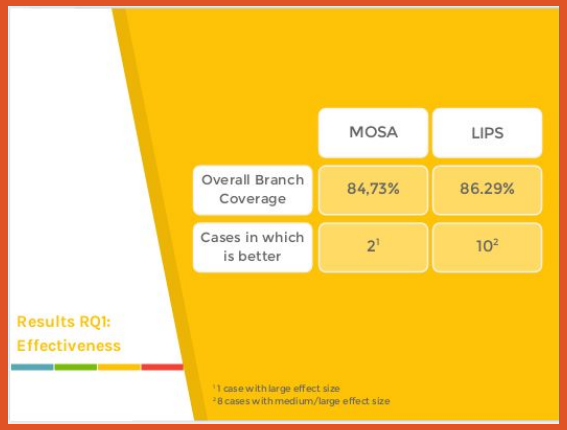
Conclusions

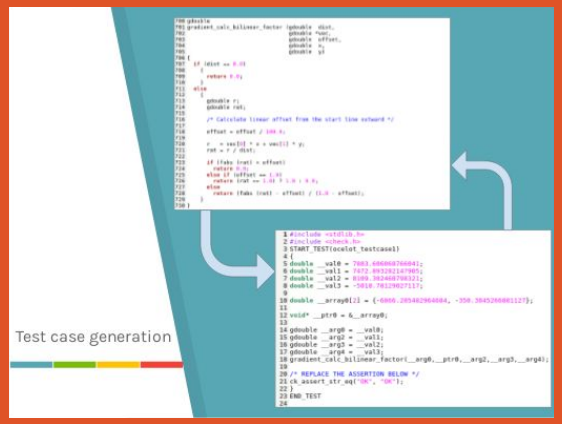


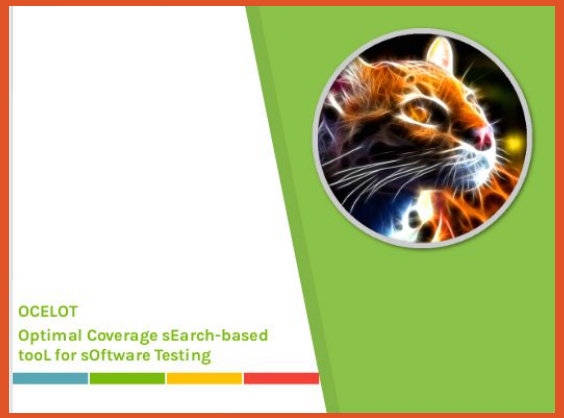


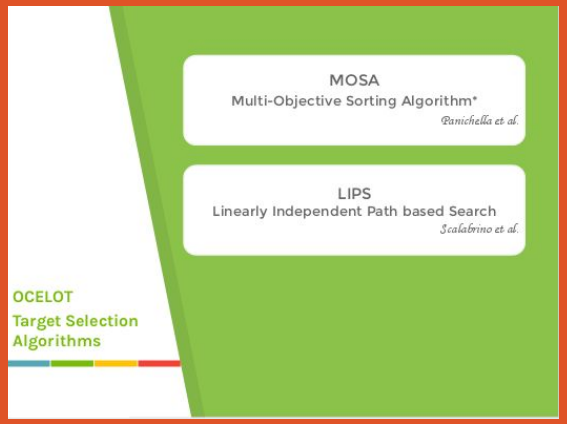


Conclusions

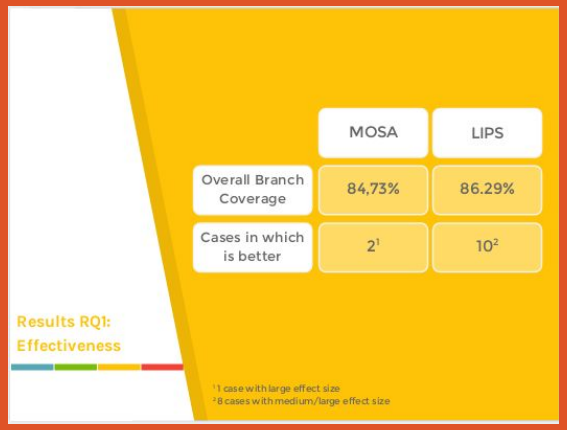








Conclusions



Test case generation

OCELOT
Optimal Coverage sEarch-based tooL for sOftware Testing

OCELOT
Target Selection Algorithms

MOSA
Multi-Objective Sorting Algorithm*
Panichella et al.

LIPS
Linearly Independent Path based Search
Scalabrino et al.

MOSA VS LIPS

Research Questions

- RQ1 Effectiveness — Branch Coverage
- RQ2 Efficiency — Execution Time
- RQ3 Oracle Coast — Test Suite Size

Conclusions

Results RQ1: Effectiveness

	MOSA	LIPS
Overall Branch Coverage	84,73%	86.29%
Cases in which is better	2[1]	10[2]

[1] 1 case with large effect size
[2] 8 cases with medium/large effect size

Results RQ2: Efficiency

	MOSA	LIPS
Average Execution Time	14.80s	5.03s
Cases in which is better	0	35[1]

Too much time for ranking the Pareto Fronts!

[1] with large effect size

Results RQ3: Oracle Cost

	MOSA	LIPS
Average # Test Case	4.4	6.1
Cases in which is better	35[1]	0

LIPS does not directly handle the oracle problem!

[1] 33 cases with large effect size

Test case generation

OCELOT
Optimal Coverage sEarch-based tooL for sOftware Testing

OCELOT
Target Selection Algorithms

MOSA
Multi-Objective Sorting Algorithm*
Panichella et al.

LIPS
Linearly Independent Path based Search
Scalabrino et al.

MOSA VS LIPS

RQ1 Effectiveness — Branch Coverage
RQ2 Efficiency — Execution Time
RQ3 Oracle Coast — Test Suite Size

Research Questions

Conclusions

Results RQ1: Effectiveness

	MOSA	LIPS
Overall Branch Coverage	84,73%	86.29%
Cases in which is better	2[1]	10[2]

[1] 1 case with large effect size
[2] 8 cases with medium/large effect size

Results RQ2: Efficiency

	MOSA	LIPS
Average Execution Time	14.80s	5.03s
Cases in which is better	0	35[1]

Too much time for ranking the Pareto Fronts!

[1] with large effect size

Results RQ3: Oracle Cost

	MOSA	LIPS
Average # Test Case	4.4	6.1
Cases in which is better	35[1]	0

LIPS does not directly handle the oracle problem!

[1] 33 cases with large effect size

Worse than LIPS in terms of branch coverage
Worse than LIPS in terms of execution time[1]

LIPS*
no collateral coverage

	MOSA	LIPS*
Average # Test Case	4.4	4.25
Cases in which is better	6	7

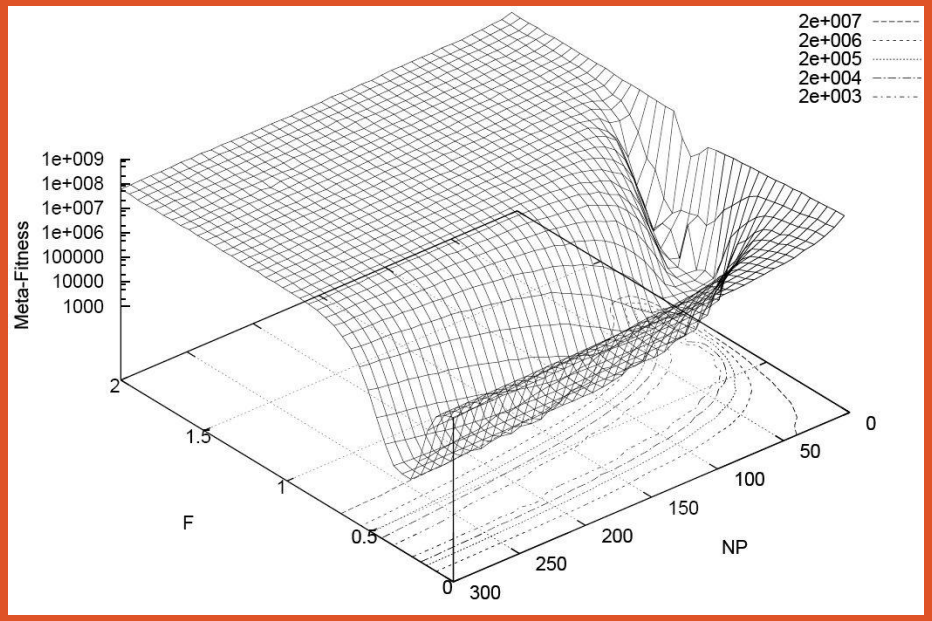
[1] Although better than MOSA

Future works

Future works

Replicate on larger dataset

Future works



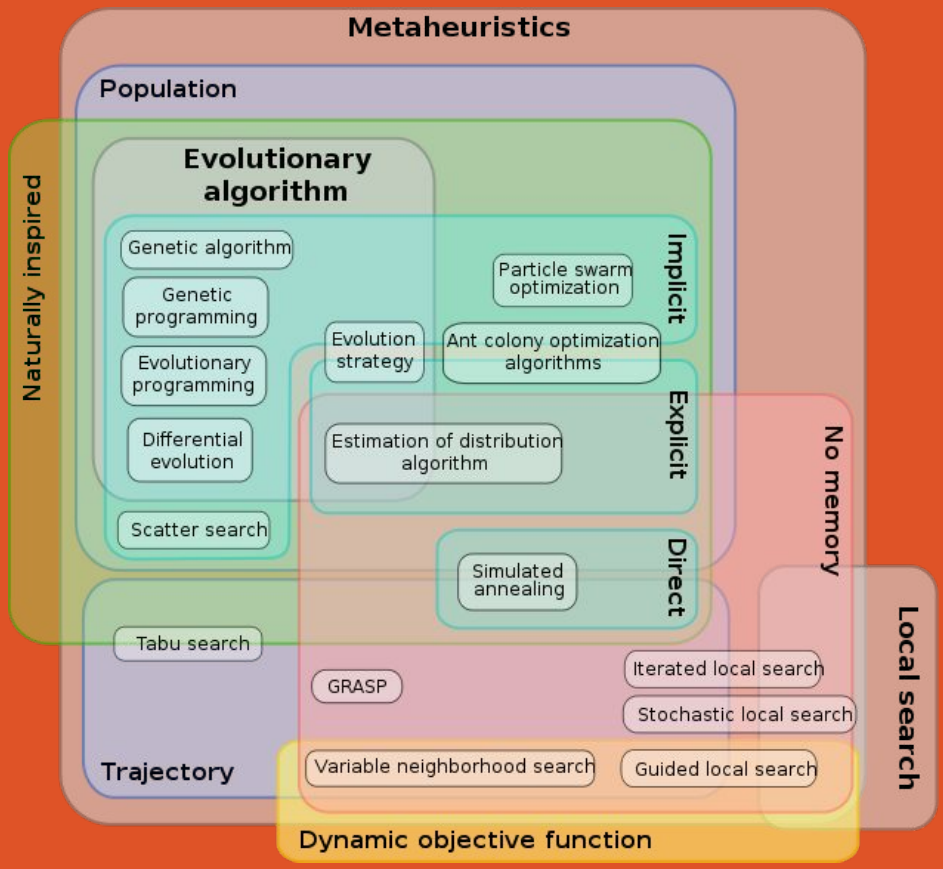
Replicate on larger dataset

Study C landscapes

Future works

- Replicate on larger dataset
- Study C landscapes
- Add LIPS to Evosuite

Future works



Replicate on larger dataset

Study C landscapes

Add LIPS to Evosuite

Go beyond Genetic Algorithm

Thanks for your attention!

Questions?

Dario Di Nucci

University of Salerno

ddinucci@unisa.it

http://www.sesa.unisa.it/people/ddinucci/