Evolutionary Computation Algorithms in Automatic Test Cases Generation

Liam McDevitt
lm15ue@brocku.ca

Department of Computer Science
Brock University
COSC 5P07 Presentation

March 24, 2021

Overview

Introduction

Evolutionary Computation

Automatic Test Case/Data Generation

Ant Colony Optimization

- Software Test Data Generation Using ACO

Particle Swarm Optimization

- Test Cases Generation Based on PSO & MRs

Genetic Algorithms

- Automatic Test Case Generation Based on GA & MA

Conclusions & Future Work

Introduction

- One of the most important, timely, and costly tasks is testing [1], [2].
- Testing usually comes with given inputs and an expected output.
- The ultimate goal in software testing is to ensure the software meets the clients requirements.
- Software quality arises from comprehensive testing.
- To ensure wide range and extensive tests of the software, test cases are often created.
- Creating these test cases often takes time, money and are often prone to errors themselves.
- Having an automatic intelligent way to generate a test suite full of cases would be a valuable asset to assure software reliability and confidence.
- This is how automatic test case generation was born.

Introduction (cont.)

- Evolutionary Computation (EC) algorithms have been used to generate profitable test cases for software in the past:
  - Genetic Algorithms (GA) [1]
  - Bee Colony Algorithm (BCA) [3]
  - Harmony Search (HS) [4]
  - Cuckoo Search (CS) [3]
  - Firefly Algorithm (FA) [4]
  - Ant Colony Optimization (ACO) [5]
  - Particle Swarm Optimization (PSO) [2]

Evolutionary Computation

- Evolutionary Computation (EC) is a group of algorithms inspired by nature to help find favourable solutions to global optimization problems through the fundamental idea of biological evolution [6].
- The overarching technique is to evolve a population, where each member of the population represents a candidate solution [6].
- These candidate solutions are evaluated by a fitness function [6].
- Over a series of generations a population is contingent on various forms of stochastic altercations to the candidates, i.e. selection and mutation [6].
- As the population moves from one generation to the next our goal is to optimize our designed fitness function [6].

Automatic Test Case/Data Generation

- The ordinary underlying notion for test data generation using Evolutionary Computation is that we have a search space resembling a test suite and our fitness function is known as the, “test adequacy criterion” [6].
- In essence, the “test adequacy criterion” is our testing goal, i.e. what we’re trying to optimize [6].
- The goal is that our test suite from our generated inputs meets our goal as closely as possible [6].
- Our fitness function is our human input for defining how well we’re meeting our goal numerically [6].

Automatic Test Case/Data Generation (cont.)

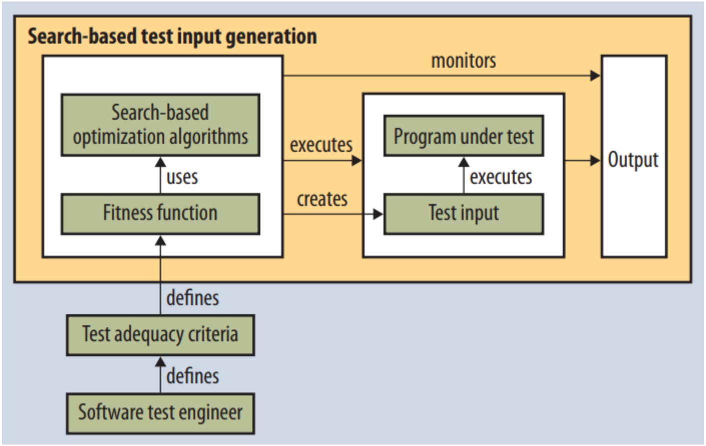

Figure 1: A standard Evolutionary Computation test generation blueprint. Taken from *Software Engineering Meets Evolutionary Computation* by Mark Harman [6].

Ant Colony Optimization

- Ant Colony Optimization (ACO) is a stochastic optimization strategy mainly used for finding favourable paths through graphs [7].
- Influenced by the real-world behaviour of ants due to their incredible teamwork abilities [7].
- Ants are able to find optimal paths through graphs by simulating their pheromone-based link [7].
- As ants explore the search space they keep track of their path and the quality of the path to communicate it with the rest of the ants in the hopes that further ants find better paths over time [7].

Figure 2: Ants working together towards a common goal.

Software Test Data Generation Using ACO

In 2005, a paper written by Li and Lam called, "*Software Test Data Generation Using Ant Colony Optimization*" introduced an Ant Colony Optimization (ACO) algorithm paired with UML Statechart diagrams for generating test data to be used for testing program execution [5].

- Converts a UML Statechart diagram representing the system we're testing into a directed graph.
- The ants explore the crated graph with the goal of generating test data to meet a test coverage adequacy criterion.
- Overall, the presented algorithm was able to achieve solutions to the problem covering all states and each path was feasible.
- However, they were not able to generate a test suite containing non-redundant test cases, i.e. have the smallest possible number of them.

Particle Swarm Optimization

- PSO is an non-deterministic, stochastic, population-based optimization algorithm based on the social behaviour of animals in nature introduced by Kennedy and Eberhart in 1995 [8].
- Communication between these social entities drives them towards a common goal [8].
- PSO was originally based on the social flocking behaviour of birds [8].

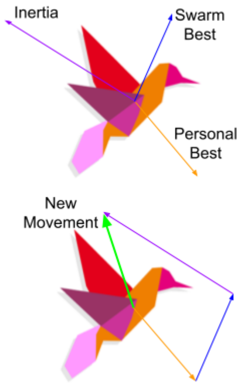

Figure 3: PSO particle movement.

Test Cases Generation Based on PSO & MRs

In 2018 a paper was written by Lv et al., called *Test cases generation for multiple paths based on PSO algorithm with metamorphic relations* to improve the efficiency of test case generation [2].

- Combines PSO with Metamorphic Relations (MRs) to ease the necessity of an Oracle question.
- A MR is the relation between inputs and outputs.
- The initial test suite is generated by the PSO algorithm which is utilized by MRs to develop new test cases.
- Since test cases are not solely being generated by the PSO algorithm the MRs are able to generate new test cases more efficiently.

Genetic Algorithms

- A Genetic Algorithm is an Evolutionary Algorithm (EA) that is used to find favorable solutions to difficult optimization problems [9].
- GAs evolve a population of candidate solutions (individuals) over a set number of generations where each candidate solution is a possible solution to our problem [9].
- A fitness function is designed to numerically indicate how well a candidate solution performs [9].

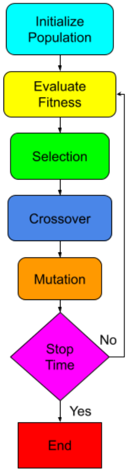

Figure 4: Genetic Algorithm model.

Automatic Test Case Generation Based on GA & MA

Automatic Test Case Generation based on Genetic Algorithm and Mutation Analysis was written by Haga and Suehiro in 2012 and it proposed a way of generating software test cases by combining genetic algorithms and mutation analysis [1].

- Initially, test cases are generated randomly and over time the test cases become polished by a GA.
- The sufficiency of the test suite is measured by mutation scores, which in software testing come from mutation analysis (MA).
- A MA adds intentional errors into the program and the mutation score is determined by how many of these errors the test suite was able to detect.
- If the mutation score does not meet a certain criteria, test cases need to be added and/or modified to progress the score.
- The writers of the paper claimed their proposed method obtained 100% boundary and branch coverage.

Conclusions & Future Work

- There are numerous Evolutionary Computation (EC) strategies for improving software testing in the field of Software Engineering (SE).
- Many of these strategies help to reduce the amount of human intervention in testing and save a great deal in development costs.
- These strategies help us to find good possible solutions to testing software.
- Since it is almost impossible to test every piece of software EC gives us a way of testing the most important aspects.
- In conclusion, EC strategies and their application to software testing shows promise and has been proposed in many areas of software testing.
- Future work involves finding ways to represent software testing problems as search-based problems for Evolutionary Computation (EC) strategies.

Bibliography

[1] H. Haga and A. Suehiro, “Automatic test case generation based on genetic algorithm and mutation analysis,” en, in *2012 IEEE International Conference on Control System, Computing and Engineering*, Penang, Malaysia: IEEE, Nov. 2012, pp. 119–123, ISBN: 978-1-4673-3143-2 978-1-4673-3142-5 978-1-4673-3141-8. DOI: 10.1109/ICCSCE.2012.6487127. [Online]. Available: http://ieeexplore.ieee.org/document/6487127/ (visited on 03/15/2021).

[2] X.-W. Lv, S. Huang, Z.-W. Hui, and H.-J. Ji, “Test cases generation for multiple paths based on PSO algorithm with metamorphic relations,” en, *IET Software*, vol. 12, no. 4, pp. 306–317, Aug. 2018, ISSN: 1751-8806, 1751-8814. DOI: 10.1049/iet-sen.2017.0260. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1049/iet-sen.2017.0260 (visited on 03/15/2021).

Bibliography (cont.)

[3] P. Lakshminarayana and T. V. SureshKumar, "Automatic Generation and Optimization of Test case using Hybrid Cuckoo Search and Bee Colony Algorithm," en, *Journal of Intelligent Systems*, vol. 30, no. 1, pp. 59–72, Jul. 2020, ISSN: 2191-026X, 0334-1860. DOI: 10.1515/jisys-2019-0051. [Online]. Available: https://www.degruyter.com/document/doi/10.1515/jisys-2019-0051/html (visited on 03/15/2021).

[4] R. K. Sahoo, D. Ojha, D. P. Mohapatra, and M. R. Patra, "Automated Test Case Generation and Optimization : A Comparative Review," en, *International Journal of Computer Science and Information Technology*, vol. 8, no. 5, pp. 19–32, Oct. 2016, ISSN: 09754660, 09753826. DOI: 10.5121/ijcsit.2016.8502. [Online]. Available: http://aircconline.com/ijcsit/V8N5/8516ijcsit02.pdf (visited on 03/15/2021).

[5] H. Li and C. P. Lam, "Software Test Data Generation using Ant Colony Optimization," en, vol. 1, p. 5, 2005.

Bibliography (cont.)

[6] M. Harman, “Software Engineering Meets Evolutionary Computation,” en, *Computer*, vol. 44, no. 10, pp. 31–39, Oct. 2011, ISSN: 0018-9162. DOI: 10.1109/MC.2011.263. [Online]. Available: http://ieeexplore.ieee.org/document/6036090/ (visited on 03/15/2021).

[7] M. Dorigo, V. Maniezzo, and A. Colorni, “Positive feedback as a search strategy,” en, p. 23, Jun. 1991.

[8] J. Kennedy and R. Eberhart, “Particle swarm optimization,” en, in *Proceedings of ICNN’95 - International Conference on Neural Networks*, vol. 4, Perth, WA, Australia: IEEE, 1995, pp. 1942–1948, ISBN: 978-0-7803-2768-9. DOI: 10.1109/ICNN.1995.488968. [Online]. Available: http://ieeexplore.ieee.org/document/488968/ (visited on 03/15/2021).

[9] M. Melanie, “An Introduction to Genetic Algorithms,” en, p. 162, 1996.