OCL-Based Test Case Optimisation with Modified APFD Metric

Kunxiang Jin[1] and Kevin Lano[1]

[1]King's College London Department of Informatics

March 08, 2024

Abstract

Testing is one of the most time-consuming and unpredictable processes within the software development life cycle. As a result, many Test Case Optimisation (TCO) techniques have been proposed to make this process more scalable. Object Constraint Language (OCL) was initially introduced as a constraint language to provide additional details to UML models. However, as OCL continues to evolve, an increasing number of systems are being expressed by this language. Despite this growth, a noticeable research gap exists for the testing of systems whose specifications are expressed in OCL. In our previous work, we verified the effectiveness and efficiency of performing the Test Case Prioritisation (TCP) process for these systems. In this study, we extend our previous work by integrating the Test Case Minimisation (TCM) process to determine whether TCM can also benefit the testing process under the context of OCL. The evaluation of TCO approaches often relies on well-established metrics such as the Average Percentage of Fault Detection (APFD). However, the suitability of APFD for Model-Based Testing (MBT) is not ideal. This paper addresses this limitation by proposing a modification to the APFD metric to enhance its viability for MBT scenarios. We conducted four case studies to evaluate the feasibility of integrating the TCM and TCP processes in our proposed approach. In these studies, we applied the multi-objective optimisation algorithm NSGA-II and the genetic algorithm independently to the TCM and TCP processes. The objective was to assess the effectiveness and efficiency of combining TCM and TCP in enhancing the testing phase. Through experimental analysis, the results highlight the benefits of integrating TCM and TCP in the context of OCL-based testing, providing valuable insights for practitioners and researchers aiming to optimise their testing efforts. Specifically, the main contributions of this work include: **1).** We introduce the integration of the TCM process into the TCO process for systems expressed by OCL. This integration benefits the testing process further by reducing redundant test cases while ensuring sufficient coverage. **2).** We comprehensively analyse the limitations associated with the commonly used metric, APFD, and then a modified version of the APFD metric has been proposed to overcome these weaknesses. **3).** We systematically evaluate the effectiveness and efficiency of OCL-based TCO processes on four real-world case studies with different complexities.

OCL-Based Test Case Optimisation with Modified APFD Metric

Kunxiang Jin and Kevin Lano†

*Department of Informatics, King's College London, London, United Kingdom.

*Corresponding author(s). E-mail(s): kunxiang.jin@kcl.ac.uk;
Contributing authors: kevin.lano@kcl.ac.uk;
†These authors contributed equally to this work.

Abstract

Testing is one of the most time-consuming and unpredictable processes within the software development life cycle. As a result, many Test Case Optimisation (TCO) techniques have been proposed to make this process more scalable. Object Constraint Language (OCL) was initially introduced as a constraint language to provide additional details to UML models. However, as OCL continues to evolve, an increasing number of systems are being expressed by this language. Despite this growth, a noticeable research gap exists for the testing of systems whose specifications are expressed in OCL. In our previous work, we verified the effectiveness and efficiency of performing the Test Case Prioritisation (TCP) process for these systems. In this study, we extend our previous work by integrating the Test Case Minimisation (TCM) process to determine whether TCM can also benefit the testing process under the context of OCL. The evaluation of TCO approaches often relies on well-established metrics such as the Average Percentage of Fault Detection (APFD). However, the suitability of APFD for Model-Based Testing (MBT) is not ideal. This paper addresses this limitation by proposing a modification to the APFD metric to enhance its viability for MBT scenarios. We conducted four case studies to evaluate the feasibility of integrating the TCM and TCP processes in our proposed approach. In these studies, we applied the multi-objective optimisation algorithm NSGA-II and the genetic algorithm independently to the TCM and TCP processes. The objective was to assess the effectiveness and efficiency of combining TCM and TCP in enhancing the testing phase. Through experimental analysis, the results highlight the benefits of integrating

TCM and TCP in the context of OCL-based testing, providing valuable insights for practitioners and researchers aiming to optimise their testing efforts. Specifically, the main contributions of this work include: **1).** We introduce the integration of the TCM process into the TCO process for systems expressed by OCL. This integration benefits the testing process further by reducing redundant test cases while ensuring sufficient coverage. **2).** We comprehensively analyse the limitations associated with the commonly used metric, APFD, and then a modified version of the APFD metric has been proposed to overcome these weaknesses. **3).** We systematically evaluate the effectiveness and efficiency of OCL-based TCO processes on four real-world case studies with different complexities.

Keywords: Object Constraint Language, Test Case Optimisation, Genetic Algorithm, Modified APFD Metric

1 Introduction

Software testing aims to validate the correctness and performance of the software system and determine whether the system specification conforms to the pre-set requirements [1]. As the size and complexity of the system increase, testing becomes one of the most labour-intensive and unpredictable processes within the software development life cycle. Typically, more than 50% of the total development time is spent on testing [2], so optimising this process is necessary and meaningful.

There are three common Test Case Optimisation (TCO) strategies employed to enhance the efficiency of the testing process, namely, Test Case Minimisation (TCM), Test Case Prioritisation (TCP), and Test Case Selection (TCS). TCM is an approach designed to accelerate the testing process by eliminating redundant and irrelevant test cases. TCP focuses on early defect detection through systematically re-ordering test cases. Meanwhile, TCS involves classifying test cases and determining which subsets are required for re-execution subsequently. It is important to note that the fundamental aim of these techniques is to benefit the system tester by reducing the effort invested in the testing process [3].

Despite the extensive research conducted on TCO techniques throughout recent decades, a substantial proportion of these studies are based on source code or system level [4]. Compared to the code-based approaches, fewer works are performed within the context of Model-Based Testing (MBT) scenarios, especially for Object Constraint Language (OCL) [5].

The MBT process has its natural advantage that it can be conducted before the actual systems implementation phase. Moreover, expressing system specifications at the model level will be language or platform independent, enabling the optimisation result to be used for all implementations of the corresponding OCL specifications.

Initially, OCL has been proposed as a constraint language to add more details to Unified Modeling Language (UML) model [6], but alongside the development of OCL itself, there are more and more systems that are expressed by OCL. Although OCL is not as sophisticated as other 3GLs[1], many researchers and practitioners are contributing to the community [7–9]. Therefore, exploring the different aspects of OCL-based techniques is necessary.

Mutation testing is a crucial technique that guides or acts as the objective for TCO processes. In our previous work [10], we established a full set of mutation operators applicable to the OCL standard library. These operators can thus be effectively utilised to facilitate the TCO processes for the systems whose specifications are expressed by OCL.

Our earlier research [11] demonstrated the effectiveness and efficiency of applying the TCP process to system specifications expressed in OCL. In this study, we build upon this foundation by incorporating the TCM process to discern whether its integration will bring additional benefits to the testing process. Further, to verify the potential benefits of integrating the TCM process, four case studies have been applied. Given that TCM represents a multi-objective optimisation problem, we adopt the NSGA-II algorithm to manage the minimisation process. A more detailed discussion of the proposed approach will be provided in *Section 4*.

The Average Percentage of Fault Detection (APFD) serves as a crucial metric for evaluating TCO processes, especially in the context of TCP. The APFD assesses how fast the test cases detect system defects. However, the classic APFD metric, predicated on the assumption that all defects within the system are identifiable, is not entirely suitable for MBT and mutation testing scenarios. Since OCL-based TCO techniques tailor these two scenarios together, this needs a revision of the original APFD metric. Further explanations of the modified APFD are demonstrated in *Section 4*.

This paper is organised as follows. Section 2 demonstrates the basic background, while related studies are discussed in Section 3. The details of TCM and TCP processes applicable to systems expressed in OCL and the overall idea of the modified APFD metric are explained in Section 4. Section 5 introduces the research questions, outlines the experimental methodology, presents the results, and proposes subsequent discussions. Possible threats to the validity of this research are mentioned in Section 6, while the final section concludes the paper and presents future works.

2 Background

The OCL was initially designed as a constraint language for the UML standard, but its application field has been substantially extended over time. Since OCL can be applied to many Model-Driven Engineering (MDE) activities, such as model transformation and specification requirements, OCL already became

[1] Third Generation Language

a key component of MDE. Several versions of the OCL standard have been released to adopt this language into various MDE application domains, and the current version of OCL is 2.4 [5].

OCL is a type of declarative language based on set theory and predicate logic and without any side effect on the system state [12]. Since OCL operates on the same abstract level as the system model, it does not rely on any specific implementation language. This inherent feature allows OCL to play a crucial role in diverse MDE activities, including the generation of test cases and code.

In the current study, we undertake TCO processes for systems whose specifications are expressed in OCL. OCL is employed to depict the system specifications via pre- and post-conditions, as demonstrated in *Fig. 1*. The operation modelled in this figure is used to calculate Macaulay duration within a Bond system. While there have been efforts to extend the OCL standard library, we focus exclusively on the standard operators since not all existing specifications or tools support the new operators. The operators within the OCL standard library are fully supported by our in-house tool, AgileUML [13].

AgileUML offers a variety of services for MDE processes, such as system modelling, test case generation, and code generation. When the input specification is OCL, AgileUML facilitates the generation of executable and abstract test cases directly from the OCL specification. Moreover, AgileUML also can perform mutation testing on OCL specifications, enabling the construction of mutated OCL specifications and the generation of the corresponding executable JAVA code. Lano, through a case study, demonstrated the primary services offered by AgileUML [14].

```
operation macaulayDuration(r : double ) : double
pre: r > -1 & r <= 1
post: upper = ( term * frequency )->floor()->oclAsType(int) &
c = coupon / frequency & period = 1.0 / frequency &
result = ( Integer.subrange(1,upper)->collect( i |
self.timeDiscount(c,r,i * period) )->sum() +
self.timeDiscount(100,r,term) ) / self.value(r);
```

Fig. 1 OCL Example

The TCO techniques deployed in this study are TCM and TCP. We chose to exclude TCS from this research, considering that the selection process only preserves a subset of test cases for re-execution, and some of the test cases might be eliminated during the selection process. Nevertheless, the discarded test cases might be useful in future testing periods. Although TCM also results in removing some test cases, it distinguishes itself from TCS in that the minimisation process only eliminates redundant test cases [15]. We provide the definitions for these problems as follows.

The TCM problem aims to reduce the size of the test suite by eliminating redundant and unnecessary test cases to improve the testing ability. The TCM problem can formally be defined as [3]:

Given: A test suite, T, a set of test requirements r_1,...,r_n, that must be satisfied to provide the desired 'adequate' testing of the program, and subsets of T, T_1 , . . . , T_n , one associated with each of the r_is such that any one of the test cases t_j belonging to T_i can be used to achieve requirement r_i.

Problem: Find a representative set, T', of test cases from T that satisfies all r_is.

When each test requirement in r_1,...,r_n is satisfied by at least one of the test cases, the testing criteria will be considered as satisfied. The newly formed test suite, T' is the representative set of test cases selected from T_is. Moreover, to maximise the effectiveness of the TCM process, the T' should be the minimal representative set of T_is.

A test suite is composed of all test cases that intend to fulfil pre-defined testing requirements. As the size and complexity of the software grow, the test suite correspondingly expands. Running all the test cases within this increasingly large test suite can become impracticable and inflate the testing budget. The technique of TCM produces a representative subset from the original test suite that attempts to meet all the requirements covered by the original suite with a reduced number of test cases. A test case is redundant when other test cases can fulfil the same requirements. Consequently, removing such a test case does not decrease the fault detection capacity.

The TCP problem aims to reveal the defects as quickly as possible by reordering the test case sequence, even if the testing procedure is prematurely halted. The TCP problem can be defined as [16]:

Given: T, a test suite, PT, the set of permutations of T, and f, a function from PT to the real numbers.

Problem: Find $T' \in PT$, such that $(\forall T'')\ (T'' \in PT)\ (T'' \neq T')\ [f(T') \geq f(T'')]$.

And in this definition, PT is all possible combinations of prioritisation of set T. Moreover, f is a function that returns a reward value, such as APFD value, to a specific ordering combination.

From the above definition, to find the potential best solution T', the prioritisation algorithm must define the set of every permutation PT of test cases. Consequently, choose the T', which can maximise the function f.

Analysing each combination of test cases is practically unfeasible, especially when dealing with a large test suite. Assuming a test suite encompasses n test cases, the size of PT will be $n!$. This circumstance might re-frame the TCP problem into a variant of the Traveling Salesman Problem, a well-known NP-complete problem [17]. Therefore, the TCP problem is usually guided by a heuristic algorithm, such as a genetic or ant colony algorithm. These algorithms employ their powerful search capabilities to construct the sequence of

test cases. In this research, we utilise the genetic algorithm to guide the prioritisation process [18]. Simultaneously, we use the NSGA-II[2] to navigate the execution of the TCM process [19].

Within the scope of MBT, elements such as system defects and actual implementations are unavailable for TCO processes. In order to overcome this situation, mutation testing often acts as a guiding mechanism for TCO procedures. Mutation testing is a fault-based testing technique that has been studied for over 50 years since it can be traced back to 1971 [20].

In mutation testing, from a program or specification p, a set of faulty versions p' called mutants are generated by making, for each p', a single simple change to the original artefact p. Mutation operators are the transition rules defining how to perform these changes and represent the mutants. Mutated versions of expressions are syntactically and type-correct, but should have a distinct semantics from their source.

A principal application of mutation testing involves the calculation of the *mutation score (MS)* for test cases. This score represents the proportion of mutants that have been successfully killed or detected, providing a measure of the effectiveness of the testing process [21]. The formal mathematical definition of the mutation score is provided subsequently.

$$MS = \frac{K_m}{T_m - E_m} \tag{1}$$

In this equation, K_m refers to the number of killed mutants, T_m is the number of generated mutants and E_m is the number of equivalent mutants. A mutant p' will be considered as killed when a different result compared to the original program p has been returned. The range of mutation scores is from 0 to 1, and a higher mutation score means higher confidence in the corresponding test cases.

In the field of MBT, scholars typically employ mutation testing to verify the effectiveness of proposed methodologies or as a guide for optimisation processes. With respect to TCO processes, mutants can simulate system defects or requirements, or alternatively, mutation scores can be utilised as an evaluative measure of the capacity for fault detection.

3 Related Works

Owing to the limited research specifically addressing TCO processes for the systems whose specifications are expressed in OCL, the related works section primarily focuses on the general aspects of TCM and TCP processes. Studies focusing solely on mutation testing have been omitted from this review, as this technique is employed in this study as a methodological tool rather than directly being the subject of investigation.

[2] Non-dominated Sorting Genetic Algorithm II

3.1 Test Case Minimisation

In [22], Tallam proposed a delayed greedy strategy for the TCM process. One potential weakness of the greedy strategy is that the early selected test cases may eventually be made redundant by subsequently selected test cases. The authors overcame this weakness by constructing a concept lattice, a hierarchical clustering that recorded the relationship between test cases and requirements. The experiment result showed that the delayed greedy strategy could select fewer test cases than the greedy strategy.

Palomo [23] used an exact search-based technique to perform the minimisation process and maintain the mutation coverage meanwhile. The approach mutated the original specifications and compared their behaviours versus the original ones for the test suite. If the mutants cannot be killed, the corresponding test cases will classify the test case as redundant.

A genetic algorithm approach is demonstrated in [24] by Bhatia. The proposed algorithm aims to provide optimal fitness value by combining the genetic algorithm and class partition. The modified GA minimises the number of test cases by finding the most error-prone test cases based on the priority of the path information. Although path analysis is an effective technique, its application in the context of MBT may still be considerably challenging.

Lin [25] proposed an approach to minimise test suite for composition service by modification impact analysis. They compared structural and variable changes and located the impacted nodes by dependency analysis. Through the influenced information, they excluded redundant test cases to perform the TCM process.

Hashim [26] proposed a TCM approach based on the firefly algorithm. The optimisation process is combined with the UML state machine diagram, by analysing the path coverage for the state machine to calculate the fitness value for each test case. Then, through a firefly algorithm to minimise the test suite. Although this study aligns directly with the MBT domain, there are differences between the state machine diagram, which models the behavioural aspect of the system, and the OCL, which models the structural aspect of the system.

Deneke [27] also proposed a TCM approach by using an evolutionary algorithm, which is based on particle swarm optimisation. The proposed approach uses requirements coverage information to guide the particle swarm optimisation process. They validate the effectiveness of their method by comparing various benchmark techniques. In contrast to this study, our approach employs the NSGA-II algorithm to lead the minimisation process. Moreover, we substitute the requirement coverage with the use of mutants for optimisation.

Li [28] proposed a mutation-based TCM approach. Based on the fault detection information of each test case, the hierarchical clustering process is conducted. Then the test case with the highest priority in each cluster is selected according to the cut-level threshold. If there exists any edge test or

edge mutation program, the corresponding one will be chosen. The edge mutation program is the faulty mutation version that can only be killed by one specific test case, and that test case is the edge test.

Bajaj [29] suggested an improved quantum-behaved particle swarm optimisation for TCO processes, which include the TCM process. The proposed approach has been validated against various evolutionary algorithms. They used fault coverage or statement coverage as the fitness function to guide the optimisation process. The coverage information for the MBT process still needs further exploration since the actual implementations are not yet available.

3.2 Test Case Prioritisation

Rapos proposed a novel methodology that employed fuzzy logic to guide the TCP process, as reported in their study [30]. They transformed UML models into symbolic execution trees. Subsequently, they used test suite size, symbolic execution tree size, relative test case size, and output significance as inputs via 39 fuzzy rules to infer the priority. The efficacy of their proposed approach was assessed through a comparative analysis with a random strategy.

In a recent study, Shin [31] employed an alternating variable method to execute a model-based TCP process. The researchers customised a set of mutation rules that were applied to the state machine diagram, which subsequently led to the generation of test cases in accordance with the mutants. The test case prioritisation process is based on the fault coverage criteria and guided by the alternating variable method. Then the effectiveness of the proposed approach is examined by the APFD metric. Instead of performing mutation testing to state machine diagrams, we applied this technique to OCL specifications.

Pospisil [32] conducted the TCP process by employing an enhanced variant of the Adaptive Random Prioritisation (ARP) method. In contrast to the traditional ARP, they substituted the original distance function with a multi-criteria decision-making approach. Additional parameters, such as path complexity and coverage information, were incorporated in determining the priority.

Pan [33] disclosed the outcomes of a systematic literature review that focused on the application of machine learning methods to the TCP process. Gupta [4] conducted a comprehensive examination of multiple-objective and hybrid strategies addressing the TCP problem while also proposing future directions for research in the field.

Both Ma [34] and Rattan [35] leverage the genetic algorithm to navigate the TCP process. However, the methodologies in these two articles exhibit significant differences. Ma utilised control flow diagrams and path information, whereas Rattan employed an extended system dependence graph to guide the prioritisation process.

Sornkliang [36] conducted research closely related to the MDE context. In their approach, they attach the weight information to each node in the UML activity diagram. Then test path priorities are established based on the computed scores for each path.

In their research, Chaudhary [37] extensively compared the TCP capabilities of four unsupervised clustering algorithms through five distinct case studies. The DBK-means algorithm[3] was found to be superior in performance compared to the rest. Meanwhile, Morozov [38] suggested a model-based TCP methodology. This method relies on analysing fault activation and error propagation for application in automotive systems.

Sun [39] introduced an approach for the TCP process in metamorphic testing by path analysis. Their method starts by analysing feasible paths with symbolic execution, followed by generating relevant test cases via a constraint solver. These produced test cases are prioritised by computing the distances between them based on statement coverage.

4 OCL-Based TCO Processes

In this section, we will first discuss the modified APFD metric, which serves as one of the objectives within our proposed algorithms. Following this, we will detail the algorithms employed for executing OCL-based TCO processes.

4.1 Modified APFD Metric

One of the primary evaluation metrics used in TCO processes is the APFD, which measures how quickly or early the test suite can detect system defects [40]. The equation used to calculate the original APFD is as follows:

$$APFD = 1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{nm} + \frac{1}{2n} \tag{2}$$

In the presented equation, TF_i represents which test case first detects the i^{th} fault, while n indicates the total number of test cases in the test suite, and m represents the total number of faults within the software system. In the context of MDE, because system faults cannot be known in advance, the faults also can be used instead of the artificially injected defects.

The APFD metric is designed to range between 0 and 1, with a higher value indicating a more efficient fault detection ability. However, the maximum actual APFD value is $1 - \frac{1}{2n}$, which is obtained when the first test case detects all faults. On the other hand, the minimum APFD value is $\frac{1}{2n}$, which is reached when the last test case detects all faults.

This means that the actual APFD result does not traverse the full range from 0 to 1, which represents *the first shortcoming of the original APFD*. For instance, in an extreme scenario where a single test case in the suite can detect all system faults, the APFD value will only reach 0.5, which is a relatively low result but is already the maximum value (the maximum value $1 - \frac{1}{2n}$).

The second point is that the original APFD value assumes all defects are detectable. This assumption is unfeasible when performing mutation testing in an MDE environment. Mutation testing injects mutants into the system and then validates whether the test cases can kill the artificial mutants. Due to

[3] Density-Based K-means Algorithm

mutated specifications being generated based on the mutation rules, we cannot guarantee that the test suite can detect all system defects in this scenario.

One possible solution to undetectable faults is assuming that TF_i is the last position in the test suite. However, this cannot distinguish whether the test suite cannot detect the defects or the last test case detects the defect. Walcott proposed another possible solution, that when a fault is missed, the TF_i equals the number of test cases plus one [41]. This is a feasible solution to the second imperfection of the APFD metric, but in the extreme scenario that all faults can not be detected, the APFD value will be $-\frac{1}{2n}$, which does not range from 0 to 1 anymore.

The third imperfection is that even when two test orders have the same APFD value, one may still be better than the other. Considering that there are three test cases and six faults, the fault detection information is shown as *Fig. 2*.

	F1	F2	F3	F4	F5	F6
T1		✓	✓	✓	✓	
T2	✓	✓	✓			
T3				✓	✓	✓

Fig. 2 An Example of Fault Detection

There are two test orders, {1, 2, 3} and {3, 2, 1}. These two orders have the same APFD value since the $TF_1 + TF_2 \cdots + TF_m$ is the same. For the first one is $2 + 1 + 1 + 1 + 1 + 3$, and for the second one is $2 + 2 + 2 + 1 + 1 + 1$, where all of them are 9. TCP aims to detect faults as early as possible, even if the testing process is prematurely halted. From the APFD results, these two test orders have the same effectiveness. However, if the testing process is halted after executing only one test case, the first order can detect four faults, while the second order can detect three faults. In this scenario, the original APFD value may not adequately reflect the effectiveness of the TCP approach.

To summarise, the original APFD metric suffers from the following three disadvantages. Firstly, its actual value range is not ranged between 0 and 1. Secondly, the metric may not provide an accurate assessment when faults are undetectable. Finally, even when two test orders have identical APFD values, one may still be superior to the other.

While some studies, such as [42], have attempted to address the issues identified with the original APFD metric, none have specifically tackled its application within the context of MDE. To address these shortcomings, we propose introducing a reward system that increases the APFD value when a fault is detected by the first test case. Simultaneously, we suggest including a penalty system that reduces the APFD value if the fault cannot be detected by any test case.

We proposed a modified APFD metric in this work. The modified APFD value can be calculated by *Equation 3* and *Equation 4*.

$$APFD_m = 1 - \frac{TF_1+TF_2+\cdots+TF_m}{nm} + \frac{1}{2n} + (\alpha - \beta) * \lambda \quad (3)$$

$$\lambda = \frac{1}{2nm} \quad (4)$$

In this metric, the first part is the same as the original APFD metric. We called this part a reward and penalty factor in the second part $(\alpha - \beta) * \lambda$. α is the number of faults that can be detected by the first test case, and β is the number of faults that cannot be detected by any test case. λ is the reward or penalty value, which equals to $\frac{1}{2nm}$.

Using the modified APFD value, this metric ranges from 0 (all defects are undetectable) to 1 (the first test case detects all defects). Through the improvements, this metric has the reward and penalty mechanism to deal with the aforementioned situations.

4.2 Proposed Approach

In this subsection, we explain the algorithms we have proposed for executing the OCL-based TCO processes. For the TCM and TCP processes, we employ the NSGA-II and genetic algorithm, respectively. Our approach treats the TCM process as a multi-objective problem, while the TCP process is considered a single-objective problem. The followings are a detailed description of these two processes.

4.2.1 Test Case Minimisation

Algorithm 1 shows the pseudo-code of the general TCM process.

In *Algorithm 1*, the input is an OCL specification, while the output is the selected (minimised) test cases. The first two lines are the preparation works for the algorithm. When the input specification for the AgileUML tool set is OCL, the mutated OCL specification, test suite and corresponding executable JAVA programs can be generated. The generated test suite is treated as the original one, waiting to be minimised.

In *line 2*, AgileUML will analyse the original test suite and mutants to extract the necessary information, mainly focusing on which test case can detect which mutants. Despite various attempts to make OCL directly executable [43–45], when system specifications are expressed in OCL, the test suite cannot run directly against these specifications. To tackle this obstacle, we

Algorithm 1 Test Case Minimisation

Input: OCL Specification
Output: Selected Test Cases

```
1: AgileUML ← OCL Specification (mutated specifications, test suite, executable Java program are generated)
2: information ← agileUMLAnalysis(testSuite, mutants)
3: initialPopulation(testSuite)
4: until NSGA-II stop criteria do
5:     selection, crossover, mutation operators
6:     setObjective(0, selectedSize)
7:     setObjective(1, faultDetectionRate)
8:     setObjective(2, APFDMetric)
9:     fitnessEvaluation(population)
10: end until
11: solutions ← getNSGA-IISolutions()
12: result ← solutionAnaysis(solutions)
```

initially produce corresponding JAVA specifications via model-to-text transformation, which are then used to test the suite. It is worth noting that this is just one viable solution we have opted for and is not the exclusive answer.

line 3 - *line 10* are the main processes of the NSGA-II algorithm. Start with the randomly generated population, then through the evolutionary operators iteratively to solve the corresponding problem. The bit data structure is used to represent individuals, and one example is *Fig. 3*.

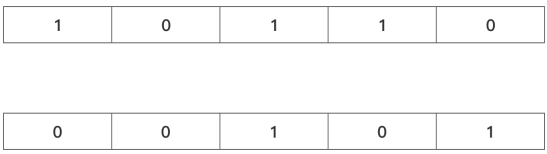

Fig. 3 Individuals in NSGA-II

In this example, the test suite has five test cases, and in these two individuals, test cases {1, 3, 4} and {3, 5} will be included in the minimised test suite separately. In the TCM problem, the length of the chromosome equals the number of test cases within the original test suite. Each bit position is used to describe whether the corresponding test case is selected for the minimised test suite, where the value of 1 indicates that the test case has been selected and 0 indicates that it has been excluded.

The selection operator is tournament selection, which randomly selects a subset of the population from the original one and then chooses the best one according to the fitness value. A larger tournament size increases the chance

of selecting the best individual but also increases the computational cost of the selection process. In this work, we set this value as 2.

The crossover operator in this work is two points crossover, which selects two crossover points in the parent solutions, and swaps the corresponding parts between these points to create new offspring solutions, which shows like *Fig. 4*. Meanwhile, the mutation operator we chose is the bit flip mutation, which flips each bit position (0 to 1 and 1 to 0) according to the mutation probability.

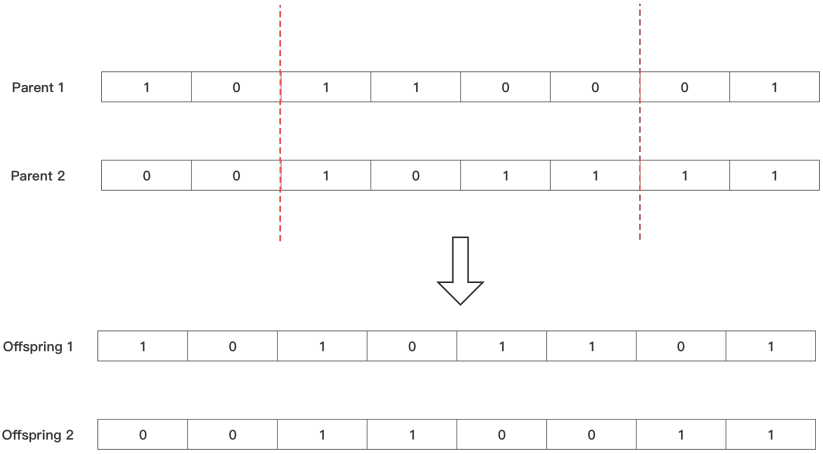

Fig. 4 Two Points Crossover

One of the essential steps within the NSGA-II algorithm is the fitness evaluation, which corresponds to the multiple objectives. The primary aim of the TCM process is to reduce the testing efforts by minimising the number of test cases, hence the first optimisation objective is the *Number of Test Cases*.

In addition, the reduction should not threaten or significantly impair the fault detection capability, leading to the second optimisation objective, the *Fault Detection Ability*. Given scenarios where actual fault detection information is lacking within the MDE or OCL context, this attribute will be replaced by the mutant detection ability.

The TCM process solely determines the selection of test cases into the minimised suite without altering their sequence. Intuitively, this process will not be related to the APFD metric. However, it is meaningful in deciding the redundant test case to eliminate. For instance, if only test case 2 and test case 5 can detect two particular faults, the optimisation process would prefer to remove test case 5 instead of test case 2. This preference is because removing test case 5 allows the faults to be detected earlier in the testing process. As such, the third objective is the *Original or Modified APFD metric*.

In short, there are three objectives, the number of test cases, fault or mutant detection ability and the original or modified APFD metric.

The NSGA-II algorithm generates a solution set known as the Pareto set. This set contains a group of solutions where none can outperform the others based on all objectives simultaneously. However, we need to select one final solution as the minimisation result. The selection criteria for this solution are based on achieving the maximum fault detection ability with the smallest size. This choice aligns with the objective of the TCM process, which aims to minimise the number of test cases while ensuring a high fault detection rate.

4.2.2 Test Case Prioritisation

As the same procedure of our previous work [11], the following is the pseudo-code of the TCP process.

Algorithm 2 Test Case Prioritisation

Input: Selected Test Cases **or** OCL Specification
Output: Prioritised Test Cases

```
1: initialPopulation(selectedTestSuite)
2: until stop criteria do
3:     selection, crossover, mutation operators
4:     setObjective(APFDMetric)
5:     fitnessEvaluation(population)
6: end until
7: result ← getGASolution()
```

The inputs for the TCP process, as shown in *Algorithm 2*, can either come from the minimised test cases from the TCM process or from the original OCL specification when conducting the TCP process exclusively. The first approach involves the merging of TCM and TCP processes, whereas the second approach runs the TCP process in isolation. In instances where the input is the OCL specification, some preparation work is needed, similar to the first two lines within *Algorithm 1*. TCP, in contrast to TCM, has a single objective. Thus, the basic genetic algorithm is employed for the optimisation.

In TCP problems, the chromosome is the permutation of the test cases, so the chromosome length is precisely the same number of test cases within the test suite, like *Fig. 5*.

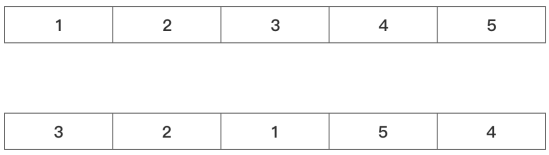

Fig. 5 Example of Individual

In this example, there are two individuals, which are {1, 2, 3, 4, 5} and {3, 2, 1, 5, 4}. The sequence of numbers is the execution order of the corresponding. The first individual will execute the first test case at the beginning, then the second, third, fourth, and fifth. However, the second individual will start with the third test case, then the second, first, fifth and finally the fourth one. Due to the TCP problem being modelled as a permutation problem, each distinct number will only appear once in the individual.

As in the original study, the crossover operator used is the Partially Mapped Crossover (PMX), the mutation operator is the permutation swap mutation, and the selection operator is binary tournament selection. However, in contrast to the original study, this research proposes a modified APFD metric. This study focuses on examining the applicability of the TCM process within the OCL-based TCO processes, and for further details on the TCP process, refer to our previous research work.

The TCM and TCP processes can be applied separately or in combination to the systems whose specifications are expressed in OCL. However, it is advisable to carry out TCM before TCP when combining these processes. The primary function of TCM is to select a subset from the original test suite, whereas TCP re-orders the test cases. By performing TCM first, the search space for TCP is decreased, which enhances overall efficiency. On the other hand, the results of the TCP process do not reduce the search space for the TCM process.

Both TCM and TCP processes consistently employ the (modified) APFD value as an optimisation objective. This metric measures how fast the corresponding test suite detects software defects. Since the OCL-based TCO processes are a model-based strategy, the actual system defects are not readily available when these optimisation activities are conducted. To address this issue, we simulated the defects artificially according to the mutation operators. The operators we used are derived from our prior work, where we proposed a full set of mutation operators for the OCL standard library [10].

4.2.3 Applied Mutation Operators

As this paper does not concentrate on the definition of mutation operators for the OCL expressions, we only present a subset of the full set of operators. The operators we focus on here are relevant to the *Collection* type within the OCL standard library. The format of the descriptions is: the original standard specifications marked with •, the mutants begin with MO_i (i indicates the number of mutation operators), and the comments are introduced with %. The detailed operators and their classifications are available in our previous work.

- **= (c: Collection(T)): Boolean**
  MO_1 : <>
- **<> (c: Collection(T)): Boolean**
  MO_1 : =
- **size(): Integer**
  MO_1 : size() - 1 MO_2 : size() + 1

- **includes(object: T): Boolean**
  MO_1 : excludes(object)
- **excludes(object: T): Boolean**
  MO_1 : includes(object)
- **count(object: T): Integer**
  MO_1 : count(object) - 1 MO_2 : count(object) + 1
- **includesAll(c2: Collection(T)): Boolean**
  MO_1 : *c2 →exists(e | self →excludes(e))*
- **excludesAll(c2: Collection(T)): Boolean**
  MO_1 : *c2 →exists(e | self →includes(e))*
- **isEmpty(): Boolean**
  MO_1 : notEmpty()
- **notEmpty(): Boolean**
  MO_1 : isEmpty()
- **max(): T**
  MO_1 : min()
- **min(): T**
  MO_1 : max()
- **sum(): T**
  NULL % no corresponding mutation operator
- **product(c2: Collection(T2)): Set(Tuple(first: T, second: T2))**
  NULL % no corresponding mutation operator
- **selectByKind(type: Classifier): Collection(T)**
  MO_1 : selectByType(type)
- **selectByType(type: Classifier): Collection(T)**
  MO_1 : selectByKind(type)

The development of these mutation operators followed specific strategies: 1) Negation of original operations. 2) The difference between OCL standard and common programming languages. For instance, differences related to collection-type elements are considered. Most common programming languages have a 0-based index, while OCL uses a 1-based index. 3) Common errors are addressed, such as misunderstandings in semantics. A prime example would be the confusion between *→including* and *→excluding*.

The design was limited to those operations that currently exist within the OCL standard library. Despite proposed additions to future OCL types or operations, like those in [46], we have not included mutation operators for these types or operations because they are not universally supported across all OCL tools.

Although these strategies have proven to be useful, they are not flawless and will likely need to be improved upon in future research. Not all operations within the standard library have corresponding mutation operators. When we cannot identify a suitable mutation operator for a particular operation based on our strategies, we choose to neglect it.

5 Evaluation

In this section, we propose three research questions to assess the feasibility of the OCL-based TCO processes. The corresponding experimental procedure is designed, and the relevant results are presented.

5.1 Research Questions

In this study, we aim to apply TCO techniques to systems whose specifications are expressed by OCL, expanding on previous work to introduce the TCM process. We propose the following research questions to evaluate our approach:

RQ 1. Effectiveness: These research questions focus on the efficacy of our proposed method.

RQ 1.1: *How effective is the test case minimisation within the context of OCL?*

RQ 1.2: *Using the modified APFD metric, how effective is the TCP process* ***without*** *applying TCM process?*

RQ 1.3: *Using the modified APFD metric, how effective is the TCP process* ***with*** *the application of TCM process?*

RQ 2. Overhead What is the overhead when applying test case optimisation to OCL specifications?

RQ 3. Difference What are the variances between the original and modified APFD metrics within the process of TCP?

The first series of research questions primarily deals with effectiveness. To answer *RQ 1.1*, we consider that the purpose of TCM is to enhance the testing process by eliminating redundant or unnecessary test cases. Therefore, the evaluation metric for this question would be the reduction rate in size and the fault detection ability for the minimised test suite. The goal of optimisation is to decrease the size of the test suite while aiming to maintain or minimally affect the fault detection capability.

In regards to *RQ 1.2* and *RQ 1.3*, the evaluation metric is the enhancement of the modified APFD value. In this question, we concentrate solely on the modified APFD value, and the improvements relative to the original APFD value will be discussed under *RQ 3*.

In response to *RQ 2*, we measured the time taken by the TCM and TCP processes. If the overhead of implementing a new methodology is excessive, its real-world application may not be justified. As a result, the time taken by the optimisation process serves as a key evaluation metric. The overhead mainly includes the time spent generating test cases and mutants, analysing fault detection data, and the time used by the NSGA-II and genetic algorithms. The optimisation process can be deemed worthwhile if the overhead is within an acceptable range. For comprehensive results, each experiment will be performed 50 times.

In *RQ 3*, we address the need to understand the impact of our adjustments to the original APFD metric. As this modified metric is primarily utilised

for evaluation within the TCP process, we focus on comparing the differences between the original and our modified APFD metrics in the TCP context.

5.2 Experimental Procedure

To answer the research question and assess our proposed method, we utilise four distinct OCL specifications collected from real-world studies: Bond, Interest Rate, MathLib and UML2PY. These studies vary in their complexity. The specifics of each case study, including their size (as measured by lines of code) and the number of operations, are demonstrated in *Table 1*. Same as *Fig. 1*, *Fig. 6* provides an example OCL specification with a single function, including the function name, pre-conditions, and post-conditions.

Table 1 Case Study Details

Study Name	Size	Operation
Bond	93	5
Interest Rate	411	2
MathLib	212	15
UML2PY	1053	18

```
operation macaulayDuration(r : double ) : double
pre: r > -1 & r <= 1
post: upper = ( term * frequency )->floor()->oclAsType(int) &
c = coupon / frequency & period = 1.0 / frequency &
result = ( Integer.subrange(1,upper)->collect( i |
self.timeDiscount(c,r,i * period) )->sum() +
self.timeDiscount(100,r,term) ) / self.value(r);
```

Fig. 6 OCL Example

The AgileUML tool generates corresponding test cases (forming the original test suite), mutated specifications, and executable JAVA files when given OCL specifications as input. For a particular function, the number of generated test cases (at a functional level) for the four case studies ranges from 4 to 243.

The mutated OCL specifications are created in accordance with the corresponding OCL expressions and invariant variables, using the detailed transition rules based on our previous work [10]. The test cases at the functional level are constructed using the boundary value of each parameter and the various combinations of these parameters.

The following details the configurations for the two optimisation algorithms. To limit the influence of parameter configurations, the same settings were applied across all four case studies.

For the NSGA-II algorithm (used for TCM), we adopted a crossover rate of 0.9 with a two-point crossover operator. The mutation rate was set to

$1/TestSize$, employing a bit-flip mutation operator. Binary tournament selection was the chosen selection operator. The population size was fixed at 50, and the maximum iteration count was set to 1000.

As for the basic genetic algorithm (employed for TCP), the crossover rate was also 0.9, but the Partially Mapped Crossover (PMX) was the crossover operator of choice. The mutation rate was defined as$1/TestSize$, and a permutation swap mutation operator was used. Like NSGA-II, we selected binary tournament as the selection operator. The population size for this algorithm was the same, set at 50, with a maximum iteration count of 1000.

During the preliminary phase of this study, an informal combinatorial experiment was conducted in which the population interval was set at 10 and the iteration interval at 100. This led us to the final configuration of 50 for the population size and 1000 for the iterations. For our chosen case studies, these settings delivered effective outcomes in a reasonable overhead.

However, it is critical to note that this study is not primarily focused on identifying the most optimal configurations for the optimisation algorithm. Hence, the parameters we chose to work with may not be the best for all circumstances. In real-world applications, these variables and the evolutionary operators can be adjusted based on specific needs and requirements.

Given the inherent randomness in both the NSGA-II and genetic algorithms, we carried out each experiment 50 times to decrease the deviations. To maintain a consistent experiment environment, all experiments were conducted on the same machine with MacOS 12.3.1, a 2GHz Quad-Core Intel Core i5 processor, and 16GB 3733 MHz LPDDR4X RAM. The algorithms were implemented in the JAVA within IntelliJ Idea IDE.

5.3 Results Analysis

As described in *Experimental Procedure*, we performed each experiment 50 times to collect the results, *Table 2 - 9* are the collected results for Bond, Interest Rate, MathLib and UML2PY case studies.

Within the Bond case study, there are five different operations: bisection, discount, macaulayDuration, timeDiscount, and value. In Table 2, these operations are represented by the codes *A1 - A5* for simplification. The second case study, Interest Rate, has nelsonseigal and ns operations, which are substituted by *B1* and *B2* in Table 3. Similarly, for the MathLib and UML2PY case studies, we replaced the detailed operation names by *C1 - C15* and *D1 - D18* within the the *Table 4 - 6* and *Table 7 - 9*. More details about the MathLib case study are available in [47].

The results of each case study are systematically presented in four parts within the tables. The initial part demonstrates the number of test cases within the original test suite, which was directly derived from the AgileUML tool set. The corresponding original and modified APFD values are computed based on the generated mutants, and these APFD values are treated as the benchmark.

The second part demonstrates the result of the TCM process, providing average values for the test suite selection rate, fault detection loss rate, and

Table 2 Bond

Name	**A1**	**A2**	**A3**	**A4**	**A5**
Test Cases	64	80	4	80	4
Original $APFD$	0.9399	0.9187	0.6376	0.9187	0.7389
Original $APFD_m$	0.9399	0.9187	0.6836	0.9187	0.8186
TCM Process					
Selection Rate	43.5937%	43.225%	50%	43.25%	50%
Detection Loss	0%	0%	0%	0%	0%
Time (ms)	3600.38	5901.3	6251.02	5786.12	6304.36
TCP Process					
$APFD$	0.9886	0.9937	0.6383	0.9935	0.7389
$APFD_m$	0.9947	0.9996	0.6843	0.9994	0.8186
Random $APFD_m$	0.8980	0.9479	0.6078	0.9444	0.7587
Time (ms)	2188.06	2169.42	7119.42	2103.9	6158.28
TCO Process					
Selection Rate	43.5937%	43.2%	50%	42.975%	50%
Detection Loss	0%	0%	0%	0%	0%
$APFD$	0.9645	0.9855	0.5913	0.9854	0.6593
$APFD_m$	0.9781	0.9986	0.6833	0.9983	0.8186
TCP Time (ms)	2037.2	1940.76	5045.76	1916.94	4506.9
Overhead	42270.96 ms				

operation duration over 50 runs. Simultaneously, the third part solely focuses on the TCP process, which includes the average results for the prioritised test suite using the original and modified APFD metrics and the time cost. A comparative analysis is conducted between the prioritised test suite and a randomly ordered one under the modified metric.

The fourth part combines the TCM and TCP processes and includes the average selection rate, fault detection loss rate, the original APFD value and modified APFD value for the optimised test suite. For the TCO process, we initially applied the TCM process on the original test suite, followed by the TCP process. We collected the time expense for the TCP process on the minimised test suite to validate whether the TCM process could benefit the TCP process in terms of time usage.

The final line captures the average total overhead, encompassing time allocations for generating test cases and mutants, processing fault detection information, and the operational time for the NSGA-II and genetic algorithms for all operations within each case study. All time usages are measured in milliseconds during the evaluation process.

Within the aforementioned tables, certain cells are denoted by "-", indicating that following the TCM process, only a single test case remains. Consequently, the TCP process becomes redundant and unnecessary in such instances.

Following are the answers to the research questions and the discussions of the experimental results.

Answers to RQ1: The set of research questions within *RQ 1* mainly validates the effectiveness of the proposed approach. The metrics for the TCM-related question are the size reduction rate and the fault detection loss after the

Table 3 Interest Rate

Name	**B1**	**B2**
Test Cases	243	243
Original $APFD$	0.9727	0.4190
Original $APFD_m$	0.9727	0.4183
TCM Process		
Selection Rate	43.786%	42.4279%
Detection Loss	0%	0%
Time (ms)	146.44	141.52
TCP Process		
$APFD$	0.9974	0.6659
$APFD_m$	0.9993	0.6666
Random $APFD_m$	0.9828	0.6111
Time (ms)	66.98	81.94
TCO Process		
Selection Rate	44.0823%	42.6666%
Detection Loss	0%	0%
$APFD$	0.9942	0.665
$APFD_m$	0.9984	0.6666
TCP Time (ms)	26.64	28.26
Overhead	327.88 ms	

Table 4 MathLib (1)

Name	**C1**	**C2**	**C3**	**C4**	**C5**
Test Cases	3	5	5	25	5
Original $APFD$	0.6111	0.9	0.6	0.32	0.6333
Original $APFD_m$	0.7222	1	0.6	0.315	0.6333
TCM Process					
Selection Rate	33.3333%	60%	60%	34.8%	20%
Detection Loss	0%	0%	0%	0%	0%
Time (ms)	17.42	13.98	15.1	21.82	14.9
TCP Process					
$APFD$	0.8333	0.9	0.7	0.7395	0.9
$APFD_m$	1	1	0.75	0.75	1
Random $APFD_m$	0.8277	1	0.6185	0.4882	0.8806
Time (ms)	5.32	3.78	4.24	10.56	4.4
TCO Process					
Selection Rate	33.3333%	60%	60%	35.92%	20%
Detection Loss	0%	0%	0%	0%	0%
$APFD$	-	0.8333	0.6666	0.7213	-
$APFD_m$	-	1	0.75	0.75	-
TCP Time (ms)	0	3.46	3	4.12	0
Overhead	336.14 ms				

minimisation. The original and modified APFD metrics are used to evaluate the TCP-related question.

There are three kinds of test suites within our experiments: the *original test suite* (generated by AgileUML directly), the *prioritised test suite* (only applied TCP process), and *the optimised test suite* (applied both TCM and TCP processes). We performed the non-parametric Wilcoxon test on the APFD metrics to verify a significance level of 5%, with the null hypothesis that the observed

Table 5 MathLib (2)

Name	**C6**	**C7**	**C8**	**C9**	**C10**
Test Cases	7	20	5	5	5
Original $APFD$	0.2142	0.155	0.5	0.8	0.9
Original $APFD_m$	0.1785	0.14	0.5	0.85	1
TCM Process					
Selection Rate	33.7142%	33%	60%	60%	60%
Detection Loss	0%	0%	0%	0%	0%
Time (ms)	14.72	19.28	14.3	13.18	13.86
TCP Process					
$APFD$	0.5	0.395	0.9	0.9	0.9
$APFD_m$	0.5	0.385	1	0.997	1
Random $APFD_m$	0.4042	0.2145	0.836	0.97	0.52
Time (ms)	5.22	8.56	3.98	4.1	4.04
TCO Process					
Selection Rate	35.4285%	33.8%	60%	60%	60%
Detection Loss	0%	0%	0%	0%	0%
$APFD$	0.5	0.385	0.8333	0.8333	0.8333
$APFD_m$	0.5	0.355	1	1	1
TCP Time (ms)	2.8108	3.64	3.16	2.88	3.18
Overhead	336.14 ms				

Table 6 MathLib3 (3)

Name	**C11**	**C12**	**C13**	**C14**	**C15**
Test Cases	5	5	5	20	100
Original $APFD$	0.8	0.9	0.4333	0.345	0.795
Original $APFD_m$	0.85	1	0.4666	0.34	0.795
TCM Process					
Selection Rate	60%	60%	60%	31.7%	41.84%
Detection Loss	0%	0%	0%	0%	0%
Time (ms)	13.52	14.38	13.9	18.62	42.4
TCP Process					
$APFD$	0.9	0.9	0.7	0.785	0.9924
$APFD_m$	1	1	0.7333	0.7999	0.996
Random $APFD_m$	0.953	0.923	0.6346	0.4246	0.9283
Time (ms)	3.78	3.82	4.1	7.9	26.42
TCO Process					
Selection Rate	60%	60%	60%	31.9%	41.78%
Detection Loss	0%	0%	0%	0%	0%
$APFD$	0.8333	0.8333	0.5	0.7469	0.9815
$APFD_m$	1	1	0.5555	0.8	0.9902
TCP Time (ms)	2.66	2.92	2.68	3.38	8.92
Overhead	336.14 ms				

differences in the modified APFD value between compared test suites are not statistically significant.

To obtain the statistical findings, three distinct sets of comparisons were carried out. *Group A* involved comparing the original test suite with the prioritised one, aiming to investigate the potential improvement in early fault detection capability facilitated by the TCP process. *Group B* compared the

Table 7 UML2PY (1)

Name	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**
Test Cases	2	5	5	125	125	25
Original $APFD$	0.25	0.3666	0.8384	0.5949	0.3213	0.5533
Original $APFD_m$	0.125	0.3666	0.923	0.5934	0.3186	0.55
TCM Process						
Selection Rate	50%	40%	60%	41.776%	41.776%	43.68%
Detection Loss	0%	0%	0%	0%	0%	0%
Time (ms)	12.22	593.12	20205.2	23694.46	127896.86	492.86
TCP Process						
$APFD$	0.5	0.5666	0.8384	0.624	0.3346	0.82
$APFD_m$	0.5	0.5666	0.923	0.625	0.3333	0.8333
Random $APFD_m$	0.305	0.4799	0.923	0.6218	0.3326	0.7223
Time (ms)	2.68	814.32	15830	41444.58	245046	496.24
TCO Process						
Selection Rate	50%	40%	60%	41.744%	41.584%	44%
Detection Loss	0%	0%	0%	0%	0%	0%
$APFD$	-	0.4166	0.782	0.6226	0.3365%	0.803
$APFD_m$	-	0.4166	0.923	0.625	0.33%	0.8333
TCP Time (ms)	0	413.34	11093.66	14645.76	82150.2	348.76
Overhead	1958134.06 ms					

Table 8 UML2PY (2)

Name	**D7**	**D8**	**D9**	**D10**	**D11**	**D12**
Test Cases	125	125	125	125	125	125
Original $APFD$	0.7118	0.764	0.156	0.708	0.3213	0.764
Original $APFD_m$	0.7114	0.764	0.1528	0.707	0.3186	0.764
TCM Process						
Selection Rate	42.096%	42.2%	42.127%	41.88%	41.712%	42.096%
Detection Loss	0%	0%	0%	0%	0%	0%
Time (ms)	1048380	748.02	5535.2	25250.9	14143.8	739.28
TCP Process						
$APFD$	0.885	0.996	0.2024	0.7479	0.3346	0.9958
$APFD_m$	0.9032	0.9992	0.1995	0.75	0.3333	0.9997
Random $APFD_m$	0.8821	0.9357	0.1892	0.7452	0.3316	0.925
Time (ms)	862544	213.6	11878.3	38329.2	29532.8	211.1
TCO Process						
Selection Rate	42.144%	42.255%	42.16%	42.144%	42.064%	42.144%
Detection Loss	0%	0%	0%	0%	0%	0%
$APFD$	0.8955	0.99	0.2056	0.7452	0.3365	0.9905
$APFD_m$	0.9032	0.9989	0.1999	0.75	0.3333	0.9994
TCP Time (ms)	499110	219.5	3896.8	13545.9	9918	225.1
Overhead	1958134.06 ms					

prioritised test suite with a randomly ordered suite, with the intention of confirming the benefits of the TCP process over a random approach during the testing phase. Lastly, *Group C* compared the original test suite with the optimised version to verify whether the minimised test suite still advantages the TCP process. The comprehensive results are displayed in *Table 10*. While we do not explicitly report the APFD value for the random ordering under the original metric, a comparative analysis is still conducted.

Table 9 UML2PY (3)

Name	**D13**	**D14**	**D15**	**D16**	**D17**	**D18**
Test Cases	125	125	5	5	125	5
Original $APFD$	0.956	0.916	0.7787	0.7	0.5413	0.1
Original $APFD_m$	0.956	0.916	0.853	0.7625	0.5396	0.0333
TCM Process						
Selection Rate	41.76%	42.352%	40%	20%	42.384%	20%
Detection Loss	0%	0%	0%	0%	0%	0%
Time (ms)	679.84	972.8	58970.94	7264.12	60109.9	165.28
TCP Process						
$APFD$	0.996	0.996	0.806	0.7455	0.5781	0.3666
$APFD_m$	1	1	0.88	0.8734	0.5771	0.3293
Random $APFD_m$	0.9962	0.9716	0.8732	0.8155	0.5262	0.1526
Time (ms)	74.8	210.4	52146.72	6763.22	112976.4	232.82
TCO Process						
Selection Rate	41.776%	42.4%	40%	20%	42.416%	20%
Detection Loss	0%	0%	0%	0%	0%	0%
$APFD$	0.99	0.99	0.6742	-	0.5686	-
$APFD_m$	1	1	0.8598	-	0.567	-
TCP Time (ms)	66.34	222.08	29368.18	0	38629.96	0
Overhead	1958134.06 ms					

Table 10 p-value for Comparison

	Bond	**Interest Rate**	**MathLib**	**UML2PY**
A - Original	6.734926e-23	6.032983e-23	4.267474e-23	2.644815e-20
A - Modified	1.034388e-22	2.628025e-23	1.034388e-22	2.896893e-20
B - Original	2.727755e-19	6.608213e-20	4.701583e-20	5.938878e-18
B - Modified	1.358524e-19	3.300387e-20	1.109617e-19	6.376576e-18
C - Original	4.512534e-19	3.097848e-20	2.120344e-20	9.393336e-19
C - Modified	4.756323e-19	3.123845e-20	2.20862e-20	9.400991e-19

• *In response to RQ 1.1*: An analysis of *Tables 2 - 9* indicates that the execution of the TCM process on the original test suite results in a reduction of size between 40% and 80%, without any loss in fault detection capacity. Generally, the minimised test suite contains around 40% of the original test cases. These results confirm the efficacy and safety of applying the TCM process to systems whose specifications are expressed in OCL.

Fig. 7 - 9 display the distribution of the selection rate for the minimised test suite for each case study. Each plot (x-axis) in these figures corresponds to a particular operation within the respective case study. Observing these charts, it is noticeable that, except for operations *C6* and *C14* in the MathLib case, the minimisation process consistently exhibits a stable performance with only slight deviations across 50 executions.

Our analysis indicates that in most cases, the proposed TCM process generally enhances the testing process. The minimisation procedure efficiently decreases the size of the test suite without any loss in fault detection capability.

• *In response to RQ 1.2*: We examined the effectiveness of the TCP process using the original and modified APFD metrics without the involvement of the TCM process. Regardless of which metric is used for evaluation, there is an

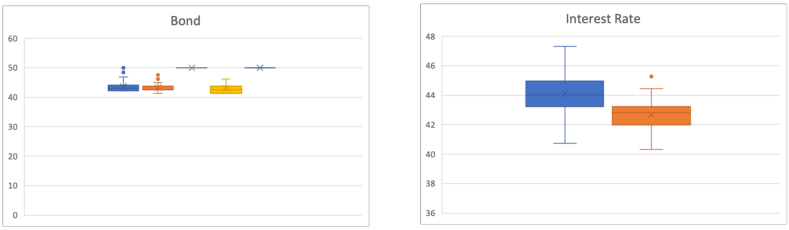

Fig. 7 TCM Process for Bond and Interest Rate

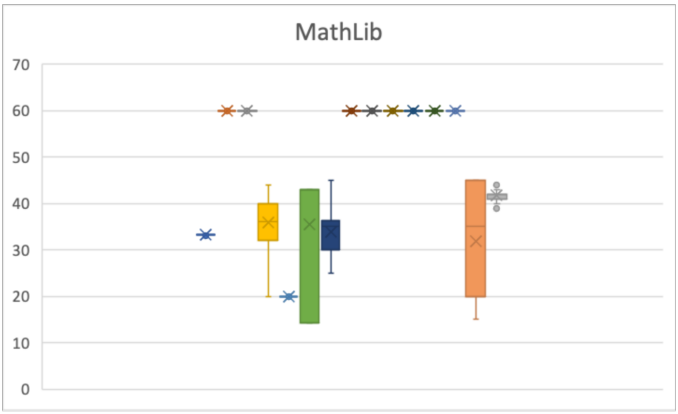

Fig. 8 TCM Process for MathLib

observable increase in the APFD value following the prioritisation process. While *A5, C2, C10, C12*, and *D3* show no improvements, we further analysed these specific instances. The absence of enhancement can be traced to the optimal sequencing already present in the original test suite, which results in the prioritisation process not offering any additional benefits to the testing process.

• *In response to RQ 1.3*: Similar to the findings in *RQ 1.2*, the optimised test suite enhances the testing process irrespective of whether the original or modified APFD metric is utilised. The p-value presented in Table 10 further confirms a significant variance between the original and optimised test suites, in which both TCM and TCP processes are applied. In the result tables, there are instances where the TCO part is marked with a "-". The reason for this is that after the TCM process, only one test case is selected, causing the execution of the TCP process to be unnecessary.

• *In summary of RQ 1*: Our findings validate that applying the TCP process, the TCM process, or a combination of both, generally brings advantages to the testing process for systems whose specifications are expressed in OCL. This observation is consistent regardless of whether the original or the modified version of APFD is used as the evaluation metric.

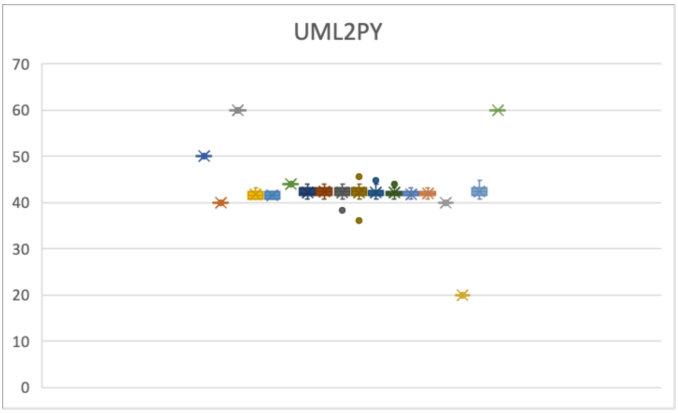

Fig. 9 TCM Process for UML2PY

In response to RQ2: Throughout our experiments, we monitored the duration of the TCP and TCM processes. The overhead incorporates aspects such as the analysis of OCL specifications and test detection information, the generation and injection of mutants, and the execution of the optimisation algorithms. The overall time required for all procedures (TCM + TCP) fluctuates between a few seconds to around 30 minutes, largely dependent on the complexity of the OCL specifications. In most instances, the overhead does not exceed the time typically spent on a lunch break. From these observations, we can deduce that the overall overhead associated with applying the TCP or TCM process to OCL-based systems is generally reasonable and manageable.

In response to RQ 3: This research question aims to uncover the correlation and variances between the original and modified versions of the APFD metric. The modified variant enhances the original by introducing a reward and penalty system for the test case and undetectable defects. Throughout our experimental procedure, we could not determine a direct correlation between these two metrics. However, we can consolidate our findings into three main observations. Firstly, when the number of defects identifiable by the first test case matches the number of undetectable defects, both metrics will produce identical results. In this case, the modified APFD metric essentially reverts to its original form. Second, when the defects detectable by the first test case outnumber the undetectable ones, the modified APFD value will typically exceed the original one. And vice versa for the third situation.

In addressing *RQ 1*, we noted that employing either the TCP process alone or in combination with the TCM process generally benefits the testing process under both the original and modified APFD evaluation metrics. Reasonably, we noted that the time required for the TCP process decreases subsequent to the TCM process due to the reduced size of the test suite. A comparison of the time spent on the TCP process under both circumstances was analysed, with the results outlined in *Table 11*. The calculated p-value indicates that

Table 11 p-value for Time Comparison

Case Study	p-value
Bond	7.056516e-18
Interest Rate	2.49044e-15
MathLib	6.871565e-14
UML2PY	7.066072e-18

performing the TCM process prior to the TCP process always reduces the time necessary for prioritisation.

Discussions: We present several discussions related to the outcomes of this study.

The first point concerns the sequence of test cases within the original test suite. Both the TCM and TCP processes alter the original test suite, and our evaluation metrics consistently compare the optimised suite with the original and random ones. This raises a question: Are these optimisations still meaningful if the initially generated test suite is considered sufficient, a condition we have noted in some of our case studies? Indeed, it is always desirable to optimise the test case generation process. However, it is crucial to note that a perfect generation process cannot be guaranteed. Furthermore, numerous pre-existing test suites still need to be optimised, illustrating the continuing applicability of these optimisation techniques.

The second discussion focuses on the modified APFD metric. Our modifications aimed to solve the limitations of the original metric, particularly its failure to range from 0 to 1 and its handling of undetectable faults. In cases where all faults are detectable, the maximum value could potentially reach 1. Conversely, the maximum value becomes the fault detection rate if some faults are undetectable. Although the modified version enhances the original in certain aspects, it is clear that the modified APFD metric could still benefit from further refinement.

Our final discussion is related to the evaluation process, specifically concerning the original APFD metric and equivalent mutants. For the original APFD metric, undetectable defects are not taken into account. In our experimental approach, when we encounter a mutant that cannot be detected, we hypothesise that the last test case will identify the mutant, thus making the TF_i equal to the total number of test cases in the test suite.

On the subject of equivalent mutants, these are mutated specifications where, although the mutation operator has been applied, the original behaviour of the specifications remains unaltered. While the detection of equivalent mutants is recognised as an undecidable problem [48], their existence does impact the mutation testing process, particularly in terms of the results of the mutation score. In our experiment process, we straightforwardly classify equivalent mutants as undetectable.

6 Threats to Validity

Threats to Internal Validity: This threat is related to potential inconsistencies in the treatment of case studies that might influence the outcomes [49]. We have mitigated these threats by ensuring all case studies were conducted under the same configuration and environment. Every algorithm was written in JAVA, and each experiment was conducted on the same machine. For both NSGA-II and genetic algorithm, consistent parameters were used across all studies, such as population size, crossover probability, and mutation operator, regardless of the differences in OCL specification. These precautions helped minimise the internal validity threats.

Threats to External Validity: Although we used four case studies for our experiments, the primary threat to external validity is the limited generalisability of these chosen studies. As with any empirical evaluation, the OCL specifications in this study may not fully represent the entire population. To mitigate this threat, we selected systems that are expressed in OCL with varying complexity, with generated test cases ranging from a handful to hundreds. Moreover, some operations within the OCL standard library lacked corresponding mutation operators in this work because, based on our strategies, we were unable to identify suitable mutation operators for these operations, leading us to omit them. Our approach of ignoring equivalent mutants and treating them as not exterminated also presents a potential validity threat. These gaps might be bridged with contributions from other OCL users and through further research.

Threats to Construct Validity: Our work uses two versions of the genetic algorithm, NSGA-II and GA, to navigate the TCM and TCP processes. The stochastic nature of the evolutionary algorithm may result in inconsistent outcomes between different executions. To minimise this threat, we ran each experiment 50 times and used the averaged results. Another threat is linked to the NSGA-II algorithm being a multi-objective optimisation algorithm, producing a Pareto set. It is impractical to examine every solution in the Pareto set manually. To deal with this threat, we selected the solution from the result set with the maximum fault detection ability and minimised test size according to the objective of the TCM process.

7 Conclusion

In this study, we have conducted TCM and TCP processes on systems whose specifications are expressed in OCL, supported by the AgileUML tool-set and guided by a genetic algorithm and its variations NSGA-II. This work has delivered three primary contributions: 1) We have proven the feasibility of applying TCM and TCP processes to the systems expressed in OCL. 2) We have innovatively modified the APFD metric, making it more compatible under the MDE and mutation testing context. 3) We have assessed the efficiency and overhead of the proposed approach through empirical studies on four real-world case studies.

The empirical results confirm the effectiveness of our proposed approach. The TCM process successfully decreases the size of the test suite while preserving its ability to detect faults, and the TCP process proves efficient in enhancing early fault detection capability. Importantly, performing the TCM process before the TCP process has been found beneficial in reducing optimisation time. The overhead associated with these methods is also reasonable, typically ranging from a few seconds to approximately 30 minutes.

Looking towards the future, we plan to extend test case optimisation to a broader array of OCL specifications to establish more general findings. We intend to consider a more extensive range of OCL expressions and operators to obtain more precise outcomes, thereby expanding the potential OCL specifications that can be utilised as case studies. Beyond TCM and TCP, we aim to investigate further optimisation strategies and compare performance across different optimisation algorithms.

References

[1] Maciel D, Paiva AC, Da Silva AR. From Requirements to Automated Acceptance Tests of Interactive Apps: An Integrated Model-based Testing Approach. In: ENASE; 2019. p. 265–272.

[2] Srivatsava PR, Mallikarjun B, Yang XS. Optimal test sequence generation using firefly algorithm. Swarm and Evolutionary Computation. 2013;8:44–53.

[3] Yoo S, Harman M. Regression testing minimization, selection and prioritization: a survey. Software testing, verification and reliability. 2012;22(2):67–120.

[4] Gupta N, Sharma A, Pachariya MK. An insight into test case optimization: ideas and trends with future perspectives. IEEE Access. 2019;7:22310–22327.

[5] Object Management Group OCLOSV. OMG Document formal/2014-02-03; 2014. .

[6] Ali S, Yue T, Zohaib Iqbal M, Panesar-Walawege RK. Insights on the use of OCL in diverse industrial applications. In: International conference on system analysis and modeling. Springer; 2014. p. 223–238.

[7] Lano K, Kolahdouz-Rahimi S. Extending OCL with map and function types. In: International Conference on Fundamentals of Software Engineering. Springer; 2021. p. 108–123.

[8] Lano K. Adding Regular Expression Operators to OCL. In: STAF Workshops; 2021. p. 162–168.

[9] Gogolla M, Burgueño L, Vallecillo A. Refactoring Collections in OCL. In: STAF Workshops; 2021. p. 142–148.

[10] Jin K, Lano K. Design and classification of mutation operators for OCL specification. In: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings; 2022. p. 852–861.

[11] Jin K, Lano K. OCL-based test case prioritisation using AgileUML. In: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings; 2022. p. 607–611.

[12] Weißleder S, Sokenou D. Automatic test case generation from UML models and OCL expressions. Software Engineering 2008. 2008;.

[13] : AgileUML repository. Available from: https://github.com/eclipse/agileuml/.

[14] Lano K, Jin K, Tyagi S. Model-based testing and monitoring using agileuml. Procedia Computer Science. 2021;184:773–778.

[15] de Castro-Cabrera MdC, García-Dominguez A, Medina-Bulo I. Trends in prioritization of test cases: 2017-2019. In: Proceedings of the 35th annual acm symposium on applied computing; 2020. p. 2005–2011.

[16] Epitropakis MG, Yoo S, Harman M, Burke EK. Empirical evaluation of pareto efficient multi-objective regression test case prioritisation. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis; 2015. p. 234–245.

[17] Silva Ouriques JF, Cartaxo EG, Lima Machado PD. Revealing influence of model structure and test case profile on the prioritization of test cases in the context of model-based testing. Journal of Software Engineering Research and Development. 2015;3(1):1–28.

[18] Whitley D. A genetic algorithm tutorial. Statistics and computing. 1994;4:65–85.

[19] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation. 2002;6(2):182–197.

[20] Lipton RJ.: Fault diagnosis of computer programs. Carnegie Mellon Univ., Tech. Rep.

[21] Gutiérrez-Madronal L, Domınguez-Jiménez JJ, Medina-Bulo I. Mutation testing: Guideline and mutation operator classification. ICCGI 2014. 2014;p. 184.

[22] Tallam S, Gupta N. A concept analysis inspired greedy algorithm for test suite minimization. ACM SIGSOFT Software Engineering Notes. 2005;31(1):35–42.

[23] Palomo-Lozano F, Estero-Botaro A, Medina-Bulo I, Núñez M. Test suite minimization for mutation testing of WS-BPEL compositions. In: Proceedings of the Genetic and Evolutionary Computation Conference; 2018. p. 1427–1434.

[24] Bhatia PK, et al. Test case minimization in COTS methodology using genetic algorithm: a modified approach. In: Proceedings of ICETIT 2019. Springer; 2020. p. 219–228.

[25] Lin X, Zhang H, Xia H, Yu L, Fang X, Chen X, et al. Test case minimization for regression testing of composite service based on modification impact analysis. In: International Conference on Web Information Systems and Applications. Springer; 2020. p. 15–26.

[26] Hashim NL, Dawood YS. Test case minimization applying firefly algorithm. International Journal on Advanced Science, Engineering and Information Technology. 2018;8(4-2):1777–1783.

[27] Deneke A, Assefa BG, Mohapatra SK. Test suite minimization using particle swarm optimization. Materials Today: Proceedings. 2022;60:229–233.

[28] Li L, Zhou Y, Yuan Y, Wu S. An Extensive Study on Multi-Priority Algorithm in Test Case Prioritization and Reduction. In: 2021 2nd Asia Service Sciences and Software Engineering Conference; 2021. p. 48–57.

[29] Bajaj A, Abraham A, Ratnoo S, Gabralla LA. Test Case Prioritization, Selection, and Reduction Using Improved Quantum-Behaved Particle Swarm Optimization. Sensors. 2022;22(12):4374.

[30] Rapos EJ, Dingel J. Using fuzzy logic and symbolic execution to prioritize UML-RT test cases. In: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST). IEEE; 2015. p. 1–10.

[31] Shin KW, Lim DJ. Model-based test case prioritization using an alternating variable method for regression testing of a UML-based model. Applied Sciences. 2020;10(21):7537.

[32] Pospisil T, Sobotka J, Novak J. Enhanced Adaptive Random Test Case Prioritization for Model-based Test Suites. Acta Polytechnica Hungarica. 2020;17(7).

[33] Pan R, Bagherzadeh M, Ghaleb TA, Briand L. Test case selection and prioritization using machine learning: a systematic literature review. Empirical Software Engineering. 2022;27(2):1–43.

[34] Ma B, Wan L, Yao N, Fan S, Zhang Y. Evolutionary selection for regression test cases based on diversity. Frontiers of Computer Science. 2021;15(2):1–3.

[35] Rattan P, Arora M, Rakhra M, Goel V, et al. A Neoteric Approach of Prioritizing Regression Test Suites Using Hybrid ESDG Models. Annals of the Romanian Society for Cell Biology. 2021;p. 2965–2973.

[36] Sornkliang W, Phetkaew T. Target-based test path prioritization for UML activity diagram using weight assignment methods. International Journal of Electrical and Computer Engineering. 2021;11(1):575.

[37] Chaudhary S, Jatain A. Performance Evaluation of Clustering Techniques in Test Case Prioritization. In: 2020 International Conference on Computational Performance Evaluation (ComPE). IEEE; 2020. p. 699–703.

[38] Morozov A, Ding K, Chen T, Janschek K. Test suite prioritization for efficient regression testing of model-based automotive software. In: 2017 International Conference on Software Analysis, Testing and Evolution (SATE). IEEE; 2017. p. 20–29.

[39] Sun Ca, Liu B, Fu A, Liu Y, Liu H. Path-directed source test case generation and prioritization in metamorphic testing. Journal of Systems and Software. 2022;183:111091.

[40] Rothermel G, Untch RH, Chu C, Harrold MJ. Test case prioritization: An empirical study. In: Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99).'Software Maintenance for Business Change'(Cat. No. 99CB36360). IEEE; 1999. p. 179–188.

[41] Walcott KR, Soffa ML, Kapfhammer GM, Roos RS. Timeaware test suite prioritization. In: Proceedings of the 2006 international symposium on Software testing and analysis; 2006. p. 1–12.

[42] Zhang X, Qu B. An improved metric for test case prioritization. In: 2011 Eighth Web Information Systems and Applications Conference. IEEE; 2011. p. 125–130.

[43] Egea M, Rusu V. Formal executable semantics for conformance in the MDE framework. Innovations in Systems and Software Engineering. 2010;6:73–81.

[44] Büttner F, Gogolla M. On OCL-based imperative languages. Science of Computer Programming. 2014;92:162–178.

[45] Brucker AD, Tuong F, Wolff B. Featherweight OCL: A Proposal for a Machine-Checked Formal Semantics for OCL 2.5. In: 15th International Workshop on OCL and Textual Modeling co-located with 18th International Conference on Model Driven Engineering Languages and Systems; 2015. p. 199.

[46] Willink E. An OCL map type. In: OCL Workshops; 2019. .

[47] Lano K, Kolahdouz-Rahimi S, Jin K. OCL libraries for software specification and representation. In: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings; 2022. p. 894–898.

[48] Offutt AJ, Pan J. Detecting equivalent mutants and the feasible path problem. In: Proceedings of 11th Annual Conference on Computer Assurance; 1996. .

[49] Miranda B, Cruciani E, Verdecchia R, Bertolino A. FAST approaches to scalable similarity-based test case prioritization. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). IEEE; 2018. p. 222–232.