Future Computing and Informatics Journal

Volume 3
Issue 2 (2018) *Vol 3,Issue 2,2018*

Article 5

2018

An optimization approach for automated unit test generation tools using multi-objective evolutionary algorithms

Samar Ali Abdallah
Arab Academy for Science, Technology & Maritime Transport, Cairo, Egypt, samarel-agouz@student.aast.edu

ramdan mowad
Future University in Egypt, Cairo, Egypt, ramdan.mowad@fue.edu.eg

Esaam Eldeen Fawzy
Arab Academy for Science, Technology & Maritime Transport, Cairo, Egypt, essam.elfakharany@aast.ed

Available online at www.sciencedirect.com

ScienceDirect

Future Computing and Informatics Journal 3 (2018) 178–190
http://www.journals.elsevier.com/future-computing-and-informatics-journal/

An optimization approach for automated unit test generation tools using multi-objective evolutionary algorithms

Samar Ali Abdallah [a,*], Ramadan Moawad [b], Esaam Eldeen Fawzy [a]

[a] *Arab Academy for Science, Technology & Maritime Transport, Cairo, Egypt*
[b] *Future University in Egypt, Cairo, Egypt*

Received 16 January 2018; revised 7 February 2018; accepted 19 February 2018
Available online 30 May 2018

Abstract

High code coverage is measured by the process of software testing typically using automatic test case generation tools. This standard approach is usually used for unit testing to improve software reliability. Most automated test case generation tools focused just on code coverage without considering its cost and redundancy between generated test cases. To obtain optimized high code coverage and to ensure minimum cost and redundancy a Multi-Objectives Evolutionary Algorithm approach (MOEA) is set in motion. An efficient approach is proposed and applied to different algorithms from MOEA Frame from the separate library with three fitness functions for Coverage, Cost, and Redundancy. Four MEOA algorithms have been proven reliable to reach above the 90 percent code coverage: NSGAII, Random, SMSEMOA,v and ε-MOEA. These four algorithms are the key factors behind the MOEA approach.

Keywords: Unit test; Automated test case generation; MOEA; Code coverage

1. Introduction

"Software testing is used as a procedure to ensure that the programs are free from bugs and to improve their function performance. In other words, it is used to determine and improve the quality of the software". Before starting the testing process, it is necessary to choose a test adequacy criterion to evaluate whether a suite is adequate concerning the test goals [1]. A test adequacy criterion defines properties that must be observed by the test suite (e.g., code coverage, functional requirements' coverage, among others).

Code coverage [9] is considered a way to guarantee that tests are testing the code. When testers run those tests, they are presumably checking that the expected results obtained. Coverage measurement is useful in many ways; it improves the testing process, gives the information to the user about the status of the verification process, and helps to find areas that are not covered. Several tools are ranging from manual and automated generation test cases to facilitate the software testing process. The future lies in the automated approach to generate test cases; however, it is prone to errors and is time-consuming. This entails a rise in the cost of the testing process.

We introduce, in our research paper, a new approach to optimize the test cases generation process to decrease the cost of running the unit tests. This process eliminates redundancies; thus, it increases efficiency. As a result, code coverage is maximized. We have incorporated four MEOA algorithms.

MOEA is a significant topic that requires careful heed when addressing real-world problems. Most real-world MOPs (Multi-Objective Problems) have constraints that need to be

* Corresponding author.
E-mail addresses: samarel-agouz@student.aast.edu (S.A. Abdallah), ramdan.mowad@fue.edu.eg (R. Moawad), essam.elfakharany@aast.ed (E.E. Fawzy).

Peer review under responsibility of Faculty of Computers and Information Technology, Future University in Egypt.

https://doi.org/10.1016/j.fcij.2018.02.004

incorporated into our search engine to avoid convergence towards infeasible solutions. Constraints can be "hard" (i.e., they must be satisfied) or "soft" (i.e., they can be relaxed), and addressing them properly has been a matter of research within single-objective EAs [2,3].

MOEA is particularly effective at finding solutions to elusive problems while another MOEA is effective at solving non-uniform Pareto fronts. If one does not know the phenotype space of the problem, using multiple MOEAs with distinctive strengths may prove better than using just one. The MOEAs could then coevolve in such a way that individuals pass from one algorithm to another in an attempt to seed the algorithms with good solutions [2].

In our research paper, we select four different algorithms: NSGAII, Random, SMSEMOA, and ε-MOEA from different libraries with different behavior to provide efficient solutions.

The paper is organized as follows. The next section, some of the related work to our research. In Section 3, we propose the approach to select test cases for automated test cases Generation tools. In Section 4, the different used datasets are presented. In section 5 the brief different algorithms are used in our research. Finally, the experiments and results are discussed in section 6.

2. Related work

Luciano S. de Souza, Pericles B. C. de Miranda, Ricardo B. C. Prudencio and Flavia de A. Barros presented in Ref. [3] a method that uses Particle Swarm Optimization (PSO) to solve multi-objective Test Case selection problems. In contrast to single-objective difficulties, Multi-Objective Optimization (MOO) optimize more than one objective at the same time. They developed a mechanism for functional Test Case selection which considers two objectives simultaneously: maximize code coverage while minimizing cost in of Test Case execution effort based on Particle Swarm Optimization (PSO). They implement two multi-objective versions of PSO (BMOPSO and BMOPSO-CDR). During this work developed a Binary Multi-Objective PSO (BMOPSO), as the solutions of the TC selection problem which represented as binary vectors; and (2) the MOPSO algorithm to work with multi-objective problems. Also, implemented the BMOPSO-CDR algorithm by adding the Crowding Distance and Roulette Wheel mechanism to the BMOPSO to select functional tests.

Luciano S. de Souza, Pericles B. C. de Miranda, Ricardo B. C. Prudencio and Flavia de A. Barros presented enhancement to previous work in Ref. [4] added the so-called catfish effect into the multi-objective selection algorithms to enhance their results. The use of the so-called "catfish" effect, for instance, has shown to improve the performance of the single objective version of the binary PSO. The catfish effect derives its name from an effect that Norwegian fishermen observed when they introduced catfish into a holding tank for caught sardines. The introduction of a catfish, which is different from sardines, into the tank resulted in the stimulation of sardine movement, so keeping the sardines alive and therefore fresh for a longer time. Similarly, the catfish effect was developed to the PSO algorithm in such way that we introduce "catfish particles" to stimulate a renewed search by the rest of the swarm's particles. Hence, these catfish particles help to guide the particles, which can be trapped in a local optimum, to new regions of the search space and thus leading to possible better areas.

Chandraprakash Patidar presented in Ref. [5] approach for test case generation algorithms using Sequence Diagram Based Testing with discrete particle swarm optimization algorithm. The approach presented automated test case generation from UML sequence diagram using discrete Particle Swarm Optimization Algorithm. In his approach introduced an algorithm that automatically creates a dependency then it generated a dependency graph from which test cases can generated test cases. Also, generating dynamic test cases by using sequence diagram as a base element for generating test cases using sequence diagram dynamic actions like interaction among objects can be taken into consideration when generating test cases. Finally, present optimization using discrete particle swarm optimization algorithm. Apply on the test cases generated by dependency graph generated by extracted information from sequence diagram.

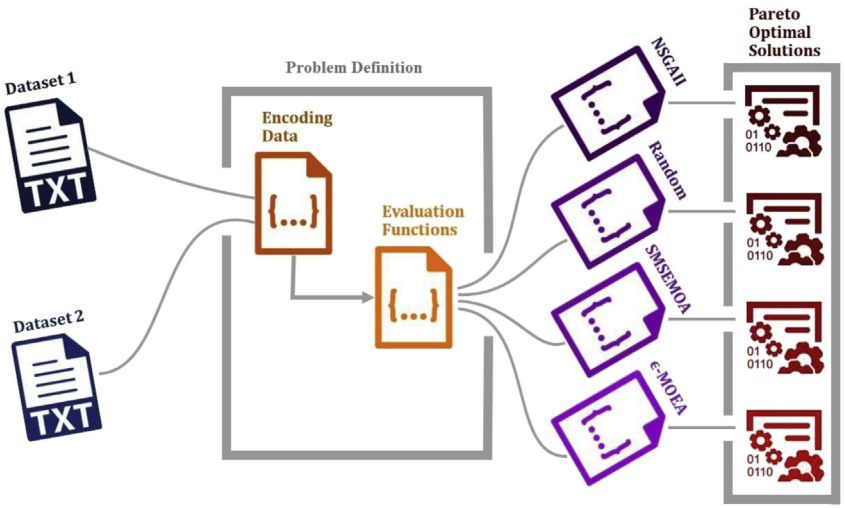

Fig. 1. Proposed approach.

Table 1
eMOEA For Dataset #1.

	Redundancy	Coverage	Cost
Sol #1	27%	80.96%	196.5
Sol #2	27%	81.66%	199.1
Sol #3	28%	82.66%	203.0
Sol #4	28%	81.65%	199.1
Sol #5	27%	78.55%	193.1
Sol #6	25%	81.22%	200.5
Sol #7	25%	79.32%	193.4
Sol #8	25%	81.44%	202.2
Sol #9	25%	80.27%	192.5
Sol #10	28%	82.31%	203.2
Sol #11	25%	79.24%	194.4
Sol #12	28%	83.91%	214.6
Sol #13	27%	80.96%	194.2
Sol #14	28%	82.06%	201.7
Sol #15	25%	80.07%	192.3
Sol #16	27%	80.10%	196.3
Sol #17	25%	79.87%	197.4
Sol #18	27%	82.17%	204.8
Sol #19	27%	80.59%	193.2
Sol #20	27%	80.71%	196.9
Sol #21	28%	82.01%	203.3
Sol #22	27%	79.84%	200.1
Sol #23	28%	81.85%	201.8
Sol #24	27%	79.57%	194.0
Sol #25	27%	82.29%	208.9
Sol #26	25%	79.85%	195.4
Sol #27	27%	80.35%	197.8
Sol #28	25%	79.96%	194.0
Sol #29	27%	79.91%	197.9
Sol #30	27%	77.90%	188.2
Sol #31	24%	79.10%	193.0
Sol #32	24%	77.58%	191.8
Sol #33	25%	80.23%	195.4
Sol #34	28%	81.96%	202.7
Sol #35	25%	78.66%	192.6
Sol #36	25%	78.82%	188.2
Sol #37	28%	81.32%	201.9
Sol #38	23%	75.94%	188.3
Sol #39	25%	79.90%	194.1
Sol #40	28%	82.63%	204.7
Sol #41	25%	80.30%	195.4
Sol #42	27%	79.68%	192.6
Sol #43	27%	80.41%	197.8
Sol #44	27%	80.27%	195.3
Sol #45	28%	79.79%	192.6
Sol #46	25%	79.41%	196.4
Sol #47	28%	83.33%	208.2
Sol #48	27%	81.31%	198.9
Sol #49	27%	80.14%	190.2
Sol #50	27%	80.30%	198.4
Sol #51	25%	77.21%	188.8
Sol #52	29%	84.00%	209.8
Sol #53	27%	81.10%	196.1
Sol #54	27%	79.95%	190.7
Sol #55	27%	79.82%	197.0
Sol #56	27%	80.38%	194.4
Sol #57	27%	79.77%	197.0
Sol #58	27%	81.61%	211.4
Sol #59	27%	77.41%	195.2
Sol #60	25%	79.27%	196.5
Sol #61	27%	79.99%	193.4
Sol #62	28%	83.03%	205.7
Sol #63	25%	77.46%	193.9
Sol #64	28%	80.32%	200.5
Sol #65	27%	79.12%	195.2
Sol #66	25%	78.82%	193.0

3. Proposed approach

In this paper, we proposed a new approach that adopts four different algorithms and three fitness functions to solve multi-objective test cases selection problems in many automated test cases generation tools. Which is facing the challenge of generating duplicated test cases and ineffective within the cost of execution time furthermore give lower coverage percentage also the redundancy among test cases. Where customers search for high-quality products, the software testing activity has grown in importance, aiming to assure quality and reliability to the product under development. Also generating a massive number of test cases is costly, reaching up to 40% of final software development cost. Thus, it is of central importance to improve the efficiency and effectiveness of the whole testing process.

In the testing process, it is necessary to choose a test adequacy criterion defines properties that must be observed by the test suit. The most important metrics we must concern about it

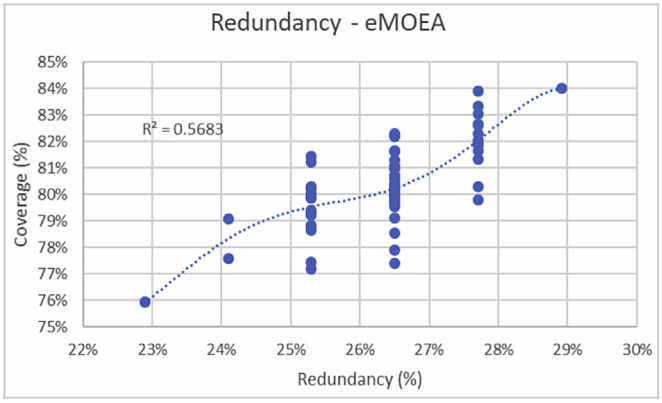

Fig. 2. ε-MOEA Coverage and Redundancy Rel. for Dataset #1.

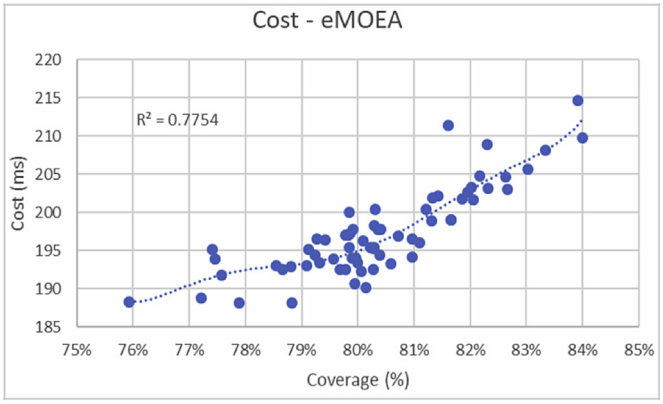

Fig. 3. ε-MOEA Coverage and Cost Rel. for Dataset #1.

Table 2
NSGAII for Dataset #1.

	Redundancy	Coverage	Cost
Sol #1	27%	78.78%	191.5
Sol #2	25%	79.25%	191.0
Sol #3	25%	80.71%	195.5
Sol #4	27%	82.41%	203.4
Sol #5	27%	82.18%	204.8
Sol #6	27%	81.76%	198.8
Sol #7	27%	78.97%	189.7
Sol #8	28%	82.76%	202.8
Sol #9	27%	81.06%	196.3
Sol #10	27%	80.55%	197.2
Sol #11	28%	82.54%	201.7
Sol #12	27%	80.76%	200.2
Sol #13	28%	82.73%	204.4
Sol #14	27%	78.99%	194.9
Sol #15	28%	82.52%	204.4
Sol #16	25%	80.38%	194.3
Sol #17	28%	80.79%	198.4
Sol #18	25%	80.54%	196.8
Sol #19	25%	80.66%	199.6
Sol #20	24%	79.18%	193.1
Sol #21	27%	80.69%	193.0
Sol #22	28%	81.29%	203.3
Sol #23	27%	80.09%	193.1
Sol #24	27%	80.80%	194.8
Sol #25	25%	79.42%	192.7
Sol #26	27%	78.85%	194.4
Sol #27	28%	82.05%	198.3
Sol #28	28%	81.93%	202.9
Sol #29	28%	83.11%	210.7
Sol #30	25%	80.67%	195.6
Sol #31	28%	82.05%	202.5
Sol #32	25%	78.63%	189.2
Sol #33	25%	80.06%	193.8
Sol #34	27%	82.27%	204.6
Sol #35	27%	81.07%	194.8
Sol #36	27%	81.70%	204.8
Sol #37	28%	79.65%	197.7
Sol #38	27%	80.42%	197.2
Sol #39	27%	80.45%	197.6
Sol #40	25%	80.43%	192.7
Sol #41	28%	83.13%	205.4
Sol #42	24%	77.03%	190.2
Sol #43	28%	80.79%	200.4
Sol #44	28%	83.11%	205.4
Sol #45	27%	79.77%	192.3
Sol #46	27%	79.30%	193.5
Sol #47	27%	80.21%	198.3
Sol #48	29%	80.12%	201.3
Sol #49	27%	80.42%	193.7
Sol #50	27%	78.57%	186.9
Sol #51	27%	79.87%	196.8
Sol #52	28%	82.04%	204.1
Sol #53	28%	80.84%	202.4
Sol #54	25%	78.57%	190.5
Sol #55	25%	77.96%	192.6
Sol #56	27%	77.71%	191.4
Sol #57	27%	81.98%	204.1

is the critical objective which is Code Coverage and the constraints which are the cost of execution time and redundancy between generated test cases. The cost and redundancy are the critical constraints to cut down these suits, to fit the available resources, without severely settling the coverage of the test fitness criterion signifying recognized. Reduce the generated test suite based on some selection criterion is known as "Test Case Selection" process.

3.1. Proposed structure

Our approach as in Fig. 1 we use two different datasets of test suits which have a different number of test cases to cover the same program and the number of requirements. Each test suit has different behavior in the total cost and redundancy between test cases. We develop encoder by java programming language to encode the different metrics in each dataset (coverage, cost, and redundancy) for each test case. As said, three fitness functions were adopted. The requirements coverage objective (to be maximized) represents the amount (in percentage) of requirements covered by a solution d in comparison to the number of requirements present in D which is the selected test case from the dataset. Formally, let $X = \{X1, \ldots, Xr\}$ be a given set of X requirements of the original suite D. Let F(Dj) be a function that returns the subset

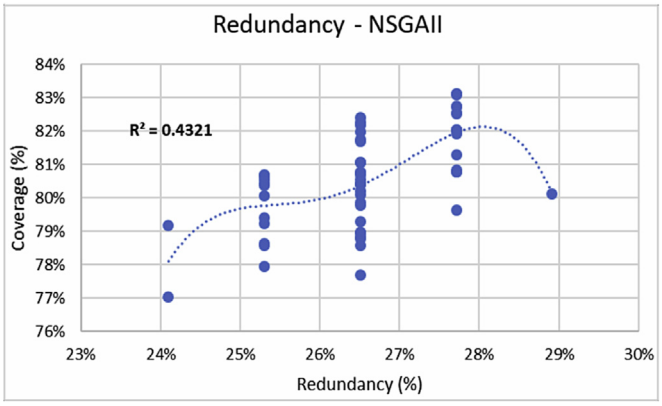

Fig. 4. NSGAII coverage and redundancy Rel. for dataset #1.

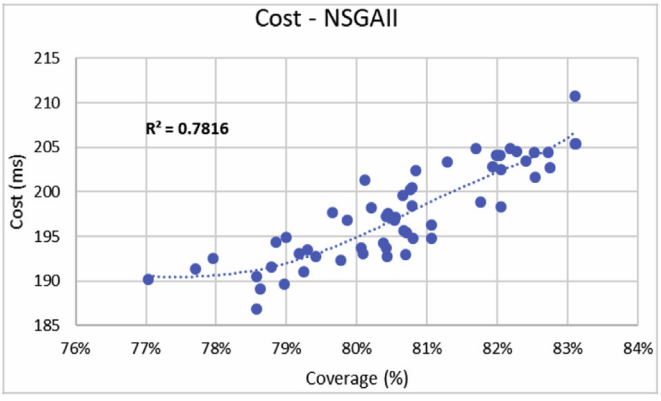

Fig. 5. NSGAII coverage and cost Rel. for dataset #1.

Table 3
SMSEMOA for Dataset #1.

	Redundancy	Coverage	Cost
Sol #1	27%	79.85%	198.4
Sol #2	28%	82.65%	203.0
Sol #3	27%	80.32%	194.9
Sol #4	27%	81.60%	199.6
Sol #5	27%	80.22%	196.9
Sol #6	27%	80.41%	200.5
Sol #7	27%	79.71%	195.6
Sol #8	25%	78.51%	194.1
Sol #9	27%	79.14%	194.3
Sol #10	25%	80.09%	195.1
Sol #11	25%	80.16%	195.9
Sol #12	28%	82.38%	202.4
Sol #13	27%	80.59%	195.9
Sol #14	25%	80.22%	195.0
Sol #15	25%	80.23%	195.9
Sol #16	28%	82.59%	204.0
Sol #17	27%	80.29%	198.3
Sol #18	27%	80.35%	198.3
Sol #19	25%	79.50%	202.1
Sol #20	28%	81.56%	200.7
Sol #21	25%	79.80%	197.9
Sol #22	27%	79.95%	197.8
Sol #23	27%	80.95%	197.0
Sol #24	27%	81.93%	202.6
Sol #25	27%	81.67%	203.5
Sol #26	25%	80.52%	196.1
Sol #27	29%	83.08%	216.5
Sol #28	25%	80.26%	193.5
Sol #29	24%	79.76%	197.7
Sol #30	27%	81.31%	199.7
Sol #31	25%	79.26%	193.5
Sol #32	29%	81.58%	202.1
Sol #33	28%	82.56%	205.2
Sol #34	28%	81.89%	203.2
Sol #35	25%	79.83%	194.6
Sol #36	28%	81.88%	204.9
Sol #37	24%	78.83%	189.8
Sol #38	27%	80.59%	201.0
Sol #39	28%	79.69%	197.2
Sol #40	28%	80.25%	201.0
Sol #41	25%	80.51%	199.6
Sol #42	25%	79.88%	198.1
Sol #43	27%	81.72%	203.7
Sol #44	28%	79.60%	195.9
Sol #45	25%	80.48%	194.9
Sol #46	28%	81.77%	203.6
Sol #47	25%	79.50%	190.4
Sol #48	25%	78.39%	193.8
Sol #49	24%	79.01%	193.9
Sol #50	27%	78.07%	188.4
Sol #51	25%	80.90%	197.5
Sol #52	28%	80.91%	200.9
Sol #53	25%	79.35%	196.9
Sol #54	25%	80.20%	193.0
Sol #55	27%	83.01%	213.3
Sol #56	25%	79.21%	197.0
Sol #57	27%	82.30%	204.5
Sol #58	28%	80.32%	200.4

of requirements in X covered by the individual test case: Dj for each one.

Then, the requirements coverage of a solution d as given:

$$\text{Coverage}(d) = 100 * \frac{|U_{d=1}\{D_i\}|}{x} \quad \text{Unit}: (\%)$$

The Cost fitness function calculated as the summation of run time in milliseconds of each test cases selected in provided solution d from dataset D as given:

$$\text{Cost}(d) = \sum_{i=0}^{i} C_i \quad \text{Unit}: (\text{ms})$$

Redundancy fitness function which is more complicated because of redundancy indicator that retrieves the test cases with redundant coverage of requirements. The fitness function is the total number of redundant test cases divided by the total number of test cases selected in provided solution d.

$$\text{Red}(d) = \frac{\sum R(D_i)}{d} \quad \text{Unit}: (\%)$$

After that, the evaluation functions run with the multi-objective algorithms to find an efficient Preto front regarding the fitness functions Coverage, Cost, and Red.

3.2. Proposed preprocesses

In this section, we will provide our standard tuning to all selected algorithms that are running the above fitness functions

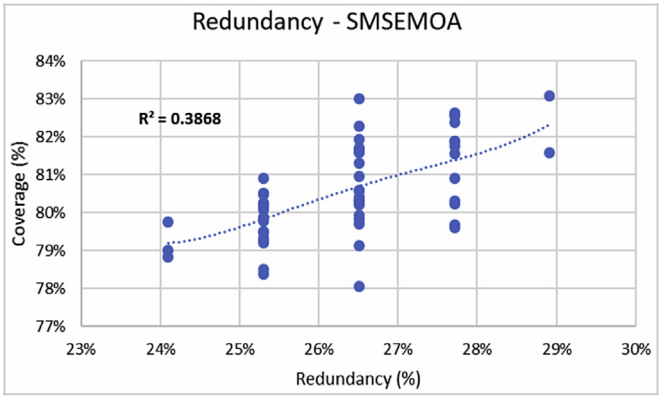

Fig. 6. SMSEMOA coverage and redundancy Rel. for dataset #1.

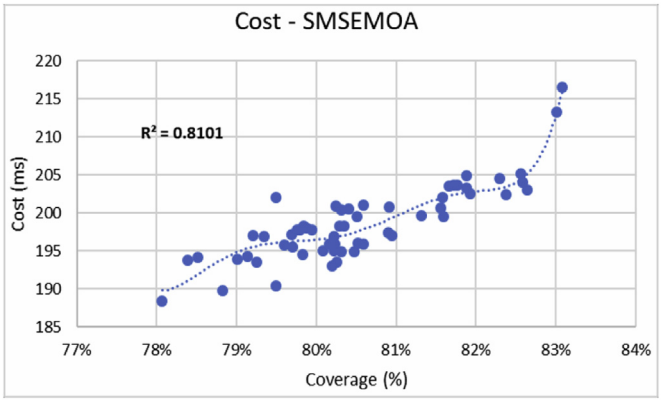

Fig. 7. SMSEMOA coverage and cost Rel. for dataset #1.

Table 4
Random for Dataset #1.

	Redundancy	Coverage	Cost
Sol #1	22%	47.76%	180.0
Sol #2	25%	53.87%	184.9
Sol #3	31%	54.46%	185.7
Sol #4	29%	63.28%	193.9
Sol #5	25%	51.42%	182.5
Sol #6	13%	49.22%	155.8
Sol #7	22%	53.76%	163.4
Sol #8	31%	58.10%	199.6
Sol #9	24%	47.89%	158.5
Sol #10	20%	49.14%	154.7
Sol #11	20%	50.64%	156.1
Sol #12	20%	49.22%	178.2
Sol #13	17%	46.97%	160.0
Sol #14	29%	59.57%	187.4
Sol #15	30%	57.33%	194.5
Sol #16	24%	41.31%	144.5
Sol #17	22%	33.96%	134.0
Sol #18	20%	47.30%	167.1
Sol #19	24%	59.61%	178.4
Sol #20	29%	57.34%	199.0
Sol #21	28%	60.01%	185.8
Sol #22	27%	52.93%	171.0
Sol #23	19%	49.31%	143.9
Sol #24	23%	46.85%	167.7
Sol #25	24%	45.63%	160.1
Sol #26	29%	66.96%	216.1
Sol #27	30%	46.48%	155.2
Sol #28	31%	57.11%	195.2
Sol #29	24%	42.87%	159.5
Sol #30	27%	55.56%	188.3
Sol #31	27%	52.19%	174.0
Sol #32	25%	43.16%	163.9
Sol #33	17%	51.34%	160.7
Sol #34	19%	51.09%	151.9
Sol #35	29%	61.35%	182.8
Sol #36	28%	63.33%	218.0
Sol #37	23%	56.88%	178.0
Sol #38	28%	60.67%	208.8
Sol #39	18%	46.12%	151.9
Sol #40	28%	55.61%	167.0
Sol #41	22%	40.43%	133.0
Sol #42	28%	53.80%	186.3
Sol #43	27%	57.61%	206.1
Sol #44	22%	41.31%	130.3
Sol #45	25%	43.49%	154.2
Sol #46	27%	47.13%	161.0
Sol #47	29%	58.09%	191.3
Sol #48	25%	54.90%	172.1
Sol #49	27%	47.81%	149.9
Sol #50	24%	46.45%	155.7
Sol #51	22%	48.50%	165.3
Sol #52	22%	55.63%	180.2
Sol #53	25%	51.75%	170.3
Sol #54	24%	53.47%	179.0
Sol #55	24%	41.76%	162.7
Sol #56	14%	42.15%	152.1
Sol #57	19%	44.08%	154.7
Sol #58	24%	39.01%	147.2
Sol #59	25%	54.39%	180.3
Sol #60	27%	56.68%	185.5
Sol #61	24%	54.72%	172.9
Sol #62	23%	49.39%	161.6
Sol #63	19%	51.34%	155.0
Sol #64	24%	50.25%	167.9
Sol #65	27%	54.54%	189.7

Table 4 (*continued*)

	Redundancy	Coverage	Cost
Sol #66	24%	45.24%	165.8
Sol #67	27%	59.63%	190.4
Sol #68	20%	44.12%	145.1
Sol #69	19%	43.15%	141.1
Sol #70	23%	47.50%	179.6
Sol #71	19%	46.97%	165.7
Sol #72	24%	50.40%	186.0
Sol #73	25%	48.70%	166.0
Sol #74	22%	49.70%	174.1
Sol #75	18%	43.07%	152.7
Sol #76	28%	57.15%	198.2
Sol #77	24%	52.34%	183.5
Sol #78	20%	48.64%	155.4
Sol #79	23%	51.83%	178.5
Sol #80	30%	57.23%	197.2
Sol #81	23%	48.64%	155.4
Sol #82	18%	51.17%	152.8
Sol #83	27%	58.33%	182.6
Sol #84	22%	51.54%	158.7
Sol #85	24%	62.10%	198.3
Sol #86	24%	46.99%	131.2
Sol #87	25%	47.71%	172.5
Sol #88	25%	50.44%	171.0
Sol #89	23%	53.97%	182.7
Sol #90	24%	42.75%	147.3
Sol #91	29%	55.05%	172.3
Sol #92	20%	45.82%	165.1
Sol #93	33%	50.45%	175.2
Sol #94	19%	55.23%	187.8
Sol #95	23%	55.33%	185.4
Sol #96	20%	52.81%	152.2
Sol #97	22%	52.43%	171.7
Sol #98	22%	51.43%	167.4
Sol #99	27%	55.10%	198.4
Sol #100	23%	47.18%	165.1
Sol #101	25%	50.13%	162.1
Sol #102	20%	51.65%	161.0
Sol #103	25%	61.18%	195.1
Sol #104	24%	53.42%	173.7
Sol #105	18%	46.94%	147.9
Sol #106	20%	45.32%	161.4
Sol #107	27%	59.38%	192.2
Sol #108	25%	49.17%	185.6
Sol #109	23%	42.96%	155.9
Sol #110	28%	58.32%	180.5
Sol #111	22%	52.95%	190.1
Sol #112	23%	49.96%	165.9
Sol #113	23%	49.24%	180.4
Sol #114	24%	61.20%	189.8
Sol #115	23%	49.66%	168.5
Sol #116	23%	55.21%	173.8
Sol #117	22%	43.34%	149.0
Sol #118	25%	49.70%	165.4
Sol #119	25%	43.34%	149.2
Sol #120	20%	40.81%	144.7
Sol #121	20%	45.71%	166.0
Sol #122	29%	62.80%	210.6
Sol #123	20%	47.76%	160.4
Sol #124	30%	55.02%	173.9
Sol #125	27%	55.93%	169.2
Sol #126	28%	59.73%	189.4
Sol #127	25%	49.09%	168.0
Sol #128	22%	42.37%	149.5
Sol #129	25%	55.26%	184.2
Sol #130	27%	50.32%	174.4

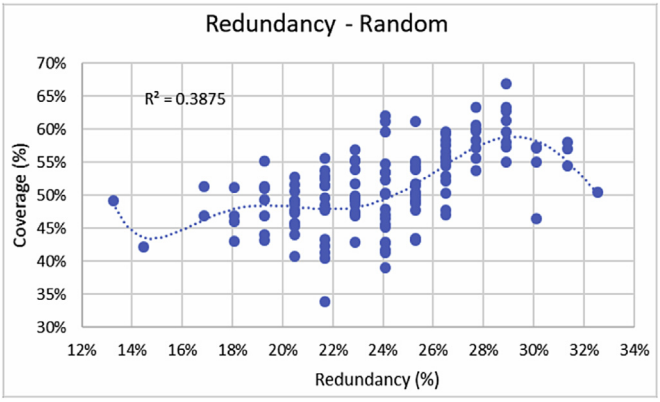

Fig. 8. Random coverage and redundancy Rel. for dataset #1.

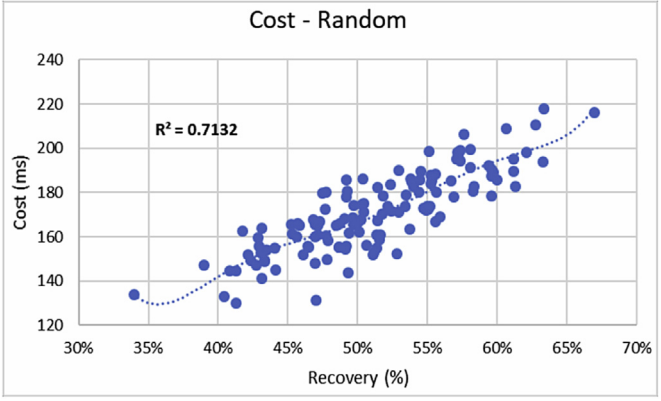

Fig. 9. Random coverage and cost Rel. for dataset #1.

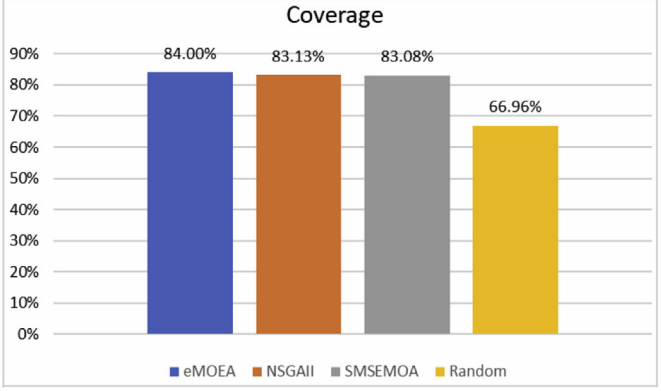

Fig. 10. Code coverage of all algorithms for dataset #1.

on our two datasets. Each algorithm has parameters include the population size, the simulated binary crossover (SBX) rate and distribution index, and the polynomial mutation (PM) rate and distribution index and Max Evaluations. We are tuning each algorithm by the same value to all parameters So that we can measure the efficiency of each algorithm. The tuning parameters values as following:

1) populationSize = 1000
2) sbx.rate = 0.9

Table 5
ε-MOEA for Dataset #2.

	Redundancy	Coverage	Cost
Sol #1	23%	90.49%	199.9
Sol #2	23%	88.68%	192.0
Sol #3	23%	90.08%	200.7
Sol #4	23%	87.22%	205.8
Sol #5	23%	89.24%	198.1
Sol #6	23%	87.70%	192.7
Sol #7	23%	89.48%	199.4
Sol #8	21%	87.16%	189.1
Sol #9	23%	90.71%	202.7
Sol #10	23%	88.92%	193.0
Sol #11	23%	88.60%	193.0
Sol #12	24%	90.66%	204.8
Sol #13	23%	89.94%	198.6
Sol #14	23%	88.74%	192.8
Sol #15	23%	88.42%	197.6
Sol #16	21%	88.15%	192.5
Sol #17	23%	87.62%	199.1
Sol #18	23%	89.15%	196.7
Sol #19	21%	89.72%	199.3
Sol #20	21%	86.04%	185.0
Sol #21	24%	90.95%	208.6
Sol #22	23%	88.24%	196.9
Sol #23	23%	90.77%	203.9
Sol #24	21%	86.93%	198.3
Sol #25	23%	88.04%	204.8
Sol #26	23%	89.18%	196.3
Sol #27	23%	90.56%	201.5
Sol #28	23%	85.68%	191.4
Sol #29	21%	87.15%	195.2
Sol #30	24%	91.48%	208.0
Sol #31	23%	90.27%	199.5
Sol #32	21%	87.51%	197.8
Sol #33	23%	88.45%	190.0
Sol #34	21%	88.98%	193.3
Sol #35	23%	88.42%	196.6
Sol #36	21%	88.72%	192.5
Sol #37	21%	89.44%	197.2
Sol #38	24%	89.48%	206.9
Sol #39	23%	91.31%	206.1
Sol #40	23%	87.53%	194.3
Sol #41	23%	91.83%	207.8
Sol #42	23%	90.07%	198.3
Sol #43	21%	87.34%	185.5
Sol #44	21%	86.43%	189.3
Sol #45	21%	89.10%	196.8
Sol #46	23%	88.60%	194.2
Sol #47	21%	85.38%	189.6
Sol #48	23%	89.20%	198.9
Sol #49	21%	88.62%	198.2
Sol #50	20%	86.67%	189.8
Sol #51	24%	90.77%	207.9
Sol #52	24%	88.61%	207.2
Sol #53	21%	84.92%	185.2
Sol #54	21%	89.27%	197.7
Sol #55	21%	88.06%	199.0
Sol #56	21%	88.69%	197.0
Sol #57	23%	88.50%	192.1
Sol #58	23%	88.65%	197.6
Sol #59	24%	89.24%	199.0
Sol #60	24%	91.27%	203.9
Sol #61	23%	87.73%	198.4
Sol #62	23%	90.61%	203.4
Sol #63	26%	92.08%	211.4

Table 5 (*continued*)

	Redundancy	Coverage	Cost
Sol #64	23%	89.52%	195.4
Sol #65	23%	91.61%	205.1
Sol #66	23%	86.01%	195.1
Sol #67	24%	91.06%	202.4
Sol #68	21%	86.16%	190.6

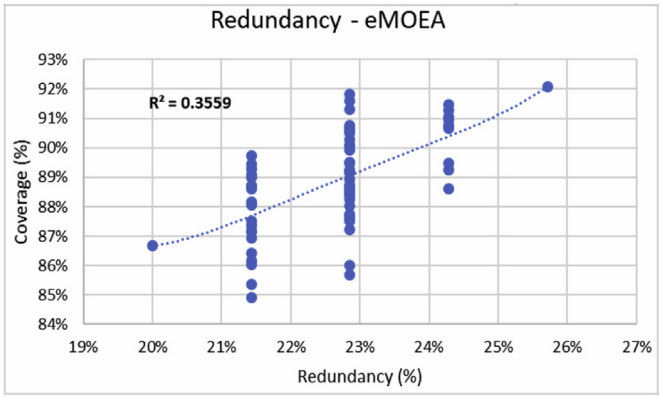

Fig. 11. ε-MOEA Coverage and Redundancy Rel. for Dataset #2.

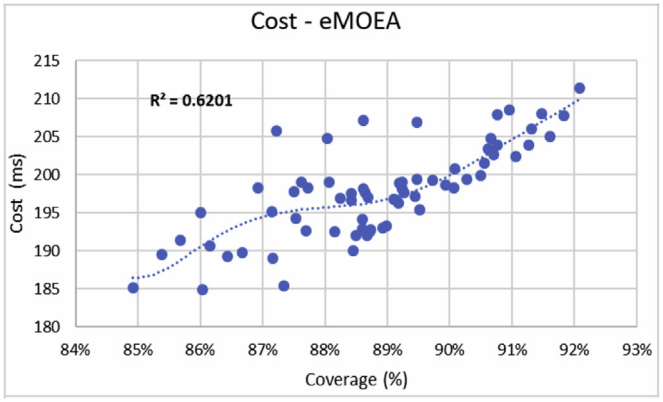

Fig. 12. ε-MOEA Coverage and Cost Rel. for Dataset #2.

3) sbx.distributionIndex = 15.0
4) pm.rate = 0.1
5) pm.distributionIndex = 20.0
6) MaxEvaluations = 1000

4. Data selection

We have two different datasets for implemented program with 745 line of code which is mean we have 745 requirements to covered by our datasets of test cases. The first dataset has 83 test cases with total code coverage 100% and total run time cost 337.6 ms, and total redundancy 40 between test cases. The second dataset have 70 test cases with total code 100% and total run time cost 299.9 ms, and total redundancy 25 between test cases. We select that

Table 6
NSGAII for Dataset #2.

	Redundancy	Coverage	Cost
Sol #1	24%	90.59%	199.8
Sol #2	24%	88.34%	196.8
Sol #3	24%	89.52%	194.4
Sol #4	24%	91.56%	204.3
Sol #5	24%	88.13%	192.3
Sol #6	21%	86.64%	194.4
Sol #7	24%	90.71%	203.3
Sol #8	24%	91.41%	206.0
Sol #9	24%	88.03%	199.6
Sol #10	23%	83.23%	185.6
Sol #11	26%	90.78%	202.9
Sol #13	23%	89.54%	197.0
Sol #15	24%	87.72%	199.0
Sol #16	23%	88.60%	192.2
Sol #17	24%	89.34%	198.0
Sol #18	26%	90%	199.3
Sol #19	23%	86.61%	189.4
Sol #20	24%	88.11%	196.1
Sol #21	23%	89.37%	197.6
Sol #22	23%	88.25%	192.4
Sol #23	23%	87.86%	196.7
Sol #25	26%	91.58%	207.9
Sol #26	24%	87.80%	192.6
Sol #27	24%	91.10%	205.8
Sol #28	23%	90.63%	202.0
Sol #29	24%	88.84%	192.7
Sol #30	23%	88.79%	196.9
Sol #32	24%	90.87%	203.8
Sol #33	21%	88.93%	195.7
Sol #34	24%	91.54%	207.6
Sol #36	24%	88.56%	194.0
Sol #37	24%	89.38%	202.5
Sol #38	26%	90.87%	207.8
Sol #39	23%	87.64%	191.9
Sol #40	23%	84.54%	181.3
Sol #41	26%	91.24%	205.3
Sol #42	24%	89.12%	196.7
Sol #43	23%	89.81%	195.8
Sol #44	24%	88.18%	192.4
Sol #45	24%	91.93%	207.7
Sol #46	24%	89.30%	198.8
Sol #49	24%	90.10%	198.7
Sol #51	23%	83.57%	184.7
Sol #52	23%	90.02%	197.3
Sol #53	24%	88.85%	205.8
Sol #54	26%	88.46%	195.7
Sol #55	23%	88.72%	198.0
Sol #56	24%	88.60%	191.9
Sol #57	24%	88.94%	199.1
Sol #58	23%	86.75%	189.9
Sol #61	24%	88.92%	193.3
Sol #62	24%	87.82%	196.8
Sol #65	24%	88.70%	199.3
Sol #66	26%	91.52%	210.8
Sol #70	24%	88.78%	191.9
Sol #71	24%	87.81%	192.7
Sol #72	26%	91.99%	207.2
Sol #73	23%	88.09%	191.4
Sol #74	23%	88.82%	192.4

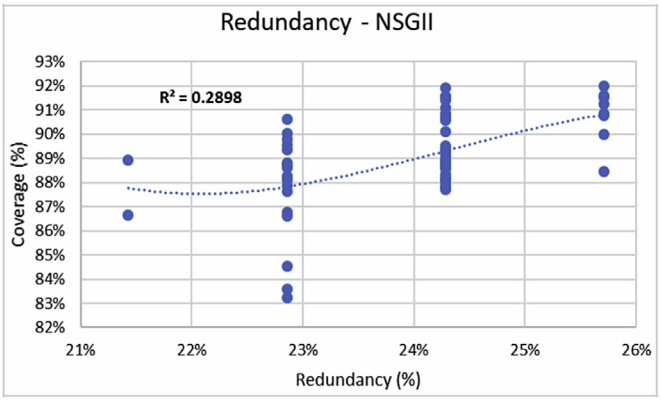

Fig. 13. NSGAII coverage and redundancy Rel. for dataset #2.

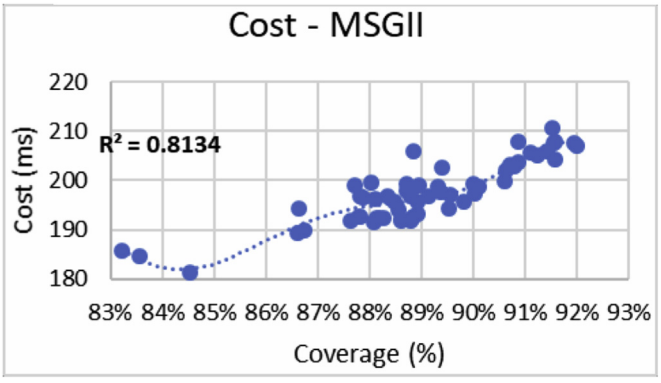

Fig. 14. NSGAII coverage and cost Rel. for dataset #2.

two datasets to prove our concept of select efficient test cases with high code coverage and minimum cost and redundancy.

5. Algorithms selection

Our selected algorithms are implemented within the MOEA Framework and thus support all functionality provided by the MOEA. The four algorithms selected from two different library Native Algorithms and JMetal Algorithms. The NSGAII, ε-MOEA and Random Search are from Native Algorithms library. SMSEMOA from JMetal Algorithms library. Each algorithm of them has different behavior to generate many solutions. We will provide a brief about each of them as follows:

5.1. ε-MOEA

ε-MOEA is a steady-state MOEA that uses ε-dominance archiving to record a diverse set of Pareto optimal solutions [6]. ε-MOEA algorithm the search space is divided into a number of hyper-boxes or grid and the diversity is maintained by ensuring that a hyper-box occupied by only one solution. There are two co-evolving populations: (i) an EA population P(t) and (ii) an archive population E(t) where t is the iteration

Table 7
SMSEMOA Dataset 2.

	Redundancy	Coverage	Cost
Sol #1	23%	87.47%	92.3
Sol #2	23%	90.25%	199.4
Sol #3	21%	89.16%	197.7
Sol #4	23%	89.65%	204.2
Sol #5	24%	91.03%	203.4
Sol #6	21%	87.58%	192.6
Sol #7	21%	87.52%	196.2
Sol #8	24%	88.27%	199.7
Sol #9	24%	85.51%	198.1
Sol #10	23%	88.36%	193.7
Sol #11	21%	89.48%	198.8
Sol #12	23%	88.58%	192.9
Sol #13	23%	89.28%	194.9
Sol #14	21%	88.55%	193.2
Sol #15	21%	89.03%	197.1
Sol #16	23%	89.16%	198.3
Sol #17	24%	88.72%	199.1
Sol #18	23%	89.24%	198.9
Sol #19	23%	89.76%	198.3
Sol #20	23%	88.53%	196.1
Sol #21	23%	87.48%	196.4
Sol #22	24%	90.53%	207.4
Sol #23	23%	87.46%	192.2
Sol #24	23%	90.47%	202.2
Sol #25	23%	91.20%	207.2
Sol #26	23%	87.49%	197.8
Sol #27	24%	88.51%	203.0
Sol #28	24%	90.17%	202.0
Sol #29	23%	87.69%	199.2
Sol #30	21%	89.25%	199.9
Sol #31	23%	87.75%	202.2
Sol #32	24%	89.99%	205.7
Sol #33	23%	91.59%	207.3
Sol #34	21%	87.82%	198.5
Sol #35	23%	87.38%	198.5
Sol #36	23%	88.68%	192.5
Sol #37	24%	90.90%	204.8
Sol #38	24%	91.76%	206.0
Sol #39	23%	88.50%	192.3
Sol #40	20%	85.12%	190.6
Sol #41	21%	88.45%	196.5
Sol #42	23%	88%	196.3
Sol #43	24%	91.24%	207.5
Sol #44	26%	92.61%	219.0
Sol #45	21%	89.89%	201.1
Sol #46	23%	88.94%	195.7
Sol #47	23%	88.91%	196.1
Sol #48	21%	89.20%	196.6
Sol #49	24%	91.33%	207.1
Sol #50	23%	90.54%	203.2
Sol #51	21%	88.38%	197.6
Sol #52	23%	85.09%	194.5
Sol #53	23%	90.32%	201.0
Sol #54	23%	88.44%	191.5
Sol #55	23%	87.79%	191.9
Sol #56	21%	87.12%	193.7
Sol #57	23%	85.01%	184.8
Sol #58	24%	91.06%	206.9
Sol #59	23%	87.88%	193.8
Sol #60	21%	85.83%	190.6
Sol #61	23%	88.62%	201.7
Sol #62	21%	88.48%	192.0
Sol #63	23%	90.53%	203.4

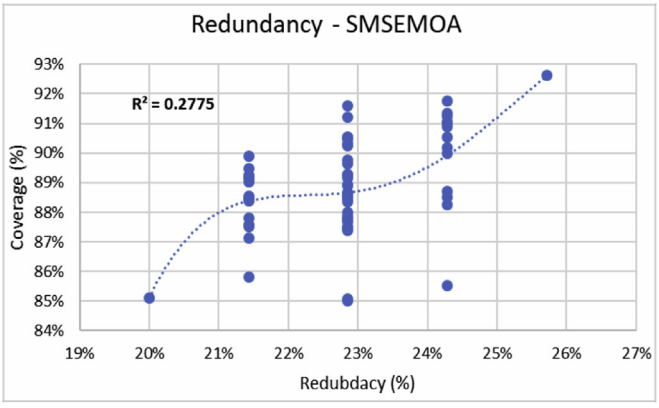

Fig. 15. SMSEMOA coverage and redundancy Rel. for dataset #2.

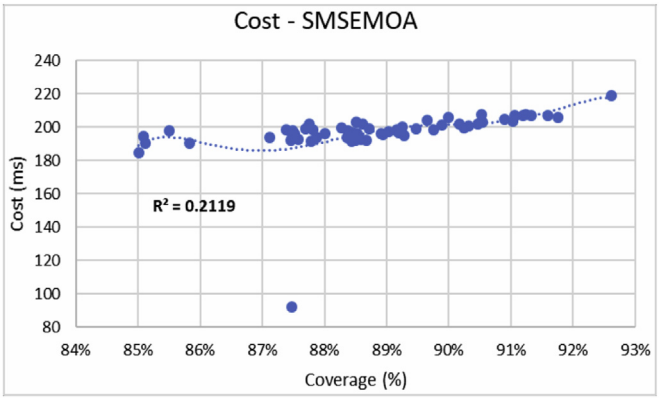

Fig. 16. SMSEMOA coverage and cost Rel. for dataset #2.

counter. The archive population stores the nondominated solutions and is updated iteratively with the e-dominance concept. Moreover, the EA population is updated iteratively with the usual domination [7].

5.2. NSGAII

The NSGA proved to be relatively successful during several years. However, it was an inefficient algorithm due to the way in which it classified individuals. Recent research has proposed an improved version of the NSGA algorithm, called NSGA-II. The non-dominated sorting algorithm-II (NSGA-II) is a generic non-explicit BB MOEA applied to multi-objective problems (MOPs)—based on the original design of NSGA [2].

5.3. SMS-EMOEA

SMSEMOA is an indicator-based MOEA that uses the volume of the dominated hypervolume to rank individuals. The SMS-EMOA has been designed to cover a maximal hypervolume with a finite number of points. The SMS- EMOA combines ideas borrowed from other EMOA, like the well-

Table 8
Random for Dataset #2.

	Redundancy	Coverage	Cost
Sol #1	16%	48.34%	141.9
Sol #2	23%	56.57%	167.9
Sol #3	20%	59.85%	166.9
Sol #4	16%	55.96%	163.4
Sol #5	11%	42.18%	117.8
Sol #6	17%	52.38%	122.9
Sol #7	19%	52.65%	157.1
Sol #8	23%	53.06%	161.4
Sol #9	19%	52.90%	145.8
Sol #10	14%	52.62%	153.1
Sol #11	17%	44.97%	129.1
Sol #12	16%	53.12%	183.3
Sol #13	11%	41.22%	140.7
Sol #14	21%	54.83%	192.7
Sol #15	24%	60.92%	173.0
Sol #16	19%	49.61%	164.1
Sol #17	14%	54.72%	136.2
Sol #18	17%	45.94%	137.9
Sol #19	21%	46.30%	141.4
Sol #20	19%	60.12%	152.6
Sol #21	16%	40.83%	142.1
Sol #22	24%	55.61%	185.9
Sol #23	16%	38.41%	98.3
Sol #24	20%	52.03%	143.6
Sol #25	20%	52.74%	143.8
Sol #26	19%	62.57%	173.7
Sol #27	20%	47.31%	134.5
Sol #28	13%	48.56%	137.8
Sol #29	17%	50.73%	136.7
Sol #30	24%	58.24%	182.3
Sol #31	20%	50.98%	148.5
Sol #32	20%	59.60%	173.5
Sol #33	21%	46.12%	142.7
Sol #34	17%	55.16%	156.9
Sol #35	16%	47.07%	150.4
Sol #36	21%	54.95%	170.9
Sol #37	19%	55.67%	154.5
Sol #38	16%	41.54%	116.7
Sol #39	16%	60.07%	168.2
Sol #40	19%	61.59%	166.5
Sol #41	17%	40.59%	130.9
Sol #42	16%	52.57%	160.6
Sol #43	21%	50.86%	163.5
Sol #44	11%	45.46%	139.6
Sol #45	19%	48.99%	140.0
Sol #46	13%	50.09%	135.2
Sol #47	14%	46.90%	154.1
Sol #48	23%	48.85%	127.4
Sol #49	14%	48.92%	148.1
Sol #50	14%	51.19%	135.3
Sol #51	17%	37.42%	112.5
Sol #52	20%	44.31%	162.5
Sol #53	16%	58.43%	176.9
Sol #54	19%	38.55%	133.0
Sol #55	20%	46.61%	152.0
Sol #56	21%	55.14%	183.2
Sol #57	16%	43.35%	129.5
Sol #58	19%	54.09%	166.6
Sol #59	16%	47.27%	122.9
Sol #60	16%	51.21%	154.7
Sol #61	16%	48.39%	152.6

(*continued on next page*)

Table 8 (*continued*)

	Redundancy	Coverage	Cost
Sol #62	24%	61.43%	188.6
Sol #63	20%	50.11%	159.5
Sol #64	16%	52.46%	136.3
Sol #65	23%	65.31%	193.7
Sol #66	17%	46.54%	136.5
Sol #67	17%	54.11%	163.2
Sol #68	26%	53.71%	148.2
Sol #69	14%	46.87%	130.9
Sol #70	19%	54.81%	146.1
Sol #71	20%	58.78%	187.8
Sol #72	19%	43.67%	141.2
Sol #73	10%	50.18%	156.2
Sol #74	26%	64.33%	188.5
Sol #75	14%	45.85%	142.3
Sol #76	17%	58.10%	157.7
Sol #77	14%	53.32%	133.5
Sol #78	26%	60.64%	188.4
Sol #79	11%	43.75%	136.6
Sol #80	14%	54.42%	145.8
Sol #81	21%	57.62%	174.2
Sol #82	14%	46.91%	118.0
Sol #83	20%	48.53%	147.3
Sol #84	19%	59.59%	176.0
Sol #85	10%	50.97%	147.0
Sol #86	19%	53.94%	135.7
Sol #87	20%	53.45%	139.4
Sol #88	24%	55.37%	155.5
Sol #89	10%	50.15%	126.0
Sol #90	19%	49.96%	146.3
Sol #91	20%	40.35%	98.4
Sol #92	19%	57.19%	144.6
Sol #93	17%	50.96%	153.9
Sol #94	20%	50.17%	148.9
Sol #95	20%	45.63%	131.9
Sol #96	17%	48.31%	147.8
Sol #97	20%	64.23%	181.1
Sol #98	13%	48.27%	159.1
Sol #99	14%	57.27%	172.7
Sol #100	14%	48.82%	156.2
Sol #101	23%	55.07%	162.9
Sol #102	16%	47.54%	108.9
Sol #103	21%	54.78%	160.8
Sol #104	10%	56.22%	129.4
Sol #105	19%	60.49%	152.3
Sol #106	20%	41.28%	143.5
Sol #107	13%	50.48%	143.6
Sol #108	17%	53.40%	132.6
Sol #109	26%	67.43%	211.2
Sol #110	20%	64.44%	153.8
Sol #111	24%	52.58%	162.5
Sol #112	19%	53.37%	176.2
Sol #113	19%	52.38%	140.7
Sol #114	20%	52.26%	171.4
Sol #115	21%	61.85%	194.5
Sol #116	20%	52.67%	166.9
Sol #117	16%	48.39%	149.8
Sol #118	19%	46.12%	135.4
Sol #119	21%	65.15%	191.1
Sol #120	11%	42.26%	118.8
Sol #121	23%	62.02%	167.4
Sol #122	16%	51.02%	163.6
Sol #123	11%	34.33%	118.7
Sol #124	16%	49.91%	131.0
Sol #125	24%	59.30%	172.3

Table 8 (*continued*)

	Redundancy	Coverage	Cost
Sol #126	23%	52.47%	157.4
Sol #127	14%	59.70%	168.7
Sol #128	14%	55.09%	161.1
Sol #129	17%	60.31%	161.6

established NSGA-II and archiving strategies. It is a steady-state algorithm founded on two pillars: (1) non-dominated sorting is used as a ranking criterion and (2) the hypervolume is applied as a selection criterion to discard that individual, which contributes the least hypervolume to the worst-ranked front [8].

5.4. Random

The random search algorithm randomly generates new uniform solutions throughout the search space. It is not an optimization algorithm but is a way to compare the performance of other MOEAs against random search. If an optimization algorithm cannot beat random search, then continued use of that optimization algorithm should be questioned [6].

6. Results and analysis

Multiple experiments are performed to improve code coverage and reduce the cost and redundancy of the selected program that has 745 lines of code for each dataset. Our primary objective to maximize the coverage of 745 line of code we have with a concern about the runtime cost of running the test cases of each dataset we have. In this section, we will present the results of our experiments for each dataset running with the selected four algorithms (ε-MOEA, NSGAII, SMSEMOA, and Random) using our approach and fitness function. We will provide the tables all solutions generated from each algorithm with each dataset. Also, we present a Scatter Chart to show the relation between Coverage and Redundancy, and Coverage and Cost to all provided solutions. In Scatter Char each point in the chart represents a suite of test cases presented in each solution. The Final chart is to show the coverage of best solution for each algorithm per dataset.

In Table 1: we present all results generated from ε-MOEA running on dataset #1 which has 83 test cases. ε-MOEA generates 66 different solutions with the best coverage 84%.

In Fig. 2 and Fig. 3 the Scatter Charts shows the relation between the Code Coverage and Redundancy and Cost for solutions that generated from the ε-MOEA algorithm.

In Table 2: we present all results generated from NSGAII running on dataset #1. NSGAII generate 57 different solutions with the best coverage 83.13%.

In Fig. 4 and Fig. 5 the Scatter Charts shows the relation between the Code Coverage and Redundancy and Cost for solutions that generated from NSGAII algorithm.

In Table 3: we present all results generated from SMSEMOA running on dataset #1. SMSEMOA generate 58 different solutions with the best coverage 83.08%.

In Fig. 6 and Fig. 7 the Scatter Charts shows the relation between the Code Coverage and Redundancy and Cost for solutions that generated from SMSEMOA algorithm.

In Table 4: we present all results generated from Random running on dataset #1. Random algorithm generates massive number of solution which is 130 solutions with the best coverage 66.96%.

In Fig. 8 and Fig. 9 the Scatter Charts shows the relation between the Code Coverage and Redundancy and Cost for solutions that generated from the Random algorithm.

Finally, in Fig. 10. The comparison code coverage chart between the four algorithms. Here we notice that small differences between the first three algorithms ε-MOEA, NSGAII, and SMSEMOA and the worst coverage from the Random algorithm. The random algorithm is the indicator for selection algorithms which is mean if you choose an algorithm and his results are worse than the effects of Random algorithms you should exclude it. The best code coverage is provided by ε-MOEA algorithm with 84%.

In Table 5: we present all results generated from ε-MOEA running on dataset #2 which have 70 test cases. ε-MOEA generates 68 different solutions with the best coverage 92.08%.

In Fig. 11 and Fig. 12 the Scatter Charts shows the relation between the Code Coverage and Redundancy and Cost for solutions that generated from the ε-MOEA algorithm.

In Table 6: we present all results generated from NSGAII running on dataset #2. NSGAII generate 74 different solutions with the best coverage 91.99%.

In Fig. 13 and Fig. 14 the Scatter Charts shows relation between the Code Coverage and Redundancy and Cost for solutions that generated from NSGAII algorithm.

In Table 7: we present all results generated from SMSEMOA running on dataset #2. SMSEMOA generate 63 different solutions with the best coverage 92.61%.

In Fig. 15 and Fig. 16 the Scatter Charts shows the relation between the Code Coverage and Redundancy and Cost for solutions that generated from SMSEMOA algorithm.

In Table 8: we present all results generated from Random running on dataset #2. Random algorithm generates a huge number of solution which is 129 solutions with the best coverage 69.43%.

In Fig. 17 and Fig. 18 the Scatter Charts shows relation between the Code Coverage and Redundancy and Cost for solutions that generated from the Random algorithm.

Finally, in Fig. 19. The comparison code coverage chart between the four algorithms for dataset #2. Also, here we notice that small differences between the first three algorithms ε-MOEA, NSGAII, and SMSEMOA and the worst coverage from the Random algorithm. The best code coverage is provided by ε-MOEA algorithm with 92.61%. In the second dataset, we notice the better solution of code coverage from the first dataset that's because of the second dataset despite having the less number of test case but with less than runtime cost and redundancy between the test cases.

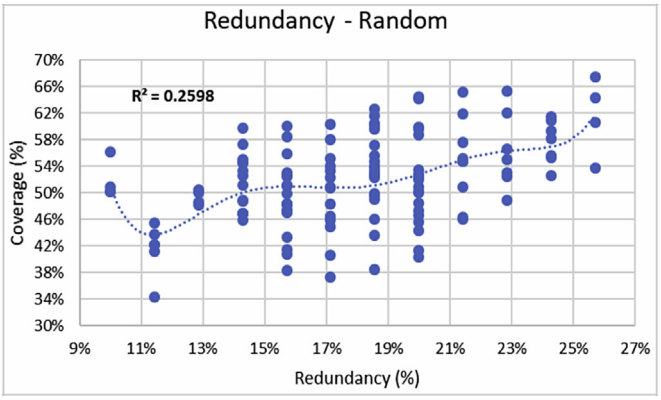

Fig. 17. Random coverage and redundancy Rel. for dataset #2.

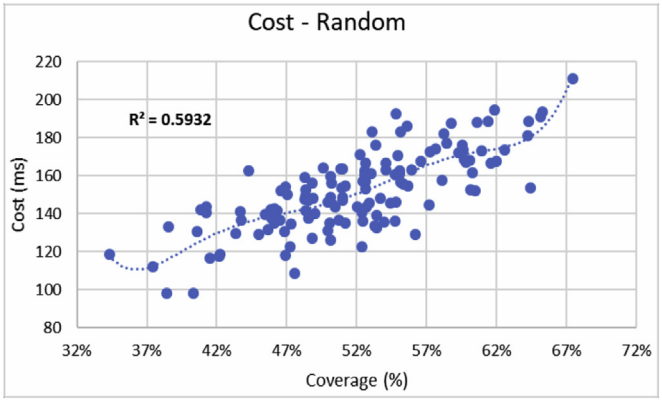

Fig. 18. Random coverage and cost Rel. for dataset #2.

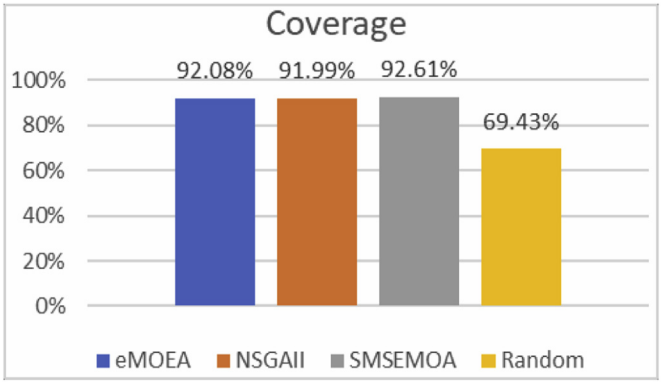

Fig. 19. Code coverage of all algorithms for dataset #2.

7. Conclusion

An efficient approach is proposed and applied to different four algorithms from MOEA Frame from the separate library with three fitness functions for Coverage, Cost, and Redundancy. Our Solution provides an efficient selection of test case for automated test cases generation tools that are suffering from low code coverage and the massive cost of the runtime of test cases cost. The experimental results demonstrate the accurate and efficient selection of test suits provided in each dataset with high code coverage percentage of

92.61%. We apply two different datasets of a different number of test cases to prove our concept of efficient selection of test cases considering the three objectives of maximizing Code Coverage and reducing the Cost and Redundancy.

References

[1] Gómez RHCA, Coello CAC, Alba E. A parallel version of SMS-EMOA for many-objective optimization problems. In: Parallel problem solving from nature − PPSN XIV lecture notes in computer science. 2016. p. 568−77.

[2] "Basic concepts," genetic and evolutionary computation series evolutionary algorithms for solving multi-objective problems. pp. 1− 60.

[3] Souza LSD, Miranda PBCD, Prudencio RBC, Barros FDA. A multi-objective particle swarm optimization for test case selection based on functional requirements coverage and execution effort. 2011 IEEE 23rd International Conference on Tools with Artificial Intelligence. 2011.

[4] de Souza Luciano S, Pruđencio Ricardo BC, Barros Flavia de A. Multi-Objective Test Case Selection: a study of the influence of the Catfish effect on PSO based strategies. In: Anais do XV Workshop de Testes e Tolerância a Falhas - WTF. 2014.

[5] Patidar Chandraprakash. Test case generation using discrete particle swarm optimization algorithm. Int J Sci Res Comput Sci Eng 2013 Jan−Feb.

[6] Li H, Zhang Q. Multiobjective optimization problems with complicated Pareto sets, MOEA/d and NSGA-ii. IEEE Trans Evol Comput 2009;13(2): 284−302.

[7] Obayashi S. Evolutionary multi-criterion optimization: 4th international conference: proceedings. Berlin: Springer; 2007.

[8] Beume N, Naujoks B, Emmerich M. SMS-EMOA: multiobjective selection based on dominated hypervolume. Eur J Oper Res 2007;181(3): 1653−69.

[9] Sakamoto K, Washizaki H, Fukazawa Y. Open code coverage Framework: a consistent and flexible Framework for measuring test coverage supporting multiple programming languages. In: The 10thInternational conference on quality software. QSIC; 2010. p. 262−9.