ISSN 2347 - 3983
Volume 9. No.12, December 2021
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter019122021.pdf
https://doi.org/10.30534/ijeter/2021/019122021

Comparison of Multi-Objective Evolutionary Algorithms to Prioritize Regression Test Cases

HadiAwad[1], Abdel Salam Sayyad[2]
[1] Master of Software Engineering, Birzeit University, Palestine, awadhadi@gmail.com
[2] Master of Software Engineering, Birzeit University, Palestine, abed.sayyad@gmail.com

Received Date : November 06, 2021 Accepted Date : November 28, 2021 Published Date : December 07, 2021

ABSTRACT

Regression testing is one of the most critical testing activities among software product verification activities. Nevertheless, resources and time constraints could inhibit the execution of a full regression test suite, hence leaving us in confusion on what test cases to run to preserve the high quality of software products. Different techniques can be applied to prioritize test cases in resource-constrained environments, such as manual selection, automated selection, or hybrid approaches. Different Multi-Objective Evolutionary Algorithms (MOEAs) have been used in this domain to find an optimal solution to minimize the cost of executing a regression test suite while obtaining maximum fault detection coverage as if the entire test suite was executed. MOEAs achieve this by selecting set of test cases and determining the order of their execution. In this paper, three Multi Objective Evolutionary Algorithms, namely, NSGA-II, IBEA and MoCell are used to solve test case prioritization problems using the fault detection rate and branch coverage of each test case. The paper intends to find out what's the most effective algorithm to be used in test cases prioritization problems, and which algorithm is the most efficient one, and finally we examined if changing the fitness function would impose a change in results. Our experiment revealed that NSGA-II is the most effective and efficient MOEA; moreover, we found that changing the fitness function caused a significant reduction in evolution time, although it did not affect the coverage metric.

Key words: Regression testing, test cases prioritization, meta-heuristic algorithms, NSGA-II, IBEA, MoCell

1. INTRODUCTION

Software testing is one of the essential activities in the software development life cycle; testing activities consume 50% of the total effort and cost of the whole development effort [1]. Regression testing is the process of ensuring that existing features and functionalities of software are still working fine, it ensures that the old code is still working correctly with the new code additions and the new code changes have no effect of what has already been implemented, and to make sure that historical defects did not emerge again in code. Challenges in regression testing are test suite size, time, and resources. In most cases, the time and resources are limited, and test suites are large; hence, the whole regression test suite could not be executed because of the time and cost constraints on running them. Having said that, it becomes more essential to run the test suite partially, either selectively or by prioritizing them to achieve some performance goal.

Many research studies were conducted to minimize the resources needed to run the regression tests with the preservation of high code coverage and the quality of the software product. The techniques applied to that cause are meta heuristic, search algorithms, and SI-based algorithms. The main objectives were to minimize the test case number or identify the order of test cases to achieve effective regression testing in a limited resources environment. It was indicated that Greedy algorithms acted well in solving test case prioritization problems when being compared to other search algorithms like Hill Climbing [2]. Besides, a new proposed hyper-volume algorithm named HGA was found to be more efficient compared to NSGA-II [3]. In addition, Hybrid algorithms were examined, PSO and GA mutation were combined and found that the new algorithms outperform the classic PSO and other algorithms [4].

In this paper, we are going to conduct an empirical study to compare the efficiency of three Multi Objective Evolutionary Algorithms which are NSGA-II [5], IBEA [6] and MoCell [7]. We are going to measure each algorithm's performance in finding the optimal solution by which we can reduce the cost of executing regression tests in terms of time and obtain high fault and branch coverage. The fitness functions used in each of the algorithms are average percentage of branch and faults coverage. Naturally, the best value would be higher faults/branch detected/covered with less execution time.

The remainder of the paper is organized as follows; Section II narrate the related work, section III elaborates more on the problem domain background, section IV talks about the experiment details, research questions and analysis tool, and section V discusses the conducted experiment and results. In

section VI we discuss the threats to validity, and in section VII we summarize and conclude the results.

2. LITRETUR REVIEW

2.1 Single Objective Studies

Zheng et al, examined a group of search algorithms for prioritizing regression test-cases based on statement coverage [2]. They compared greedy methods like Greedy, Additional greedy, and two optimal algorithms with a couple of metaheuristics algorithms which are Hill Climbing and Genetic Algorithms. The results were that Genetic Algorithms performed well when being compared to others, and greedy approaches also did well, given that the nature of landscape was multi-modal. They also investigated the fitness metrics "criterion" that was used in test-cases optimizations and measured its effect on the performance of the algorithm, the criterion they have checked are, Average Percentage of Block Coverage and Decision Coverage and Statement Coverage. They found that the choice of the criterion does not affect the efficiency of the algorithms. In contrast, the size of the test suite does have a direct impact since it determines the search space for the problem. Moreover, they found that the size of the program does not affect the complexity of test cases directly. This study has illustrated the differences between different types of search algorithms, and it compared three Greedy Algorithm with two heuristic techniques. It emphasized how the last overcomes the issues of Greedy algorithms.

Kaur et al, used Genetic Algorithm (GA) for regression test cases scheduling based on code coverage [8]. They explained in detail how to apply the GA for test case prioritization, the initial chromosomes set was randomly generated, then genetic operations are applied until reaching the optimum goal. Their fitness function was based on total code coverage is done by structural testing, which aims to visit all independent paths as a minimum once during path testing, so the GA fitness function was checking if the minimum number of test cases to cover all separate paths. The experiments were done on triangle problems to get the intended paths and 20 test-cases are generated with random paths as inputs for the GA. The GA test-cases order was compared to original, random, reveres and optimal order. They used Average Percentage of Condition Coverage (APCC) to compare the orders, and the results of the experiment demonstrated that GA is performing as expected and its order was as effective as the optimal order.

Pedemonte et al. [9] experimented to identify the performance of Systolic Genetic Search in solving the Test Suite Minimization Problem (TSMP). Besides, they compared the results to the other two competitive genetic algorithms which are Simple Genetic Algorithm and an Elitist Genetic Algorithm, as well as to four heuristics techniques namely GREE, HGSE, GREEDYR, and GREEDYE. Not only the SGS outperform the six algorithms in results, but it showed a considerable time reduction in terms of GPU implementation when being compared to the CPU implementation of SGS. The results also revealed that SGS scales well when being executed on large test suites. The experiment aimed to evaluate the effect of the test's suite size and several testing requirements on the performance of the CPU and GPU implementation when executing the SGS. It's found that the performance of CPU-based SGS has expanded considerably when considering many test requirements, whereas the GPU-based SGS has been minimally affected when being tested on a test suite. The significant point in this experiment is that they studied a test case minimization problem that is cost-aware i.e., they tried to eliminate redundant test cases and select a minimal set of test cases that achieve the testing goals and reduces the number of resources it needs during execution. The experiment was performed over seven real word programs owned by Siemen's benchmark suite; a well-known test suite was used to examine and assess the reductions algorithms. Add to this; the algorithms were evaluated against a case study from real word program which was Cisco regression test suite containing 2000+ test cases and needs around five weeks to be executed

Saraswat and Singhal [4] led a study on a hybrid approach for test case prioritization as well as optimization, and they used the Genetic Algorithm (GA) along with Particle Swarm Optimization (PSO) in their experiment. To validate their findings, they compared the GA_PSO to other optimization algorithms via the Average Percentage Fault Detection (APFD) metric. In the GA_PSO algorithm, the GA was overall an initial population for several iterations to get an optimized result, so, the output of the GA will be the initial population for the PSO, the PSO then optimizes the population to get the optimum order to run based on the APFD. The results of this hybrid algorithm were compared to the following techniques, ORIGINAL ORDER, NON-PRIORITIZED, REVERSE ORDER, RANDOM ORDER, OPTIMUM ORDER, and HYBRID GA_PSO. The results showed that the original order is the worst; the random order could be close to the optimum but never the best. There was a slight difference between the optimum and Hybrid order. Thus, the authors have proved that the hybrid GA_PSO is better than the optimum order of other algorithms.

In the study that was conducted by Mittal and Sangwan [10], they have surfed literature and came out with the examined meta-heuristic algorithms and how they performed in the domain of test selection, minimization, and test prioritization. They also addressed the problem of choosing a proper fitness metric for such an optimization problem and concluded that the Average Percentage of Fault Detected (APFD) is the most frequently and widely used metric nowadays. The study found that ABC outperforms other approaches i.e., GA, ACO, BCO and PSO in the test suite optimization problem as it makes use of the parallel behavior of the employed, onlooker and scout bees which in return make the search process much faster. Add to this that ABC

introduces a balance between exploration and exploitation by employing the scout bees to perform a global search and onlooker bees to perform a local search. In addition to that, the Cuscuta search algorithm has been used and achieved the same results asACO but better than unordered, random, and reverse order.

2.2 Multi-Objective Studies

Di Nucci et al. [3] introduced a new Hyper Volume Algorithm (HGA) and conducted a detailed experiment to examine this algorithm. They used Area Under Curve (AUC) metrics to construct their objective functions, such as Average Percentage of fault/statement/branch coverage of a test-suite. Their Results have proved that the test-case prioritization solutions produced by HGA was more effective in terms of cost than Additional Greedy, GA, and NSGA-II. Moreover, they found that HGA was more efficient and faster than NSGA-II and GA. They also compared HGA with other MOEA like Generalized Differential Evolution and Multi-objective Evolutionary Algorithm Based on Decomposition and found that HGA was 3 times efficient than those algorithms. Lastly, they repeated the experiment on large scale software such as MySQL and observed similar results.

Epitropakis et al. [11] conducted an empirical study on MOEA to prioritize regression test cases executions. Their experiment was the first to analyze the TCP problem as multi objective, they have examined seven different algorithms, three algorithms were different flavors of the additional Greedy Algorithms with different fault detection alternatives and two MOEA which are NSGA-II and TAEA and two algorithms that are hybridizations from MOEAs and the additional greedy algorithms. The algorithms were evaluated on a group of 5 GNU Utility programs from SIR repository. They found that MOEAs and Hybrid algorithms generated solutions that either equal or superior the ones generated by Greedy algorithms. This study used the cost cognizant Average Percentage of Fault Detection APFDc.

Tyagi and Malhotra proposed a three-phase approach to solve the TCP problem [12]. Their approach starts by removing all duplicated test cases using matrices operation, next they selected test cases from a test suite such that the chosen set represents the very minimal set that achieves a high fault coverage with minimal cost possible by applying multi objective particle swarm optimization (MOPSO) algorithm to optimize fault coverage to cost (execution time). Finally, they allocate priority to test cases by calculating the ratio of fault coverage to cost, higher ratio indicates higher priority, then test cases were added in an order that reflects the calculated priority. They compared their approach to other techniques such as No Ordering, Random and Reverse Ordering,they found that their approach outperforms them all. This study calculated APFD of each approach to do the algorithm's comparison.

Tulasiraman's et al. [13] proposed an improved pareto-optimal immune algorithm using three objectives, which are (1)minimizing Execution time (2) maximizing severity fault and (3) maximizing cost-cognizant average percentage of fault detected. Their experiment was conducted over an industrial project that has seven different version. They found out that their approach was superior to other algorithms such as random approach, weighted genetic algorithm, greedy algorithm, and NSGA-II. Their approach starts by analyzing the test suite to extract meta data about test cases such execution time and faults detection, then they apply the Improved Clonal Algorithm to prioritize the test cases.

Marchetto et al. [14] employed a multi objective algorithms to prioritize test cases that aims to maximize the technical and business critical faults while decreasing the execution time of test cases. Their approach started by collecting information about business requirement, test cases and source code, then they applied a metric based technique to figure out critical and error prone parts of the system and gave those a higher priority during test cases ordering. They conducted an empirical experiment on 21 Java programs with their test cases and used NSGA-II to prioritize test suite. They figured out that their approach managed to order test cases in a way that enables them to recover faults as early as possible.

This paper will mainly focus on analyzing a threeelite Multi Objective Evolutionary Algorithms in the domain of prioritizing regression test suites, we check which one is the best in optimizing this problem and find the efficient among them and whether results are affected by the fitness functions. We chose three MOEAs to ensure diversity,NSGA-II is a fast-sorting algorithm that simultaneously optimizes each objective [5], IBEA which mates selections and iteratively removes the worst between them based on a indicators and binary tournament [6] and MoCell uses an external archives and feedback mechanism to get the optimal solutions [7].

3. BACKGROUND

In this section, we are going to discuss in detail the needed knowledge for a better understanding of regression testing and optimization algorithms.

3.1 Regression Testing

While constructing up the software, developers mainly perform functional testing that includes unit, integration, acceptance and smoke testing; before delivering and shipping software, the non-functional testing starts, and that includes performance and scalability testing. During maintenance, regression tests are executed. Computer programs should be adaptable to changes, and most recent software development paradigms enhances adaptability by handling requirements changes and pushing defects' fixes during the development of new features. Even within the waterfall model where new

features are being developed, the older defects should not evolve again. Here comes the significance of regression testing which should verify that the old working programming is still functioning correctly.

Regression tests are executed whenever there is a code change due to requirement change, newly added features, bugs and performance fixes. So, as the software evolves the tests and the number of test cases boost up; Regression test suites are executed partially or entirely, in full mode the whole test suite is being executed, in selection mode, the tests suites are either selected or prioritized. With the aid of Continuous Integration platforms and automation testing frameworks, writingand performing tests became easier; and as a result, the number of test-case being executed in a single regression run is enormous.

Selection techniques and metrics of the tests subset are entirely dependent on the problem at hand and the testing method we are using. Thus, if we are using white software testing in which the design, structure, and implementation of the software are known to us, then we can apply branch coverage, code coverage or statement coverage in the prioritization process. However, if its black testing method, in which we know nothing about the design and implementation as well as not having the course code, then the above methods could not be applied, we need to use different ones such as equivalence partitioning and boundary value analysis.

Regression testing demands a lot of resources and significant time to run it entirely. Thus, development teams tend to reduce this cost and minimizes resources needed by running the tests partially. However, complexity will be faced when they need to determine the subset they are going to run, they need a certain criteria or metric that can help in selecting or prioritizing the test cases to be executed.

Test cases selection criteria and prioritization metrics are broad, we have defect detection, code coverage, test cases complexity, test cases types integration, boundaries, core and so on, and the number of mixes and matches that we could construct is big. One of the optimal goals could be executing a subset of test cases that achieve the best code coverage with less time; another goal would aim to run a subset of the test suite that is most likely to detect a problem as soon as possible and thus give a quality indication to software engineers. Some teams tried first to minimize the number of test cases needed to be carried off and then minimize them. For example, the test cases could be reduced according to their relevance to new code changes and then prioritized according to code coverage.

3.2 Test case prioritization

Test case prioritization can be simply defined as scheduling the execution of test cases according to specific criteria to achieve a goal. This prioritization intends to increase the possibility that the test suite will meet the defined goal when being executed in a constrained environment than it would if it were not prioritized. For example, Quality Assurance managers are willing to run test cases that have higher code coverage before those which have lower code coverage; others would order them against historical fault detection or time of execution. The main objective is not to discard the test cases but to execute them according to the available time and resources. Add to this, test case prioritization is needed to

1. Increase the fault detection rate.
2. Identify risky faults and regression errors earlier in the test process.
3. Maximize code coverage at a faster rate.
4. Enhance the reliability of a system.

Test case prioritization [15] is defined by the following, given *T* a test suite; *PT* the set of permutations of *T*; f is a function from *PT* to the real numbers equation, then find *T'€PT* such that,

$$(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')] \qquad (1)$$

According to the Pareto Optimality [16], the problem formulation can be specified as the following, a test case ordering T_a outperforms the ordering T_b if and only if T_a outperforms T_b in at least one objective and is not worse in all other objectives. Given two test cases ordering, T_a and T_b and a set of n functions (objectives) *f: PT→□*,T_a dominates T_b ($T_a \prec T_b$) if and only if:

$$\begin{aligned} &f_i(T_A) \geq f_i(T_B), \forall i \in 1,2,\cdots,n \\ &\exists i \in 1,2,\cdots,n: f_i(T_A) > f_i(T_B) \end{aligned} \qquad (2)$$

3.3Analyzed Metaheuristic Algorithms

"In computer science and mathematical optimization, a metaheuristic is a higher-level procedure or heuristic designed to find, generate, or select a heuristic (partial search algorithm) that may provide a sufficiently right solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity"[2].

The importance of a metaheuristic algorithm is that it provides us with the optimal solution with the tiniest execution cost. Moreover, they are performing superior in finding a solution to an optimization problem with insufficient or partial information and constrained environment with limited resources. Metaheuristic implies a learning module that guides the whole search procedure i.e. the heuristic. The heuristic is based on two main events, exploration and exploitation; exploration starts by spreading search agents into search space randomly so that to explore all options and avoid the local minimum. Exploitation, in which we assess the explored regions in search space and evaluate it carefully using a fitness metric to check if it contains a good local minimum. The results in metaheuristic methods are usually non-deterministic, meaning, if we provide them with the same

initial population, they will result in different results. Metaheuristic techniques can be classified into two main categories,

1. Single Solution-based, as iterated local search and variable-neighborhood
2. Population-based which can be classified into two more
    a. Evolutionary algorithms like Genetic algorithm
    b. Swarm Intelligence like Particle Swarm Optimization

This study will evaluate three elite Multi Objective Evolutionary Algorithms (MOEA), IBEA, NSGA-II and MoCell. The next sections will elaborate about those algorithms.

A. IBEA

Indicator based Evolutionary Algorithm (IBEA) [6] is one of multi-objective optimization algorithm that is a preference based evolutionary algorithm, its main advantages is that its capable of including the user decisions in the search process. IBEA forms the user preferences as an indicator and will be using it in the selection process of the offspring when the algorithm starts to evolve solution. IBEA solves various Multi-Objective Optimizations Problems (MOPs) by applying three main search components (1) fitness assignment, (2) diversity preservation and (3) elitism. The main contribution of IBEA is that it provides a general framework for indicator based Multiple Objective Evolutionary Algorithms (MOEAs), meaning, several quality indicators could be used and integrated within IBEA main flow to solve a MOPs in any domain.

B. NSGA-II

Non-dominated Sorting Genetic Algorithm II (NSGA-II) is another algorithm from the MOEA family, it has two versions the classical NSGA and the updated one NSGA-II, and most recent one which is NSGA-III [17]. This study takes into consideration NSGA-II due to its popularity solving MOPs and being used a lot in the studies which aims to compare algorithms. NSGA-II is following a genetic algorithm flow in terms of generating population, doing mutation and crossover to generate offspring, and applying selection to select the next generation. What distinguishes NSGA-II is that it sorts the evolved algorithm into a group of sub-population called fronts depending on the Pareto Ranking and crowding measures. NSGA-II has come out to solve two main issues in any MOEA, the first is the complexity of computation needed to evaluate individuals and the second is absence of elitism; NSGA-II employed elitism to preserve the good solutions.

C. MoCell

Multi-Objective Cellular Genetic Algorithm (MoCell) [7] is one of the EA algorithms that belongs to the family of cellular Genetic Algorithms (cGA) for solving MOPs. The main advantage of MoCell is that it uses an external archive to preserve the non-dominated solutions. MoCell allocate the population in a certain structure and apply mutation and crossover genetic operators on some individuals from the population instead of the complete set of individuals. This algorithm makes uses of cellular idea where adjacent individuals are used a lot, that's to say in each iteration every individual only do contact with its neighbor. Having said that, MoCell assures both exploration of search space and exploitation when applying GA operators only within an individual neighborhood and thus, increasing the quality of evolved solutions.

3.4 Objective Functions

It's essential to have measurement criteria to compare the prioritization/order results of used prioritization algorithms. We are considering the white box testing metrics, thus the program source code is available and techniques such as statement coverage or branch coverage can be used. In this paper, we are interested in ordering the test cases to achieve the highest rate of defect defection and branch coverage with the fewest time possible, meaning, the order of test case is better than a different order if the first detects a higher number of faults and branches with minimum execution time. Di Nucci and his team where the first to introduce the Average Percentage of Block and Decision coverage in order to be able to apply the Metaheuristic algorithms [2]. To Formulate a general formula, the Average Percentage of Elements Coverage (APEC) which denotes the weighted average of elements being covered by the test-cases execution can be calculated by the following equation "equation (3)" [18]

$$APEC = 1 - \frac{\sum_{i=1}^{m} TE_i}{n \times m} + \frac{1}{2 \times n} \quad (3)$$

Where,
T: The test suite under evaluation
m: The number of elements contained in the program under test P
n: The total number of test-cases
TE_i: The position of the first testin T that covers the element i

Element could be a fault, a certain branch or statement in the Program Under Test. The test-cases order would be better if it finds the maximum number of elements earlier than other orders.

The previous equation assumes that all elements to be covered have the same cost, but that assumption is not valid in real or industrial life. Thus, another formula will be used as the objective function in which we take into account the cost. In principle, the cost of test-case is the time it needs to be fully executed. Having said that the next equation "equation (4)" shows a different variant of the first one in which we consider the test case cos.

$$APEC_c = \frac{\sum_{i=1}^{m} \left(\sum_{j=TE_i}^{n} c_j - \frac{1}{2} c_{TE_i} \right)}{\sum_{i=1}^{n} c_i \times m} \quad (4)$$

If we assumed that all test cases have the same execution time (cost) (i.e.,$\forall c_i$ € C, $c_i = 1$), Equation 4 becomes equivalent to Equation 3.

4. EXPERIMENTAL DETAILS

This section elaborates on the proposed research method, which is based on one of the empirical strategies, an empirical experiment.

4.1 Research Questions

The following research questions have encouraged this study,

- **(RQ1)** Which algorithm is the best in finding the optimal order of test cases?
- **(RQ2)** Which algorithms is more efficient in TCP optimization problem?
- **(RQ3)** Does the results change if we changed the objective (evaluation) function?

The first question is concerned about checking the quality of the applied MOEA in optimizing the order of test cases, and whether the test case execution time is minimized while elements covering is maximized while running the regression test suite in the order generated by each algorithm. Whereas the second question aims to examine the generation time of each algorithm and checks which one is the most efficient. Our third question will examine if changing the objective function would affect the results of the first and second question.

4.2Experimental Subjects

The subjects used in this experiment are part of the GNU utilities. This study chose 6 programs which are bash, sed, flex, grep, printtokens and printtokens2 which are collected from Software-artifact Infrastructure Repository (SIR) [19]. These subjects were selected because it was used in different researches [19], [20] and [21]. All the details of the Program under test are being illustrated in Table 1, and that includes line of code, test suite size and number of faults. The selected subjects have a LOC between 5680 and 59846, and number of test cases is between 214 and 1061.

4.3 Experimental Design

The objective of the experiment is to find how well each of the two algorithms prioritizes test cases in terms of the objective function metric. The experiment methodology is to use the datasets of Experimental Subject section 4.2 and run the algorithms to get the order and finally examine and analyze the results. This experiment will mainly use two cost aware objective functions which will be calculated based on Equation (4),

- Average Percentage of Fault Coverage $APFC_c$
- Average Percentage of Branch Coverage $APBC_c$

The algorithms will be used to prioritize the test cases to find the order with higher APEC value and lower execution time, the number of max-evaluations for all algorithms is unified and was set to 25000. Hence, we neutralize the time factor in running the algorithms. Table 2 shows the Genetic algorithms configuration used in this study. The reason we chose those is that they have been used previously in other studies [22] and [11].

Table 1: Experiment Subjects

Program	Description	Line of Code	Number of Tests	Number of Faults	Language	Fault Type
Bash	Shell Language Interpreter	59,846	1,061	5	C	Seeded
Flex	Fast Lexical Analyzer	10,459	567	15	C	Seeded
Grep	Regular Expression Utility	10,068	809	10	C	Seeded
GZip	Compression Tool	5,680	214	11	C	Seeded
Sed	Non-Interactive Text Editor	14,427	360	5	C	Seeded

Table 2: Algorithm's Parameters Settings

Parameter	Value
Population size	250
Max number of Evaluations	25000
Generations	100
Independent runs	30
Crossover rate	0.9

4.4 Experiment Run and Analysis Tool

All algorithms have been implemented within jMetal [23] which is java-based tool for single and multi-objective optimizations. The execution will take place on a computation machines hosted on Google Cloud Platform with a particular specification that helps in reducing the computational time of executing the test cases.

Firstly, we will compare the quality of the solutions generated by algorithms between each other using the output of quality indicators, where text cases execution time is unified to determine which algorithms perform the best. This will be done by calculating the medians and means for each quality indicator for each algorithm for all runs, and thus we will be able to answer our first Research question. Afterward, we will compare how algorithms behaved by comparing the $APEC_c$ of all algorithms against each program under test to answer our second research question. Finally, the data of the experiments will be statically analyzed using Friedman and Wilcoxon t-test for pair wise comparison [24].

5. RESULTS AND DISCUSSIONS

In this section we will be discussing the results that we collected during ruining the experiment.

5.1 Solution Quality Analysis

In order to answer our first question, we will present the whole statistics data that describes the quality indicators of the

Pareto fronts. To achieve this, we measured both the Spread and HV quality indicators for all Pareto fronts generated by algorithms for all subject programs in each independent runs. Thus, we have 30 quality indicators for each algorithm executed over a subject program. Table 3 and Table 4 show the means of HV and Quality indicator forfound Pareto fronts.

Table 3: HV. Median and Interquartile Range

	NSGA-11	MoCell	IBEA
bash	$5.50e-01_{2.8e-03}$	$5.37e-01_{1.0e-02}$	$5.46e-01_{2.5e-03}$
flex	$6.98e-01_{6.4e-04}$	$6.94e-01_{4.4e-03}$	$6.97e-01_{4.3e-04}$
grep	$3.21e-01_{2.0e-04}$	$3.19e-01_{8.6e-04}$	$3.20e-01_{3.7e-04}$
printtokens	$9.43e-01_{8.7e-05}$	$9.43e-01_{2.5e-04}$	$9.43e-01_{1.3e-04}$
printtokens2	$9.79e-01_{1.9e-05}$	$9.79e-01_{6.4e-05}$	$9.79e-01_{2.6e-05}$
sed	$6.12e-01_{1.7e-04}$	$6.10e-01_{1.5e-03}$	$6.12e-01_{1.7e-04}$

Table 4: SPREAD. Median and Interquartile Range

	NSGA-11	MoCell	IBEA
bash	$1.00e+00_{7.8e-03}$	$1.00e+00_{4.4e-03}$	$9.99e-01_{2.2e-03}$
flex	$1.00e+00_{2.7e-04}$	$1.00e+00_{2.7e-04}$	$1.00e+00_{7.1e-04}$
grep	$1.00e+00_{1.4e-04}$	$1.00e+00_{5.0e-05}$	$1.00e+00_{5.4e-04}$
printtokens	$1.00e+00_{5.6e-04}$	$1.00e+00_{2.8e-04}$	$1.00e+00_{3.7e-04}$
printtokens2	$1.00e+00_{5.0e-04}$	$1.00e+00_{1.0e-04}$	$1.00e+00_{2.2e-04}$
sed	$1.00e+00_{4.9e-04}$	$1.00e+00_{1.6e-04}$	$1.00e+00_{1.5e-04}$

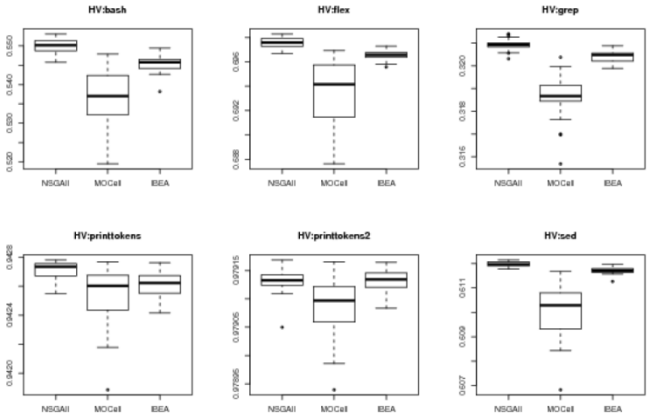

Figure 1: HV Quality Indicator

In terms of HV quality indicators, and if we look at Table 3 and figure 1, we can see that NSGA-II has achieved the best Pareto Front and outperforms both IBEA and MoCell. Yet, IBEA has achieved a better result and was somehow close to NSGA-II. MoCell achieved the worse results. As for the spread of the solutions on the Pareto, it's obvious that the three algorithms have achieved similar results regardless of the program under test. Yet, as shown in Table 4 and Figure 2, IBEA has achieved a slightly better results than the NSGA-II and MoCell. Based on the above measurements of Spread and HV quality indicators, the algorithms can be ranked according to Friedman statistical analysis.

Table 5 below shows the ranking of algorithms according to HV measurements considering reduction performance (distributed according to chi-square with 2 degrees of freedom: 12.0). We can notice that NSGA-II has achieved the highest ranking with 3.0 score followed by IBEA with 1.98 and Finally NSGA-II

Table 5: Average ranking of the algorithms considering HV

Algorithm	Ranking
MoCell	3.0
IBEA	1.98
NSGA-II	0.99

Table 6: Average ranking of the algorithms considering Spread

Algorithm	Ranking
MoCell	1.33
IBEA	2.0
NSGA-II	2.65

Table 6 above shows the algorithms ranking considering the Spread quality indicator measurements and reduction performance (distributed according to chi-square with 2 degrees of freedom: 5.333333333333336). We found that IBEA achieved the best Spread score followed by MoCell and NSGA-II respectively

To better understand how the algorithms optimized the TCP problems; Figure 3 and Figure 4 show the Pareto front of the three algorithms when trying to generate test cases for bash and flex programs. We can see that NSGA-II has generated in most of the cases a non-dominated set of solutions. All the data and the rest of the figures can be found publicly here [1].

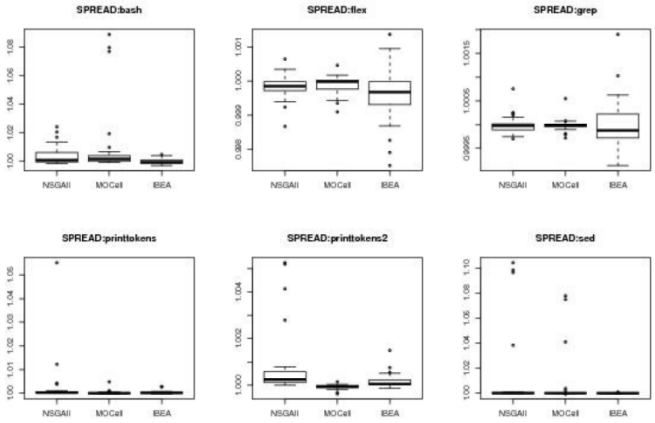

Figure 2: Spread Quality Indicator

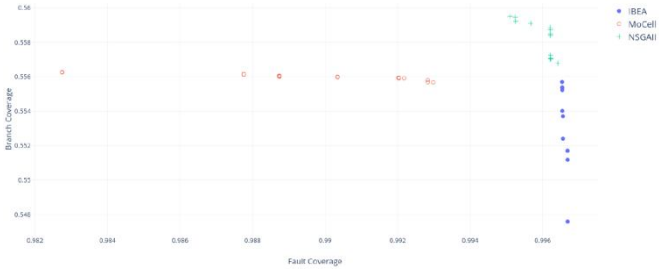

Figure 3: Pareto front of the algorithms when trying to generate test cases for bash

[1]Experiment Github repo.URL:https://github.com/HadiAwad/MOEA_TCO

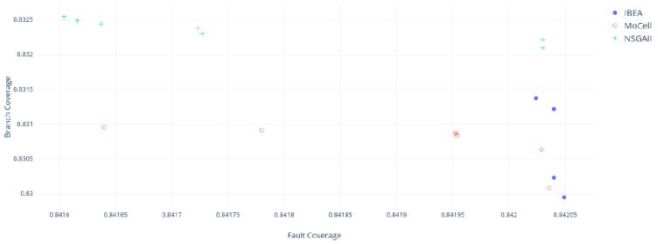

Figure 4: Pareto front of the algorithms when trying to generate test cases for flex

Table 7: Evolution Time of Algorithms for each Program

	Bash	Flex	Grep	Printtokens	Printtokens2	Sed
IBEA	2341201.77	84802.90	241259.33	41569.93	39020.80	38770.93
MoCell	2391202.27	77985.90	239448.97	34303.67	32308.13	30649.57
NSGA-II	2370850.00	76446.40	235666.50	30721.70	28322.97	29955.30

Thus, we can conclude that NSGA-II has generated the best solutions and outperformed other algorithms in test-cases prioritization problem. Besides, NSGA-II algorithm has produced a set of solutions that were well spread over the Pareto since it achieved a close Spread scores when being compared to its rivals.

5.2 Efficiency Analysis

Regarding the efficiency and to answer the second question, we have measured the time needed by each algorithm to optimize and order the test cases. Table 7 shows the average time consumed by each algorithm when being invoked for a certain subject.

Figure 5 displays the time in box charts, we have eliminated the "Bash" subject for better illustration. Its very clear that there is no significant differences between the algorithms. However, to answer our second question, we found that NSGA-II has the least execution time, followed by MoCell and finally IBEA. It's worth noting here that this result is expected and makes sense since IBEA is well known to have a high evolution time due its high computation cost in

5.3 Fitness Function Effect

To answer the third question, we have repeated the experiment but, in this time, we have used Average Percentage of Element Coverage regardless of the cost as a fitness function. Meaning, equation (3) is used in evaluating the populations during evolution, which means that all test cases have the same cost, thus cost is unified or neutralized in the second part of the experiment.

Tables 8 and Table 9 show the Median of HV and Spread quality indicators respectively when the cost was unified. When comparing the results, we found that the quality of the solutions of the algorithms did not change upon changing the fitness function; NSGA-II achieved again the highest HV and thus the best solutions and outperformed its competitors. However, in terms of solutions spread, MoCell achieved the best spread and outperformed both NSGA-II and IBEA.

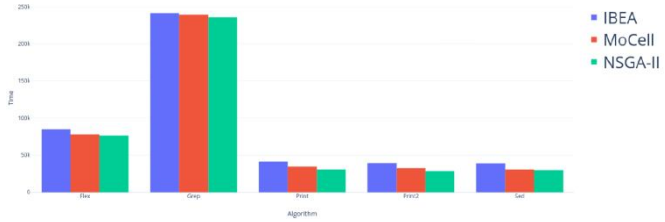

Figure 5: Box Charts of Evolution time

Table 8: HV. Median and Interquartile Range for Unified Cost

	NSGA-11	MoCell	IBEA
bash	$5.52e-01_{2.7e-03}$	$5.37e-01_{8.8e-03}$	$5.48e-01_{2.7e-03}$
flex	$6.96e-01_{9.4e-04}$	$6.90e-01_{3.4e-03}$	$6.94e-01_{1.7e-04}$
grep	$3.21e-01_{3.7e-04}$	$3.19e-01_{1.2e-03}$	$3.20e-01_{4.0e-04}$
printtokens	$9.42e-01_{1.5e-04}$	$9.42e-01_{4.1e-04}$	$9.42e-01_{3.4e-04}$
printtokens2	$9.79e-01_{1.2e-04}$	$9.79e-01_{1.0e-04}$	$9.79e-01_{9.3e-05}$
sed	$6.07e-01_{4.9e-04}$	$6.03e-01_{1.7e-03}$	$6.06e-01_{7.6e-04}$

Table 9: HV. Median and Interquartile Range for Unified Cost

	NSGA-11	MoCell	IBEA
bash	$9.99e-01_{3.3e-03}$	$9.99e-01_{2.0e-03}$	$9.99e-01_{2.8e-03}$
flex	$1.00e+01_{1.0e-03}$	$9.99e-01_{6.5e-04}$	$1.00e+00_{9.2e-04}$
grep	$1.00e+01_{1.6e-03}$	$9.99e-01_{7.2e-04}$	$9.99e-01_{7.8e-04}$
printtokens	$1.00e+01_{1.1e-03}$	$1.00e+01_{3.5e-04}$	$1.00e+01_{1.1e-03}$
printtokens2	$1.00e+01_{3.1e-04}$	$1.00e+01_{2.3e-04}$	$1.00e+01_{2.2e-04}$
sed	$1.00e+01_{6.0e-03}$	$9.99e-01_{9.2e-04}$	$1.00e+01_{7.8e-04}$

Next, we will check if the ranking of algorithms has changed when changing the fitness function, Tables 10 and Table 11 show the Friedman test results of HV and Spread quality indicators for the subject algorithms.

We can notice that the order has not changed for both quality indicators, but it's worth shading light on the fact that IBEA has achieved a better HV score compared to the first part of experiment.

As for evolution time, the following table 12 shows the algorithm evolution time when cost is unified among all test cases. If we look back at Table 7, we can see that in most of the subjects (except the Bash program) the execution time was less when fitness function was APEC, thus, we can conclude that the choice of fitness functions has a considerable effect on the evolution time.

Table 10: HV Average ranking of the algorithms when Cost is unified

Algorithm	Ranking
MoCell	2.835
IBEA	2.165
NSGA-II	0.999

Table 11: Spread Average ranking of the algorithms when Cost is unified

Algorithm	Ranking
MoCell	1.165
IBEA	1.833
NSGA-II	3.0

Table 12: Evolution time when fitness function is calculated according to equation 3

	Bash	Flex	Grep	Printtokens	Printtokens2	Sed
IBEA	2412699.35	83875.3	222938.15	39686.6	38199.95	36505.4
MoCell	2443904.3	76785.65	218515.05	32711.95	31979.15	29725.0
NSGA-II	2389893.0	74192.55	213061.35	29395.85	28233.9	29005.25

Table 13: HV Pairwise Comparison for (bash, flex, grep, printtokens, printtokens2, sed)

	MoCell	IBEA
NSGA-II	▲ ▲ ▲ ▲ ▲ ▲	▲ ▲ ▲ ▲ – ▲
MoCell		–

Table 14: Spread Pairwise Comparison for (bash, flex, grep, printtokens, printtokens2, sed)

	MoCell	IBEA
NSGA-II	– – –	– –
MoCell		– ▲▲–

5.4 Statistical Analysis

To measure if the randomness aspect of GA algorithms has any effect on the observed results, we have conducted statistical analysis to check weather our results are statistically significant or not. Previously, we have shown the Friedman test results, now we will be showing the Wilcoxon pairwise tests considering significance level as 0.05. Table 13 and Table14shows the pairwise comparison between algorithms over HV and Spread quality indicators respectively. And we can see that there is a significant difference in most of the cases, thus our results are statistically proven that they were not affected by GA Randomness.

6. THREATS TO VALIDITY

This section analyzes the threats of validity, which cares about the possibility of endangering the trustworthiness of the study or the existence of biasing factors that could affect the proposed empirical research and its results.

Threats to construct validity, we identified two significant risks related to the experiment design, the first one is concerned about the choice of our effectiveness metric AFPD which does not consider the severity of faults. We tried to mitigate this by taking into consideration another important metric which is test execution time (cost). Yet, in future work and since this experiment is repeatable, we might introduce one more effectiveness metric to capture the effectiveness of prioritization fully. The second one is related to the nature of metaheuristic algorithms which could generate different optimized solutions for the same input. This could cause the generation of solutions that are better than the real optimized ones and thus the existence of outliers. We mitigated this by repeating the algorithms 30 times and conducting statistical analysis tool to make sure that the collected results are statistically different.

Threats to internal validity, the main threat we identified is the application used in implementing the algorithms (jMetal), and its hidden effect on the results. This was mitigated by conducting a small experiment on a small dataset in which the optimum order is known to us, the algorithms were calculated 30 times and all result were compared to the optimal solution and found that they are realistic. Thus, we have high confidence that the algorithms implementation is correct.

Threats to external validity, which is concerned about the generalization of the study and the represent-ability of the experimental subjects. The first threat could be related to the size of programs; sizes of the programs used in this experiment are between medium and large, and the test cases numbers are somehow huge which makes us assume that the results could be generalized. Add to this, the subject programs were used extensively in this problem domain.

7. CONCLUSION

This paper conducted a comparison between three popular Multi Objective Evolutionary Algorithms in prioritizing regression test cases problems. IBEA, MoCell and NSGA-II were analyzed. The aim of this study was to find out which algorithms generates the best solution, and which is the most efficient one. Moreover, we intended to examine if changing the fitness functions influences the results and evolution time. Two experiments were conducted, one that takes the test cases execution cost into consideration and the others assumes that all test cases have the same cost (execution time). In The first part, we found that NSGA-II is the most effective and efficient algorithm since it generated the optimal solutions and found to have the less evolution time. In the second part, we changed the fitness function nature and checked what effect does it have on the results. We found that the results did not change significantly, NSGA-II still outperforms IBEA and MoCell in both optimizing the problem and having the less evolution time. We observed a general reduction in evolution time for all algorithms, and that was totally dependent on the fitness function nature which in the second part needed less execution time.

Future work may include analyzing other algorithms, as well as analyzing more fitness functions and objectives to further understand how the algorithms behave.

REFERENCES

1. C. Doungsa-ard, K. P. Dahal, M. A. Hossain, T. Suwannasart, **An automatic test data generation from uml state diagram using genetic algorithm**. (2007). *on Neural Networks*, Vol. 4, pp. 570-578, July 1993.
2. Z. Li, M. Harman, R. M. Hierons**, Search algorithms for regression test case prioritization**, IEEE Transactions on software engineering 33 (4) (2007) 225–237.
3. D. Di Nucci, A. Panichella, A. Zaidman, A. De Lucia, **A test case prioritization genetic algorithm guided by the hypervolume indicator**, IEEE Transactions on Software Engineering (2018).
4. P. Saraswat, A. Singhal, **A hybrid approach for test case prioritization and optimization using meta-heuristics techniques**, in: 2016 1st India

International Conference on Information Processing (IICIP), IEEE, 2016, pp. 1–6.

5. K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, **A fast elitist non-dominatedsorting genetic algorithm for multi-objective optimization**: Nsga-ii, in: International conference on parallel problem solving from nature, Springer, 2000, pp. 849–858.
6. E. Zitzler, S. Künzli, Indicator-based selection in multiobjective search,in: **International Conference on Parallel Problem Solving from Nature**, Springer, 2004, pp. 832–842.
7. A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, E. Alba, Mocell: **A cellular genetic algorithm for multiobjective optimization**, International Journal of Intelligent Systems 24 (7) (2009) 726–746.
8. A. Kaur, S. Goyal, **A genetic algorithm for regression test case prioritizationusing code coverage**, International journal on computer science and engineering 3 (5) (2011) 1839–1847.
9. M. Pedemonte, F. Luna, E. Alba, **A systolic genetic search for reducing the execution cost of regression testing**, Applied Soft Computing 49 (2016) 1145–1161.
10. S. Mittal, O. P. Sangwan, **Prioritizing test cases for regression techniques using metaheuristic techniques**, Journal of Information and Optimization Sciences 39 (1) (2018) 39–51.
11. M. G. Epitropakis, S. Yoo, M. Harman, E. K. Burke, **Empirical evaluation of pareto efficient multi-objective regression test case prioritization,** in: Proceedings of the 2015 International Symposium on Software Testing and Analysis, 2015, pp. 234–245.
12. M. Tyagi, S. Malhotra, **Test case prioritization using multi objective particle swarm optimizer**, in 2014 International Conference on Signal Propagation and Computer Technology (ICSPCT 2014), IEEE, 2014, pp. 390–395.
13. M. Tulasiraman, N. Vivekanandan, V. Kalimuthu, **Multi-objective test case prioritization using improved pareto-optimal clonal selection algorithm**, 3D Research 9 (3) (2018) 32.
14. A. Marchetto, M. M. Islam, W. Asghar, A. Susi, G. Scanniello, **A multiobjective technique to prioritize test cases**, IEEE Transactions on Software Engineering 42 (10) (2015) 918–940.
15. G. Rothermel, R. H. Untch, C. Chu, M. J. Harrold, **Prioritizing test cases for regression testing, IEEE Transactions on software engineering** 27 (10) (2001) 929–948.
16. C. M. Fonseca, P. J. Fleming, **An overview of evolutionary algorithms in multiobjective optimization,** Evolutionary computation 3 (1) (1995) 1–16.
17. K. Deb, H. Jain, **An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach**, **part i: solving problems with box constraints**, IEEE transactions on evolutionary computation 18 (4) (2013) 577–601.
18. R. Krishnamoorthi, S. S. A. Mary, **Regression test suite prioritization using genetic algorithms**, International Journal of Hybrid Information Technology 2 (3) (2009) 35–52.
19. H. Do, S. Elbaum, G. Rothermel, **Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact**, Empirical Software Engineering 10 (4) (2005) 405–435.
20. S. Yoo, M. Harman, **Using hybrid algorithm for pareto efficient multiobjectivetest suite minimisation**, Journal of Systems and Software 83 (4) (2010) 689–701
21. T. Y. Chen, M. F. Lau, **Dividing strategies for the optimization of a test suite**, Information Processing Letters 60 (3) (1996) 135–141.
22. S. Yoo, M. Harman**, Pareto efficient multi-objective test case selection**, in: Proceedings of the 2007 international symposium on Software testing and analysis, 2007, pp. 140–150.
23. J. J. Durillo, A. J. Nebro, jmetal: **A java framework for multi-objective optimization**, Advances in Engineering Software 42 (10) (2011) 760–771.
24. W. J. Conover, W. J. Conover, **Practical nonparametric statistics** (1980).