Improving Assertion Oracles with Evolutionary Computation

Valerio Terragni
University of Auckland
Auckland, New Zealand
v.terragni@auckland.ac.nz

Gunel Jahangirova
Università della Svizzera italiana
Lugano, Switzerland
gunel.jahangirova@usi.ch

Mauro Pezzè
Università della Svizzera italiana
Lugano, Switzerland
Schaffhausen Institute of Technology
Schaffhausen, Switzerland
mauro.pezze@usi.ch

Paolo Tonella
Università della Svizzera italiana
Lugano, Switzerland
paolo.tonella@usi.ch

ABSTRACT

Assertion oracles are executable boolean expressions placed inside a software program that verify the correctness of test executions. A perfect assertion oracle passes (returns true) for all correct executions and fails (returns false) for all incorrect executions. Because designing perfect assertion oracles is difficult, assertions often fail to distinguish between correct and incorrect executions. In other words, they are prone to false positives and false negatives.

GAssert is the first technique to automatically improve assertion oracles by reducing false positives and false negatives. Given an assertion oracle and a set of correct and incorrect program states, GAssert employs a novel co-evolutionary algorithm that explores the space of possible assertions to identify one with fewer false positives and false negatives. Our evaluation on 34 Java methods shows that GAssert effectively improves assertion oracles.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; • **Theory of computation** → **Assertions**; **Bio-inspired optimization**; **Evolutionary algorithms**.

KEYWORDS

Evolutionary Computation, Program Assertions, Test Generation, Mutation Analysis, Co-evolutionary Algorithms

ACM Reference Format:
Valerio Terragni, Gunel Jahangirova, Mauro Pezzè, and Paolo Tonella. 2021. Improving Assertion Oracles with Evolutionary Computation. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3449726.3462722

```
public static int min(int x, int y) {
1  int min;
2  if (x <= y) {
3      min = x;
4  } else {
5      min = y; // mutant M1 : min = y + 1;
6  }
7  assert(min < x); // assertion oracle
8  return min;
9 }
```

Figure 1: Example of assertion oracle with FPs and FNs.

1 INTRODUCTION

Software testing aims to verify the correctness of a piece of software by executing test cases. The challenge of distinguishing correct from incorrect test executions is called the *(test) oracle problem*, and it is recognised as one of the fundamental challenges in software testing. A popular form of oracles are executable boolean expressions (called assertion oracles) that predicate on the values of program variables at specific program points. If the expression evaluates to true the test execution is deemed to be correct, incorrect otherwise.

Assertion oracles often fail to distinguish between correct and incorrect executions [2], that is, they are prone to both false positives and false negatives. A **false positive (FP)** is a correct program state in which the assertion fails (but should pass). A **false negative (FN)** is an incorrect program state in which the assertion passes (but should fail). We obtain correct and incorrect program states by executing test cases on instrumented versions of the original and faulty versions of the method under test, respectively. Faulty versions are obtained by seeding artificial faults (called mutants).

Figure 1 shows a Java method that returns the minimum between two integers x and y. The comment at Line 5 shows a seeded fault (mutant) M_1 used to produce incorrect program states. Line 7 shows an assertion oracle ($min < x$) that suffers from both FPs and FNs. An example of FP is the program state `{x=3, y=5, min=3}` induced by the test case `min(x=3, y=5)`. This is a correct program state but the assertion returns false. An example of FN is the program state `{x=9, y=7, min=8}` induced by the test case `min(x=9, y=7)` and the program mutation M_1. This is an incorrect program state but the assertion returns true. Our goal is to automatically improve assertion oracles by minimizing their FPs and FNs.

2 GASSERT

We recently proposed **GAssert** [4, 5], the first technique to automatically improve assertion oracles. Given an assertion oracle and a set of correct and incorrect program states, GAssert explores the space of possible assertions with a co-evolutionary algorithm to search for an improved assertion with zero FPs and the lowest number of FNs. GAssert favors assertions with zero false positives, as false alarms are known to trigger an expensive debugging process. For the example in Figure 1, GAssert returns an improved assertion $((min == x)$ OR $(min == y))$ AND $((min \leq x)$ AND $(min \leq y))$ that intuitively captures the expected behavior of a "min" function.

GAssert explores the huge space of candidate assertions with a **co-evolutionary algorithm** that formulates the oracle improvement problem as a multi-objective optimization problem (MOOP) with three competing objectives: (i) minimizing the number of FPs, (ii) minimizing the number of FNs, (iii) minimizing the size of the assertion (to improve the readability of assertions).

Classic multi-objective evolutionary approaches (e.g., NSGA-II) often rely on *Pareto optimality* to produce solutions that offer the best trade-off between competing objectives. However, in our case not all assertions with an optimal trade-off between FPs and FNs are acceptable solutions. Indeed, GAssert aims to obtain assertions with zero FPs and the lowest number of FNs. Alternatively, primarily focusing on reducing FPs may also be inadequate, as there may not be enough *evolution pressure* to also reduce the FNs.

To address the challenge, GAssert proposes a **co-evolutionary approach** that evolves in parallel two distinct populations of assertions ($\mathcal{P}_{FP}$ and $\mathcal{P}_{FN}$) with two competing fitness functions that reward less false positives and less false negatives, respectively. Both fitness functions consider the remaining objectives only in tie cases. These populations periodically migrate their best individuals to exchange genetic material useful to improve the secondary objectives. Eventually, $\mathcal{P}_{FP}$ will be more likely to produce assertions with zero FPs and fewer FNs. In fact, migrating the best individuals of $\mathcal{P}_{FN}$ adds in $\mathcal{P}_{FP}$ assertions with a decreasing number of FNs.

GAssert includes canonical tree-based evolutionary operators. It also implements novel operators specifically designed for the oracle improvement problem. For instance, GAssert proposes the *best-match selection criterion* that exploits semantic information about the correct and incorrect states that each assertion *covers*. It selects the first parent randomly and the second one using *weighted random selection*, where assertions with a higher *weight* are more likely to be selected. The weight is defined as the number of correct (or incorrect) states in which the second parent correctly returns true (or false), while the first parent does not. Intuitively, the criterion increases the chances of crossover between two complementary individuals that are likely to yield a fitter offspring.

3 EVALUATION RESULTS

We evaluated GAssert on 34 methods from 7 Java code bases [4]. We evaluated the ability of GAssert to improve an initial set of assertion oracles generated by Daikon [1], the most popular invariant generator for Java. For each method, we ran GAssert ten times with a time budget of 90 minutes. We compared GAssert with a Random variant of GAssert where the guidance provided by our fitness functions is replaced by a random choice.

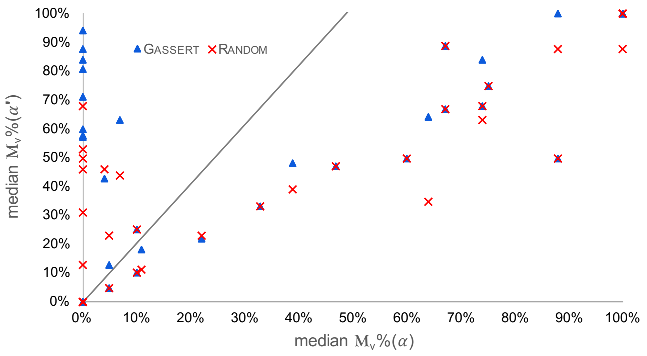

Figure 2: Mutation score (i.e., percentage of detected seeded faults) improvement of the 34 methods.

When executed with unseen tests and mutants, the GAssert-improved assertions (α') increase the mutation score (i.e., percentage of detected seeded faults) by 34% (on average) with respect to the mutation score obtained with the initial assertions (α).

Figure 2 plots the median mutation score for each pair of initial and improved assertions w.r.t. GAssert and Random. If a point is on the diagonal it means that the corresponding approach did not improve the mutation score wrt the initial assertion. Most of GAssert points are above the diagonal, which means that GAssert produced improved assertion with a higher mutation score.

We also compared GAssert with a set of human-improved assertions collected from 102 developers [3] obtainingu comparable results.. Remarkably, 10% of the human-improved assertions achieve a lower mutation score than the assertions improved by GAssert.

4 CONCLUSION

Automatically improving assertion oracles is crucial to increase the fault-detection capabilities of test cases. The evolutionary algorithm of GAssert is effective at exploring the search space of possible assertion oracles. Moreover, the evolutionary operators of GAssert defined specifically for the oracle improvement process show promising results that are worth exploring further.

ACKNOWLEDGEMENTS

This work was partially supported by the Swiss SNF project *ASTERIx* (SNF 200021_178742) and the H2020 project *PRECRIME*, funded under the ERC Advanced Grant 2017 Program (ERC Grant Agreement n. 787703).

REFERENCES

[1] Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. 1999. Dynamically Discovering Likely Program Invariants to Support Program Evolution. In *Proceedings of the International Conference on Software Engineering*. 213–224.

[2] Gunel Jahangirova, David Clark, Mark Harman, and Paolo Tonella. 2016. Test Oracle Assessment and Improvement. In *Proceedings of the International Symposium on Software Testing and Analysis*. 247–258.

[3] Gunel Jahangirova, David Clark, Mark Harman, and Paolo Tonella. 2019. An Empirical Validation of Oracle Improvement. *IEEE TSE* (2019).

[4] Valerio Terragni, Gunel Jahangirova, Paolo Tonella, and Mauro Pezzè. 2020. Evolutionary improvement of assertion oracles. In *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1178–1189.

[5] Valerio Terragni, Gunel Jahangirova, Paolo Tonella, and Mauro Pezzè. 2021. GAssert: A Fully Automated Tool to Improve Assertion Oracles. In *Proceedings of the International Conference on Software Engineering (Companion Volumne)*.