MuKEA-TCP: A Mutant Kill-based Local Search Augmented Evolutionary Algorithm Approach for Test Case Prioritization

Ekincan Ufuktepe
University of Missouri - Columbia
Columbia, MO, USA
euh46@missouri.edu

Deniz Kavzak Ufuktepe
University of Missouri - Columbia
Columbia, MO, USA
deniz.kavzakufuktepe@mail.missouri.edu

Korhan Karabulut
Yasar University
Izmir, Turkey
korhan.karabulut@yasar.edu.tr

***Abstract*—The test case prioritization (TCP) problem is defined as determining an execution order of test cases so that important tests are executed early. Different metrics have been proposed to measure importance of test cases. While coverage and fault-detection based measures have benefits and have been used in a lot of studies, mutation kill-based measures have emerged in TCP recently, since they have benefits addressing issues with other approaches. Moreover, in the TCP problem, finding the optimal solution has a complexity of the factorial of the number of test cases, making meta-heuristic algorithms a highly suitable approach. In this study, we propose an end-to-end pipeline for TCP, Mutation Kill-based Evolutionary Algorithm (MuKEA-TCP), which allows users to have fast and efficient TCP results from existing source code, or directly from the mutant kill report of a system, without the need for any coverage information or real faults. An evolutionary algorithm utilizing Average Percentage Mutant Killed (APMK) as the objective function augmented with a local search procedure enhancing is used in MuKEA-TCP. We performed our case study on five open-source Java projects, in which we compared the APMK values of the final TCP results of some well-known greedy algorithms, and MuKEA-TCP using different initialization methods. Our results have shown that providing additional method as an initial input to the proposed augmented evolutionary algorithm has improved the results and outperformed other methods for our case study. Findings of this study have shown that using an evolutionary algorithm augmented with local search with mutation kill-based APMK as the objective function enhances the commonly used greedy prioritization methods, with a minor execution time trade-off.**

***Index Terms*—test case prioritization, software testing, search-based software engineering, evolutionary algorithms**

I. Introduction

Software evolution is an inevitable process of the software development life-cycle, in which it is quite common that a part of it breaks down. To overcome this, regression testing is performed, which generally requires running all test cases. Running the complete test suite may take longer than the available time and budget. For instance, in an industrial project [1], [2], running the entire test suite has taken seven weeks. On the other hand, when all the tests are executed, we want to reveal the faults sooner. Therefore, we want to prioritize the test cases in a test suite. The test cases can be prioritized based on their coverage information, history of revealing previous faults, or mutation kill success.

Test case prioritization (TCP) problem aims to find an execution order of available test cases in a test suite T to maximize a selected objective function. The formal definition of TCP problem [1] is given below:

Given: *T, a test suite; PT, the set of permutations of T; g, a function from PT to the real numbers.*
Problem: *Find* $T' \in PT$ *such that:*

$$\forall T'' \in PT, g(T') \geq g(T'').$$

To solve the TCP problem with an exact solution, all the possible permutations need to be checked, which is infeasible in real projects with a huge number of test cases. Even with smaller number of test cases, the effort is not meaningful, since a sub-optimal solution is sufficient in the TCP problem, which is mostly used for efficiency in running a test suite in a common software project. Therefore, defining the TCP problem as an optimization problem maximizing a metric that measures the quality of the TCP order is appropriate. The objective function for TCP may vary for different purposes and with respect to the usable information at hand about the project and test suite. The objective of the TCP can be detecting the faults sooner, by maximizing the widely used Average Percentage Faults Detected (APFD) [1], measuring the weighted average of the percentage of faults detected. However, in real life it is impossible to know the faults that are exposed by a test before running all tests [3].

On the other hand, coverage-based objective functions such as Average Percentage Branch Coverage (APBC), Average Percentage Decision Coverage (APDC) and Average Percentage Statement Coverage (APSC) can be calculated. The issue in using coverage information is that it is not an effective measure and the effectiveness might vary based on the software system [4]. Another objective function that is used in TCP is Average Percentage Mutant Kill (APMK) [5], [6], which uses mutant kill information and gives a higher priority to high quality tests. Furthermore, studies show that using mutants in TCP is an effective measure [7], [8] and are valid substitutes for real faults [9]–[11], which has motivated our study to use APMK as an objective function to prioritize test cases.

Since the TCP problem is an optimization problem of the chosen objective function, different deterministic, heuristic and meta-heuristic optimization algorithms can be used instead of heuristic approaches. Greedy algorithms can be very effective in TCP, however, they are prone to converge to a local optimum. The evolutionary algorithm (EA) is easy to implement, fast and can be run in parallel. We provided different greedy algorithm test case orderings as an initial input to the EA. We used the cross-over and mutation operators in the EA, and supported it with a local search (LS) method to find a better local optimum or even a global optimum.

The overall diagram of the proposed TCP pipeline MuKEA-TCP is given in Fig. 1. MuKEA-TCP takes a project with its test suite as inputs. The rest of the pipeline can also be used when there is a provided mutant kill report, meaning the project and the original test suite is not a requirement for the method, when a mutant kill report is provided. Mutants are injected and mutation analysis is conducted, producing a mutant kill report, which will be used for mapping between individual test cases and individual mutants for each test case mutant map (TCMM). This map provides detailed information on the test cases regarding their mutant kill successes. Once the TCMM map is created, an initial TCP population is generated using any desired method. The initial population is given to the mutant kill-based EA augmented with LS (MuKEA), which uses APMK as the objective function. At the end of each generation, further exploitation is done on the top k test cases in the offspring. The k value is decided adaptively with respect to the TCMM created in the beginning. Finally, as the termination criteria is reached, the best TCP order is found to be used in other testing purposes, such as regression testing.

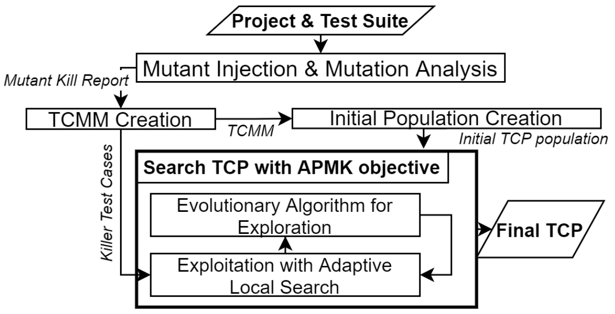
Project & Test Suite
Mutant Injection & Mutation Analysis
Mutant Kill Report
TCMM Creation
TCMM
Initial Population Creation
Initial TCP population
Killer Test Cases
Search TCP with APMK objective
Evolutionary Algorithm for Exploration
Exploitation with Adaptive Local Search
Final TCP

Fig. 1: Pipeline of proposed MuKEA-TCP.

In this study, our hypothesis is that providing a TCP order by a greedy method as an initial input into an EA augmented with a LS procedure, improves the APMK result of that greedy TCP method. Therefore, to investigate whether our hypothesis is correct, we want to answer the following research questions.

RQ1: Does using an EA augmented with LS improve the APMK results of an initially provided greedy TCP method input? We perform an empirical study on using the greedy TCP methods' output as an input to the EA with and without a LS procedure. Then, the APMK results are compared and analyzed by using a statistical test.

RQ2: Does LS augmentation improves the APMK results of an EA-based TCP method We want to investigate if an LS augmentation has a significant improvement over the EA-based TCP methods without a LS augmentation. We compare the EA-based TCP methods with and without LS augmentation and compare their APMK with a statistical test.

This paper makes the following main contributions:
(i) Method: We present an end-to-end pipeline for TCP that uses mutation kill information with a LS augmented EA. Mutant-based APMK is used as the objective function in the proposed method, which has benefits over well-known objective functions that are being used in the literature. Using APMK metric allows addressing the following issues:

(a) Fault-based objective function APFD requires prior knowledge of the faults, which cannot be known in real scenarios. Whereas, the mutants can be injected to represent faults without knowing the actual faults. If the real faults were known, the prioritization would be trivial.

(b) Coverage-based objective functions APBC, APDC, and APSC, which may not provide enough information for a test case to measure its effectiveness on revealing faults. Whereas, injecting mutants to represent possible faults of common fault classes provides good guidance on revealing fault.

(ii) Tool: We develop and make a publicly available TCP[1] tool. The tool implements the end-to-end pipeline of the TCP method we present. Furthermore, we have included the greedy TCP methods in our tools for comparison and reusability.

(iii) Dataset: We provide our mutation kill dataset to other practitioners and for better reproducibility of our study.

II. Related Work

Many studies use different meta-heuristic algorithms for TCP utilizing different objective functions.Our literature study has found that, ant colony optimization has been extensively applied in TCP. Singh et al. [12] have used ant colony optimization using APFD as an objective function. To prioritize test cases, Lu et al. [13] used a hybrid ant colony system with a sorting-based LS approach to accelerate the convergence speed. Zhang et al. [14] have also used an ant colony optimization on prioritizing test cases, based on a novel coverage-based objective function they call eAPWC (enhanced average percentage of win-Cost coverage). Bian et al. [15] has proposed another ant colony based TCP that used APSC for their objective function. In another study, the Firefly algorithm [16] is used with the APFD measurement selected as the objective function. Konsaard and Ramingwong [17] have used genetic algorithms to prioritize test cases based on code coverage.

Although meta-heuristic algorithms have been widely used in TCP, coverage-based and fault-based objectives have been used in majority, while only two of those works have used mutation-based APMK as the objective function [5], [6] to utilize mutant kill information. Using a mutant kill-based objective function addresses some of the main issues of using

[1]https://github.com/ekincanufuktepe/mukea-tcp

fault-based and coverage-based objective functions in assessment of the effectiveness of tests. The fault-based metrics need faults to calculate the fitness, which cannot be known before running the tests. In fact, if faults were known before running the test cases, there would be no need for TCP. Moreover, the coverage-based metrics tend to be unstable in revealing real faults. This work proposes an approach where the advantages of using a meta-heuristic algorithm and using a mutant kill-based metric for objective function are combined for an end-to-end TCP pipeline. A hybrid meta-heuristic algorithm is used with parallelism for both improving the optimization result and the timing performance of the proposed algorithm.

III. Methodology

A. Pre-processing and Initial Population Creation

The MuKEA-TCP pipeline takes a project with its test suite as the input. Once there is a mutant kill report, a TCMM is created. The TCMM map consists of a detailed map in between the test cases and the mutants these test cases kill. During the initial population creation, two greedy approaches that are commonly used in TCP are selected: *Total* and *Additional* [1]. We adapted these techniques for mutation kill information as *Total Mutant Kill Prioritization Technique* (TT) and *Additional Mutant Kill Prioritization Technique* (AT) respectively. The TT and AT techniques are given below:

TT: Prioritize test cases in descending order of the number of killed mutants. In other words, the test cases are sorted starting from the test case that kills the most number of mutants, to the test case that kills the least number of mutants. If there are more than one test cases that kill the same amount of mutants (a tie), the tied test cases are selected randomly and added to the prioritization order.

AT: Prioritize test cases in descending order of the number of mutants that are not killed yet (additionally killed mutants). If there is more than one test case that additionally kills the same amount of mutants (a tie), the tie is broken by randomly selecting a test case and adding to the prioritization order. After the randomly selected test case is added to the prioritized list, the mutant kill information is updated to increase the priority of other test cases that kill additional mutants that haven't been killed yet.

B. Mutant Kill-based Local Search Augmented Evolutionary Algorithm (MuKEA)

The TCP search with APMK is the last step of the proposed TCP pipeline, called MuKEA. The overall diagram of MuKEA is given in Fig. 2. An initial population consisting of permutation candidates is generated and given into the algorithm as an input. Until the termination criteria is reached, the fitness evaluation is done using mutant kill-based APMK metric for current population. The best TCP candidates are selected, with respect to the given mutation rate, a mutation operation is performed on candidates with a given probability, and with respect to given crossover rate, crossover operation is performed on randomly selected parent candidates to generate offspring. The evaluation, selection of the fittest, mutation and crossover operations continue until the given termination criteria is met. Finally, the TCP candidate with the highest APMK value is returned as the best TCP candidate.

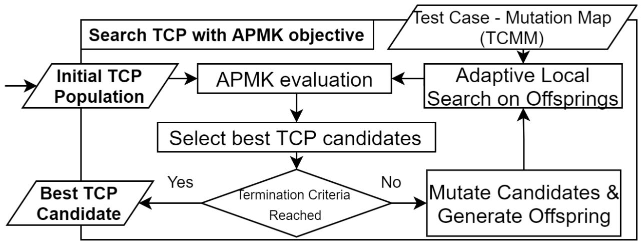
Search TCP with APMK objective
Test Case - Mutation Map (TCMM)
Initial TCP Population
APMK evaluation
Adaptive Local Search on Offsprings
Select best TCP candidates
Yes
No
Best TCP Candidate
Termination Criteria Reached
Mutate Candidates & Generate Offspring

Fig. 2: Details of MuKEA.

1) *Objective Function:*

APMK is a measure that is inspired and modified from APFD [1]. Instead of measuring how fast faults are detected, APMK measures how fast the mutants are killed by the generated ordering. Higher values of APMK indicate faster killed mutants. Formally, let T be the test suite containing n test cases, and let M be the set of k mutants revealed by T. For ordering T', let TM_i be the order of the first test case that reveals the i^{th} mutant. The APMK value of T' is calculated by the following Equation 1:

$$APMK = 1 - \frac{TM_1 + \cdots + TM_k}{nk} + \frac{1}{2n} \qquad (1)$$

2) *Local Search Augmentation:*

Whenever an offspring is generated after a crossover and mutation operation, it is not always guaranteed to have better prioritized test cases. Therefore, to increase the exploitation and the chances to have better offspring, we augmented LS into the EA after the offspring are generated. However, since LS is an expensive process, it is generally limited with a specific number of iterations. Additional information is acquired in the creation of TCMM and the initial population. It can be used to refine the LS domain and to adaptively perform the LS to improve efficiency. Since we have the information of killer and non-killer test cases, we can discard the non-killer test cases in the current TCP candidate, assuming these test cases are already at the last indexes in the current TCP order found by EA. Thereby, if there are in total 25 non-killer test cases in the TCMM, where we have a total of 100 test cases, the LS is performed on the top 75% test cases in the current TCP order.

IV. Case Study

We have performed our case study on five different open-source Java Projects that are obtained from GitHub. These projects are commonly used in software engineering related research [18] to enable controlled testing studies. Since we are not interested in the defects of the projects, we did not use the versions that are provided by Defects4J [18]. Instead, we obtained the latest versions of the projects from their original repositories. The selected projects, their versions and SLOCs (Source Lines of Codes) are given in Table I.

TABLE I: Selected Project and Mutant Information

Project name	Version	SLOC	Killed Mut.	Survived Mut.
jsoup	1.13.1	21K	2950	2131
commons-cli	1.4	7K	613	157
commons-codec	1.15	23K	2666	1374
commons-csv	1.8	7K	549	144
jackson-core	2.11.3	47K	7607	6441

A. Selection of Mutation Framework and Mutant Types

Since we selected open-source projects coded in Java for our case study, we used a mutation framework called Pitest [19] for the Java projects. It was reported that Pitest has been designed to generate stable mutations and minimize the number of equivalent mutants [20]. However, since that this study only focuses on the killed mutants and not the mutation score, equivalent mutants are not an issue for our method. On the other hand, Pitest is a state-of-the-art mutation framework, actively maintained and improved, and in addition, widely used in the research community [21]–[23]. Therefore, in this study we used Pitest (version 1.5.2) to generate mutants and obtain a mutation kill report. To generate mutants, we used the default mutator group, which includes 11 types of mutant operators. For each project, the number of generated mutants, the number of killed mutants and survived mutants are given in Table I. In our case study, we calculated the APMK based on the killed mutants and discarded the surviving mutants, since that APMK is only interested in the killed mutants.

In Table II, we show the test case information for each project, the number of killing test cases, which stands for the number of test cases that kill at least one mutant. The number of non-killing test cases is the number of test cases that does not kill any mutant. Also, the total number of test cases and the average execution time of the test suite are given.

TABLE II: Selected Project Test Information

Project name	Killing Test Cases	Non-Killing Test Cases	Test Cases	Execution Time (Sec.)
jsoup	742	13	755	41.23
commons-cli	355	0	355	0.102
commons-codec	884	75	959	43.13
commons-csv	325	4	329	4.31
jackson-core	550	1	551	7.78

B. Selection and Usage of Meta-heuristic Framework for Evolutionary Algorithm

We used Opt4J [24] to implement our MuKEA-TCP approach. In addition to EAs, Opt4J contains other meta-heuristic algorithms such as; particle swarm optimization and simulated annealing. There are several reasons we used Opt4J, which are; its support for multi-threading, including various operators, its extendable nature, and flexibility due to its dependency injection pattern usage. Furthermore, Opt4J is written in Java. Our mutation report parsing mechanism, greedy algorithm implementations, are written in Java, making it easier to integrate our implementation with Opt4J. Opt4J does not include a LS mechanism, however, it has been mentioned in [24], its *mutate*, *copy* or *diversity* operators can be used for LS. In this study, we have extended Opt4J by implementing a variation of EA augmented with a LS procedure that uses multi-threading and swap operator.

C. MuKEA Implementation Details

In our MuKEA implementation, we use a permutation of test IDs as the solution representation. The index of a test case in this permutation is the order of execution for that test. We use a standard EA that uses crossover and mutation operators to generate offspring. We apply LS to each offspring after crossover and mutation operators. The EA and operator parameters are as follows: Alpha is 100, Mu is 25, Lambda is 25 with 100 generations. Crossover Operator is Bucket with rate of 0.75, Mutation operator is Insert with Adaptive rate starting at 0.30, and the Number of threads is 12.

To augment the LS approach to our EA-based TCP approach, we extended the Opt4J framework, by adding classes/modules for our MuKEA approach. Since LS is an expensive approach, we fixed our LS iteration count to 20, which can be modified. However, it is important to recall that the LS is used on every offspring. At every iteration, 25 offspring are generated. These factors increase the execution cost, therefore, we used a multi-threaded approach for each offspring to reduce the cost. The LS cost can also be reduced by decreasing the offspring (Lambda) or LS iteration size.

D. Compared Techniques and Evaluation

We compared our proposed method with random TCP and two well-known greedy approaches that are given in Section III-A, TT and AT. To calculate the lower bound for each project's TCP result, we run 50 ordered prioritized test suites (RND). Each execution's APMK are calculated and the mean of 50 APMK is set as the lower bound to determine the significance of the TCP methods. To find the best setup for the EA-based TCP, we perform an empirical study by setting up six different variations of the EA-based TCP methods. For a population of n TCP order candidates:

EA-RND: Init. EA with n random order candidates.
EA-TT: Init. EA with population of 1 TT order candidate and $n-1$ random order candidates.
EA-AT: Init. EA with population of 1 AT order candidate and $n-1$ random order candidates.
EA-RND-LS: Init. MuKEA with n random order candidates.
EA-TT-LS: Init. MuKEA with population of 1 TT order candidate and $n-1$ random order candidates.
EA-AT-LS: Init. MuKEA with population of 1 AT order candidate and $n-1$ random order candidates.

Finally, we run each TCP method 50 times and calculate the mean of APMK. Based on the APMK results we perform a statistical test to find if there is a significant difference between the TCP methods.

V. Results and Discussion

The results of our case study is given in Table III, with a Random approach (RND) as the lower boundary, two greedy

TABLE III: Mean of APMK Results for each Project

jsoup				commons-cli				commons-codec			
Tech.	**Mean**	**Rank**	**Time**	**Tech.**	**Mean**	**Rank**	**Time**	**Tech.**	**Mean**	**Rank**	**Time**
EA-AT-LS	96.4951	A	36.0552	EA-AT-LS	96.92	A	3.3931	EA-AT-LS	96.9017	A	39.9116
AT	96.4882	A	0.0523	EA-AT	96.8874	A	0.1610	EA-AT	96.8968	A	0.7459
EA-AT	96.4867	A	0.6409	AT	96.8819	A	0.0067	AT	96.8962	A	0.0493
EA-TT-LS	95.9204	B	41.7850	EA-TT-LS	96.6641	A	3.9633	EA-TT-LS	96.1197	B	56.0926
EA-RND-LS	95.8049	B	46.4945	EA-RND-LS	96.6455	A	3.8616	EA-RND-LS	95.9501	B	65.5146
EA-TT	90.7947	C	0.96	EA-RND	93.2338	B	0.1941	EA-TT	90.8819	C	1.1234
EA-RND	89.8979	D	1.0501	EA-TT	93.0070	B	0.1914	EA-RND	88.7861	D	1.8879
TT	87.7951	E	0.0226	RND	82.8897	C	-	TT	84.7481	E	0.015
RND	81.6273	F	-	TT	81.6634	D	0.0031	RND	76.4857	F	-

commons-csv				jackson-core			
Tech.	**Mean**	**Rank**	**Time**	**Tech.**	**Mean**	**Rank**	**Time**
EA-AT-LS	97.5538	A	2.7535	EA-AT-LS	93.3747	A	116.9351
EA-AT	97.5256	A	0.1440	AT	93.3637	A	0.1442
AT	97.5181	A	0.0041	EA-AT	93.3633	A	2.2564
EA-TT-LS	97.3915	A	2.8201	EA-TT-LS	92.8069	B	116.8809
EA-RND-LS	97.3593	A	2.9067	EA-RND-LS	92.7162	B	126.1468
EA-TT	94.2669	B	1.1724	EA-TT	87.4731	C	3.0605
EA-RND	93.8199	B	0.1676	EA-RND	84.8409	D	3.4552
TT	87.2646	C	0.0022	TT	84.2613	E	0.0153
RND	80.3624	D	-	RND	72.9723	F	-

approaches (TT, AT) and six variations of the EA. AT performs better than RND, AT-based methods perform better than those that are based on TT, and EA utilizing LS is the best among the AT-based methods. Moreover, adding LS improves the results of all the other methods and using EA can improve the results of TT-based methods significantly. However, performance improvement of EA is small for AT-based methods, since AT by itself achieves very high APMK values.

To answer the RQ1 and RQ2, we performed a one-way ANOVA on the APMK values with a significance level 0.05, followed by a Tukey HSD posthoc test for grouping the TCP methods. We ranked all TCP methods, from highest A to lowest F, provided in Table III with the TCP APMK mean results. EA-AT-LS has outperformed every TCP method. However, there is no statistical significance between the top three TCP methods: EA-AT-LS, EA-AT and AT, and in some projects there are no significant difference with EA-TT-LS and EA-RND-LS as well. It is important to acknowledge that the AT approach is a powerful method, and achieves high APMK results that are close to the global optimum. Yet, the randomness in AT's algorithm can prevent achieving the global optimum. Therefore, we used an EA with and without a LS procedure to find a better APMK. We have performed our case study with same AT initial inputs in EA-AT and EA-AT-LS. For EA-AT, we observed some improvements on some AT inputs. On the other hand, when we provided the same AT inputs to EA-AT-LS, we observed an improvement in all of our 50 executions on every project, which gave us the confidence to say EA-AT-LS did improve the APMK results.

To answer RQ2 we used other LS augmented EA TCP methods, EA-TT-LS and EA-RND-LS, to observe whether they have improved the APMK results or not. In Table III, we see there is a significant difference between EA-TT-LS and TT, also between EA-RND-LS and RND. This shows that augmenting LS into EA has improved the TCP methods TT, RND, and also EA-TT and EA-RND. However, since we have found that there is no significant difference between EA-AT-LS and AT, we performed an empirical study where we preserve the first 5%, 10% and 20% order of AT and shuffle the remaining order of AT, performing 50 executions for each. Our finding about AT and the projects, which is given in Table IV show that there are super killer test cases that kill most of the mutants. Therefore, there is not much of an improvement left to be made for the EA-AT-LS approach, so the LS search space can be easily reduced with an adaptive LS space. Therefore, we have used the adaptive LS that uses the number of non-mutant killing test cases and subtracts it from the size of the test suite. In our future studies, we plan to extend the number of projects for our case study to investigate the influences of super killer test cases on EAs and AT.

TABLE IV: Mean of APMK for 5%, 10%, 20% and 100% Preserved Initial Orders of AT for Each Project

	AT			
Project Name	**5%**	**10%**	**20%**	**100%**
jsoup	91.0855	93.1927	95.2729	96.4892
commons-cli	90.5772	93.9297	96.6262	96.8753
commons-codec	91.2480	94.2599	96.2225	96.8963
commons-csv	93.0221	95.4922	97.5127	97.5166
jackson-core	84.5319	88.2042	91.1852	93.3633

In Table III, the mean of execution times in seconds of each TCP method are given for each project. All of the experiments are carried out on an i7-10750H CPU 2.59GHz processor with 16GB RAM computer. The execution time increases can be seen in the LS augmented TCP methods in Table III. However, due to the multi-threading LS implementation, the execution times are reduced to an acceptable level, which is a minimum of 2.7535 seconds and a maximum of 126.1468 seconds (approximately two minutes) among the projects.

A threat to validity is that our evaluations are only based on 5 Java projects, by comparing on 9 approaches (2 greedy approaches, 1 random, 6 different variations of EA-based approaches). The evaluations we have made may not be representative, causing our results not to be generalized. Our approach might behave differently on different programs and design, such as ones from different application domains or those that are not written in Java.

Another threat to validity is that the greedy approaches and the other TCP methods we have proposed have ignored that there might be dependent test cases in the selected Java projects. Therefore, each TCP approach that we have evaluated in this study might have generated infeasible test cases execution order, which can cause flaky tests [25]. Therefore, in future we plan to find the dependent test cases and discard the possibility to generating infeasible TCP orders.

VI. Conclusion and Future Work

In this study we have proposed a LS augmented EA approach to prioritize test cases. Our approach has used APMK as the objective function, which measures how fast mutants are killed in a software. The LS augments the EA to enhance the offspring (new ordered test cases) after the crossover and mutation operators.

In total, we proposed 6 different EA-based TCP methods, by providing three different initial inputs. We provided 3 types of initial inputs; random, order of TT, and order of AT. Then, we used these 3 initial inputs on EA with and without augmented LS. Furthermore, we have compared EA-based TCP approaches with 2 well-known greedy approaches (*TT* and *AT*), and random to determine the lower boundary. We have compared these TCP methods on 5 different Java projects, to see if LS improves the APMK score.

Our results have shown that EA-AT-LS (LS augmented EA with AT order provided), has outperformed every TCP method. Nevertheless, we have also observed that augmenting LS in EA provided with greedy approach TCP orders as inputs has been significantly improved. These improvements can be seen on the TT and randomly generated orders trivially. However, the improvements on AT orders, were very small. We have observed that, even with a small preserved amount of AT order, the APMK score wasn't remarkably different than the fully preserved AT order. This has shown us that there are test cases that kills too many mutants in the program, which does not give too many options for the augmented LS EA to increase the APMK score. As future work, we plan to use a multi-objective approach to prioritize test cases by taking into account of dependent test cases.

References

[1] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Trans. Softw. Eng.*, vol. 27, no. 10, pp. 929–948, 2001.

[2] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Trans. Softw. Eng.*, vol. 28, no. 2, pp. 159–182, 2002.

[3] Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Trans. on Softw. Eng.*, vol. 33, no. 4, pp. 225–237, 2007.

[4] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Int. Conf. on Softw. Eng.*, pp. 435–445, 2014.

[5] L. Gonzalez-Hernandez, B. Lindström, J. Offutt, S. F. Andler, P. Potena, and M. Bohlin, "Using mutant stubbornness to create minimal and prioritized test sets," in *IEEE Int. Conf. on Softw. Quality, Rel. and Security*, pp. 446–457, 2018.

[6] D. Shin, S. Yoo, M. Papadakis, and D.-H. Bae, "Empirical evaluation of mutation-based test case prioritization techniques," *Softw. Testing, Verification and Rel.*, vol. 29, no. 1-2, p. e1695, 2019.

[7] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?," in *ACM Int. Conf. on Softw. Eng.*, pp. 402–411, 2005.

[8] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using mutation analysis for assessing and comparing testing coverage criteria," *IEEE Trans. Softw. Eng.*, vol. 32, no. 8, pp. 608–624, 2006.

[9] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser, "Are mutants a valid substitute for real faults in software testing?," in *ACM SIGSOFT Int. Symp. on Found. of Softw. Eng.*, pp. 654–665, 2014.

[10] R. Gopinath, C. Jensen, and A. Groce, "Mutations: How close are they to real faults?," in *IEEE Int. Symp. on Softw. Rel. Eng.*, pp. 189–200, 2014.

[11] M. Papadakis, D. Shin, S. Yoo, and D.-H. Bae, "Are mutation scores correlated with real fault detection? a large scale empirical study on the relationship between mutants and real faults," in *IEEE/ACM Int. Conf. on Softw. Eng.*, pp. 537–548, 2018.

[12] Y. Singh, A. Kaur, and B. Suri, "Test case prioritization using ant colony optimization," *ACM SIGSOFT Softw. Eng. Notes*, vol. 35, no. 4, pp. 1–7, 2010.

[13] C. Lu, J. Zhong, Y. Xue, L. Feng, and J. Zhang, "Ant colony system with sorting-based local search for coverage-based test case prioritization," *IEEE Trans. on Rel.*, 2019.

[14] W. Zhang, Y. Qi, X. Zhang, B. Wei, M. Zhang, and Z. Dou, "On test case prioritization using ant colony optimization algorithm," in *IEEE Int. Conf. on High Perform. Comput. and Commun.;IEEE Int. Conf. on Smart City;IEEE Int. Conf. on Data Sci. and Syst.*, pp. 2767–2773, 2019.

[15] Y. Bian, Z. Li, R. Zhao, and D. Gong, "Epistasis based aco for regression test case prioritization," *IEEE Trans. on Emerging Topics in Comput. Intell.*, vol. 1, no. 3, pp. 213–223, 2017.

[16] M. Khatibsyarbini, M. A. Isa, D. N. Jawawi, H. N. A. Hamed, and M. D. M. Suffian, "Test case prioritization using firefly algorithm for software testing," *IEEE Access*, vol. 7, pp. 132360–132373, 2019.

[17] P. Konsaard and L. Ramingwong, "Total coverage based regression test case prioritization using genetic algorithm," in *IEEE Int. Conf. on Elect. Eng./Electron., Computer, Telecommun. and Inf. Technol.*, pp. 1–6, 2015.

[18] R. Just, D. Jalali, and M. D. Ernst, "Defects4j: A database of existing faults to enable controlled testing studies for java programs," in *ACM SIGSOFT Int. Symp. on Softw. Testing and Anal.*, pp. 437–440, 2014.

[19] H. Coles, T. Laurent, C. Henard, M. Papadakis, and A. Ventresque, "Pit: a practical mutation testing tool for java," in *Int. Symp. on Softw. Testing and Anal.*, pp. 449–452, 2016.

[20] T. Laurent, M. Papadakis, M. Kintis, C. Henard, Y. Le Traon, and A. Ventresque, "Assessing and improving the mutation testing practice of PIT," in *IEEE Int. Conf. on Softw. Testing, Verification and Validation*, pp. 430–435, 2017.

[21] Q. Luo, K. Moran, D. Poshyvanyk, and M. Di Penta, "Assessing test case prioritization on real faults and mutants," in *IEEE Int. Conf. on Softw. Maintenance and Evolution*, pp. 240–251, 2018.

[22] Q. Luo, K. Moran, L. Zhang, and D. Poshyvanyk, "How do static and dynamic test case prioritization techniques perform on modern software systems? an extensive study on github projects," *IEEE Trans. Softw. Eng.*, vol. 45, no. 11, pp. 1054–1080, 2018.

[23] J. Chen, Y. Lou, L. Zhang, J. Zhou, X. Wang, D. Hao, and L. Zhang, "Optimizing test prioritization via test distribution analysis," in *ACM Joint Meeting on European Softw. Eng. Conf. and Symp. on the Found. of Softw. Eng.*, pp. 656–667, 2018.

[24] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4j: a modular framework for meta-heuristic optimization," in *Annual Conf. on Genetic and Evolutionary Computation*, pp. 1723–1730, 2011.

[25] W. Lam, A. Shi, R. Oei, S. Zhang, M. D. Ernst, and T. Xie, "Dependent-test-aware regression testing techniques," in *ACM SIGSOFT Int. Symp. on Softw. Testing and Anal.*, pp. 298–311, 2020.