Journal of King Saud University – Computer and Information Sciences 34 (2022) 6324–6338

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Test case prioritization for model transformations

Saqib Iqbal [a], Issam Al-Azzoni [a,*]

[a] College of Engineering, Al Ain University, Al Ain, United Arab Emirates

ARTICLE INFO

Article history:
Received 28 March 2021
Revised 9 August 2021
Accepted 9 August 2021
Available online 21 August 2021

Keywords:
Model transformations
Model-driven engineering
Regression testing
Test case prioritization

ABSTRACT

The application of model transformations is a critical component in Model-Driven Engineering (MDE). To ensure the correctness of the generated models, these model transformations need to be extensively tested. However, during the regression testing of these model transformations, it becomes too costly to frequently run a large number of test cases. Test case prioritization techniques are needed to rank the test cases and help the tester during the regression testing to be more efficient. The objective is to rank the fault revealing test cases higher so that a tester can only execute the top ranked test cases and still be able to detect as many faults as possible in the case of limited budget and resources. The aim of this paper is to present a test prioritization approach for the regression testing of model transformations. The approach is based on exploiting the rule coverage information of the test cases. The paper presents an empirical study which compares several techniques introduced by our approach for prioritizing test cases. The approach is complemented with a tool that implements the proposed techniques and can automatically generate test case orderings.

1. Introduction

Model-driven Engineering (MDE) simplifies development of complex and large systems by representing all types of development artifacts and viewpoints of the system in models. Models provide abstraction which helps in focusing on the most important and critical problems besides improving representation of artifacts, increasing design efficiency and compatibility. Due to the increase in the size and complexity of modern software, MDE has been gaining immense popularity. Model transformation is one of the most important activities of the MDE, which allows manipulation and transformation of models (Lúcio et al., 2016). The process of model transformation is supported by transformation languages which are composed of transformation rules. Each rule defines the transformation of an element of the input model to an element of the target model. The rules match the parts of the input model to transform them into parts of the target model.

The correctness of the models is heavily dependent on the correctness of execution of transformation processes. The target models, therefore, are tested to ensure the correctness similar to the traditional testing of the source code. Regression testing, which ensures that the models are correct after some changes have been made in response to the testing, is also an important activity in the MDE. Regression testing, however, is a tedious work because the number of test cases grow exponentially as a result of rigorous testing. To reduce such overhead cost of regression testing, techniques such as regression test selection (Harrold et al., 2001; Rothermel and Harrold, 1994), test suite minimization (Harrold et al., 1993; Korel et al., 2002) and test case prioritization (TCP) (Malishevsky et al., 2006; Rothermel et al., 2001) are employed. TCP intends to select minimal number of test cases with maximum coverage in order to reduce the overhead cost of regression testing. This paper presents a TCP approach for the regression testing of model transformations. The presented approach exploits several techniques for ranking test cases on the basis of their rule coverage. This, in return, helps testers to first run the test cases that have an expected high number of revealed faults if they face time and budget constraints.

The main contributions of this paper are summarized as follows:

1. We present an approach for test case prioritization for model transformations.

* Corresponding author.
E-mail addresses: saqib.iqbal@aau.ac.ae (S. Iqbal), issam.alazzoni@aau.ac.ae (I. Al-Azzoni).

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2021.08.011
1319-1578/

2. We present several techniques that can be exploited by our approach to find a test case ordering that maximizes fault detection effectiveness.
3. We present a tool that implements our approach and can automatically generate test case orderings.
4. We present the results of an empirical study that compares the proposed techniques on several model transformation case studies.

The organization of the paper is as follows. First, we provide the necessary background in Section 2. A motivating example is presented in Section 3. Our proposed approach for test case prioritization is presented in Section 4. Section 5 presents our empirical study and discusses the results of our experiments. The related literature is discussed in Section 6. The conclusion and future work are discussed in Section 7.

2. Background

MDE has been increasingly gaining popularity with the software systems becoming more complex and larger. MDE provides an alternate solution for the development of such complex systems by providing a model-based way of understanding and solving problems related to design and development. This section presents some of the basic concepts of MDE.

2.1. Model-driven engineering

MDE refers to documenting, designing and implementing a system using models as primary artifacts. The requirements, design and implementation of the system is represented in models at different abstraction levels. Models help in reducing complexity of the artifacts for analysts and designers to focus on the issues related to the implementation of the solution to the business problems. Models are translated into concrete artifacts such as source code, configuration code and test scripts using automated methods. Rumbaugh et al. (1991) define a model as an abstraction of a complex concept in order to understand it before implementing it. MDE has achieved continuous popularity over the past couple of decades for engineering complex and scalable systems with enhanced productivity, better quality, and reduced maintenance cost (Brambilla et al., 2017; Mussbacher et al., 2014; Van Der Straeten et al., 2008). The technique has also been proven useful in reverse engineering and in the representation of high abstraction of code and data in software evolution processes (Mazón et al., 2007; Ramón et al., 2014). In MDE, models are represented and transformed using Domain Specific Languages (DSLs), which are more customizable compared to the general-purpose Unified Modelling Language (UML). DSLs can represent models for applications from various domains, such as mobile, web-based or embedded (Kelly and Tolvanen, 2008). The syntax of DSMLs is usually described using meta-models that define the elements of the models and the relationships between them.

2.2. Metamodels

Metamodels are models that define structure, syntax and constraints to build valid models. For example, UML has metamodels that have definition and constraints to define UML elements, such Package, Class, Association, etc. The models built from the metamodels are supposed to conform to the definitions and constraints defined in the metamodels. Metamodels are defined for general purpose languages, such as UML, and also for domain specific languages. Metamodels are models themselves which dictate the way models following the metamodels can be defined. The structure of the defined models, associations between them and the rules and constraints applied on the models must conform to their metamodels. The syntax of DSLs is defined by the metamodels but semantic and constraints are not fully defined by metamodels. Model transformations resolve these issues by providing semantics to a DSL, for instance, they can define the rules of translating a DSL to a different domain for further analysis (Troya and Vallecillo, 2014), thereby making them reusable and flexible for more comprehensive analysis (Moreno-Delgado et al., 2014).

2.3. Model transformation

Model transformation is the cornerstone of the MDE framework, which manipulates models and translates them into other models or into code (Brambilla et al., 2017; Sendall and Kozaczynski, 2003). A model-to-model (M2M) transformation is a program that runs on a transformation engine to convert one or more input models to one or more target models by following defined transformation rules (specifications). The process is illustrated in Fig. 1.

Model Transformation languages are based on metamodels which make the transformation process reusable for the model instances which conform to the transformation language. Several transformation languages have been proposed in the last decade, such as AGG (Taentzer et al., 2004), ATOM (Lara and Vangheluwe, 2002), VIATRA (Csertan et al., 2002), QVT (Greenyer and Kindler, 2010), Kermeta (Jézéquel et al., 2011), JTL (Cicchetti et al., 2011), and ATL (Jouault et al., 2006). A commonly used transformation language is ATL due to its flexibility, support to metamodeling and integration with tools such as Eclipse.

2.4. Regression testing of models

Regression testing is the process of re-testing of the system after the system has gone through some modifications as a result of regular testing. The regression testing ensures that the modifications did not have an unintended effect and the system still con-

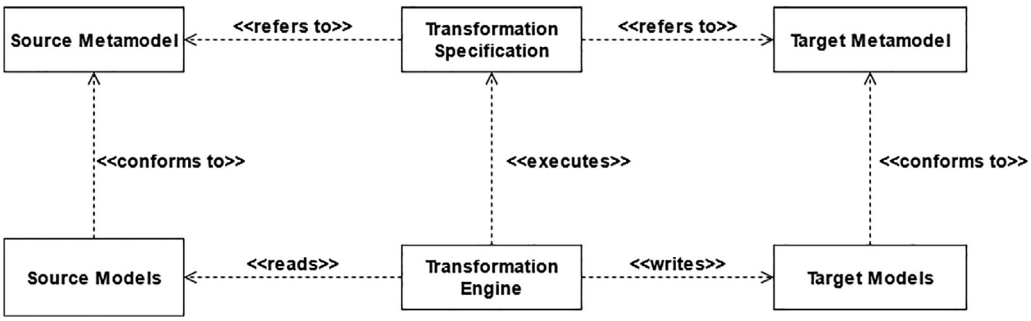


Fig. 1. Model Transformation Pattern (reproduced from Czarnecki and Helsen (2006)).

forms to its specifications (IEEEStd2010,). Since the MDE follows an incremental development process, the models change over time similar to the source code. These changes affect the system properties and functions, which prompts re-testing (regression testing) of the changed models. The models also required to be re-tested when the transformation rules or the coverage criteria are modified. The completeness and correctness of regression testing ensure quality, consistency and robustness of models in MDE (Brottier et al., 2006; Mottu et al., 2008).

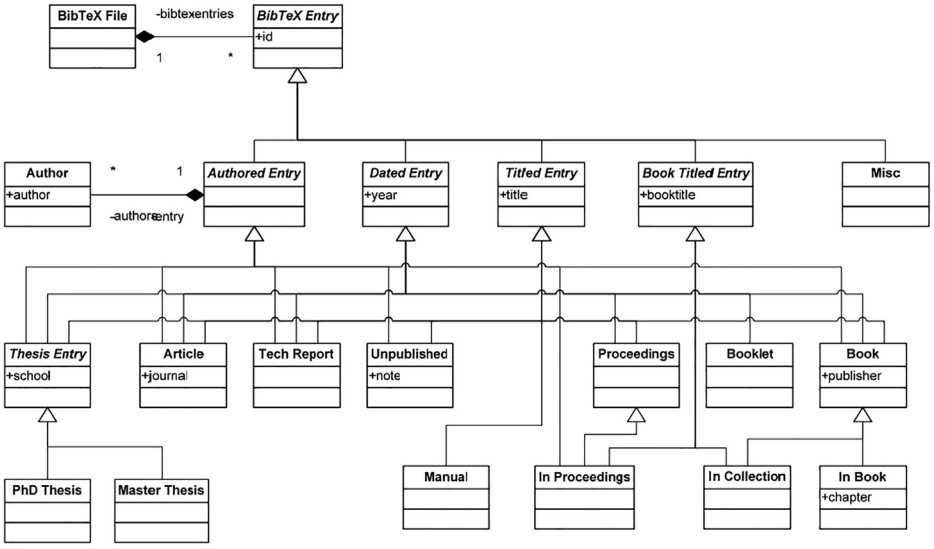
Despite its critical importance, regression testing also poses a big overhead of cost, time and effort due to the large size of test cases (Yoo and Harman, 2012; Zhang et al., 2016). Some studies have shown that the regression testing counts for 80% of the testing cost (Chittimalli and Harrold, 2009). To reduce the cost of regression testing, a number of techniques have been proposed, such as Regression Test Selection, Test Suite Minimization and Test Prioritization (Yoo and Harman, 2012). Regression Test Selection (Harrold et al., 2001; Rothermel and Harrold, 1994) only produces test cases that are affected by the modifications. Test Suite Minimization (Harrold et al., 1993; Korel et al., 2002) minimizes the size of test cases by eliminating redundant test cases. Test Prioritization (Malishevsky et al., 2006; Rothermel et al., 2001) reorders and prioritizes test cases in order to effectively identify faults and errors.

Out of the three techniques, Test Selection and Test Suite Minimization techniques are considered risky options as they often exclude critical test cases in order to reduce the size of test cases (Lou et al., 2019). On the contrary, Test Prioritization does not exclude any test case but rather reorders the test cases based on a priority criteria. Therefore, Test Prioritization does not suffer from losing fault-detection effectiveness and is considered a safe option compared to other approaches (Elbaum et al., 2014; Marijan et al., 2013; Srivastava and Thiagarajan, 2002).

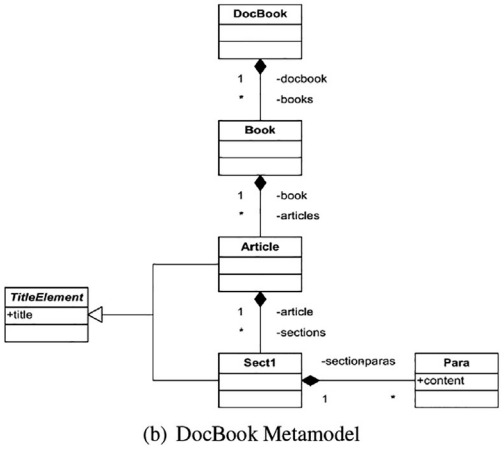
3. Transformation example

In this example, we consider a model transformation example that uses ATL to transform a model that conforms to the BibTeX metamodel to a model that conforms to the DocBook metamodel. The code of the example was obtained from Atenea (2021) and is presented in the Appendix. Model transformation from BibTeX to DocBook models is a classic example used by several authors in the literature to present and evaluate new techniques and approaches for model transformations (Alkhazi et al., 2020; Troya et al., 2018a).

The BibTeX and DocBook metamodels are shown in Fig. 2(a) and (b), respectively. The names of abstract classes are shown in italics. In the BibTeX metamodel, a bibliography is modeled by a *BibTeXFile*



(a) BibTeX Metamodel



(b) DocBook Metamodel

Fig. 2. Metamodels of the BibTeX2DocBook transformation (taken from Atenea (2021)).

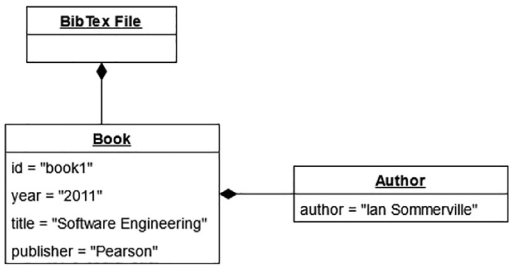


Fig. 3. Sample BibTeX model.

element which is composed of elements that inherit from the abstract *BibTeXEntry* element. There are several abstract classes which directly inherit from *BibTeXEntry*: *AuthoredEntry*, *DatedEntry*, *TitledEntry*, *BookTitledEntry*, and *Misc*. Concrete BibTeX entries inherit from one or more of the abstract classes according to their set of obligatory fields. For example, *Book* is one example of a concrete BibTeX entry. It inherits from *AuthoredEntry*, *DatedEntry*, and *TitledEntry* abstract classes. This means that for a book entry, in addition to the *publisher* attribute, a book has three inherited attributes: *id*, *year*, and *title*. Also, a book has an inherited reference, named *authors*, to a list of *Author* elements. Fig. 3 shows an example model that conforms to the BibTeX metamodel. In this example, the BibTeX file is composed of a single book whose attributes and author are shown in the figure.

In the case of the DocBook metamodel (Fig. 2(b)), a DocBook document is modeled by a *DocBook* element. This element is composed of several *Book* elements and each *Book* element is composed of several *Article* elements. An *Aricle* is composed of several sections that are ordered (a section is an instance of the class *Section 1*). A *Section 1* element is composed of several ordered paragraphs (a paragraph is an instance of the class *Para*). The classes *Article* and *Section 1* inherit from the abstract class *TitleElement*. This means that articles and sections have the *title* as an inherited attribute. Fig. 4 shows an example model that conforms to the *DocBook* metamodel and which was generated by the BibTeX2DocBook transformation which is to be explained next.

The BiBTeX2DocBook transformation (shown in the Appendix) contains nine rules. A rule specifies how target model elements should be generated from a source model element. Hence, a rule consists of an input pattern (with an optional filter condition) which must be matched on the source model element to fire the rule and an output pattern which specifies the target model elements to be created when the rule is fired. The filter conditions are specified in OCL. Each output pattern can have several bindings which specify how to initialize the attributes and references of the target model elements. The bindings are also expressed in OCL.

A full explanation of the BiBTeX2DocBook transformation program can be found in ATLTransformationExample (2021). Here, we attempt to explain the rules related to the transformation of the sample input model shown in Fig. 3. In this transformation, three rules are executed: *Main*, *Author*, and *TitledEntry_Title_NoArticle*. The first rule, *Main*, builds the basic structure of a *DocBook* from a *BibTeXFile*. It creates four sections, *se1* ... *se4*, each with its own title. The paragraphs of each section are to be set when the remaining rules are executed. Rule *Main* uses three helpers: *authorSet* which returns the sequence of authors in the input BibTeX model with no repeated names, *titledEntrySet* which returns the sequence of *TitledEntries* in the input BibTeX model with no repeated titles, and *articleSet* which returns the sequence of *Articles* in the input BibTeX model with no repeated journal names.

The rule *Author* creates a paragraph for each author and sets its content as the author's name. The rule *TitledEntry_Title_NoArticle* is executed on each *TitledEntry* that is included in the set *titledEntrySet* and that is not an article. When this rule is executed, it creates two paragraphs: the first paragraph which is referenced by the first section in the *DocBook* and the second paragraph which is referenced by the third section. Note that the content of the first paragraph is set by calling the helper *buildEntryPara* which builds the contents of paragraphs according to the corresponding entry types.

In the sample input model (Fig. 3), there is a single entry in the BibTeX file which is a book. A book is an *AuthoredEntry* and hence can be authored by several authors, a *DatedEntry* and hence has a *year* attribute, and a *TitledEntry* and hence has a title. A book has an id as well since it is a *BibTexEntry* element. Fig. 4 shows the generated output model. The *DocBook* is composed of a single *Book* element which is composed of a single *Article* element. The *Article* element is composed of the four sections which present the book information as a DocBook composed document.

During the testing of a model transformation, the tester creates a large number of test cases. In each test case, the tester needs to define an input test model. This test model must conform to the source metamodel. The tester can then run a transformation to obtain the corresponding output (target) model. To validate the transformation, the tester needs to check the satisfaction of several constraints that are defined on the input and output models. These constraints are typically expressed in OCL and there are automated tools for checking the satisfaction of OCL constraints (MDTOCLChecker, 2021). For a test case to be considered as a pass,

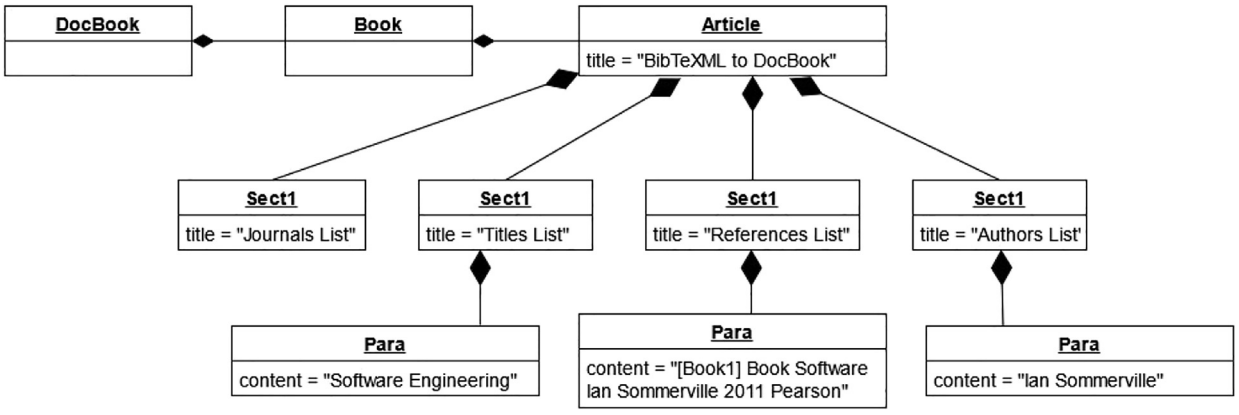
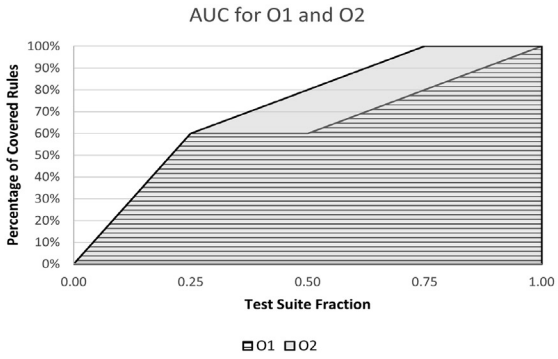


Fig. 4. Sample DocBook model.



	T1	T2	T3	T4
R1	X			
R2	X	X		
R3	X	X		
R4			X	
R5				X

Fig. 5. AUC measure for different test case orderings.

the constraints on the test case's input model and its corresponding output model must be met. This approach for testing aims to semantically check the correctness of model transformations against a set of assertions. It has been applied in other works on model transformation testing (Alkhazi et al., 2020; Troya et al., 2018a). In the *BibTeX2DocBook* transformation, for example, consider the following OCL constraint:

```
SrcAuthor.allInstances()-> forAll(a |TrgPara.
allInstances()-> exists(p |p.content = a.author and p.-
section.title = 'Authors List'))
```

This constraint states that for each author in the BibTeX file, there is a paragraph within a section named 'Authors List' and this paragraph's content is the name of the author. Note that for the test case with the input model in Fig. 3 and the output model in Fig. 4, this constraint is satisfied.

The tester needs to check that all constraints are satisfied in each test case. If there is at least one constraint that is not satisfied, then the test case is considered as a failing test case. To validate the model transformation, the tester defines a large number of test cases and attempts to ensure that all the test cases pass. In regression testing of a model transformation, the test cases are re-run whenever there is a change to the transformation program. The goal is to discover faults in the model transformation by failing test cases. In cases of a large number of test cases and frequent changes to the model transformation program, discovering a failing test case as early as possible can save testing overheads. The work presented in this paper attempts to prioritize test cases that are more likely to fail. Given a set of test cases, we aim to order these test cases so that the more likely ones to fail appear before the other test cases. Hence, the tester can discover faults as early as possible by re-running a fewer number of test cases.

4. Test case prioritization for model transformations

In this section, we present several TCP techniques which can be applied on model transformations. These techniques are evaluated in Section 5.

The presented TCP techniques use rule coverage information of each test case. The rule coverage information can be extracted by analyzing the source code of the model transformation program or by analyzing the trace model of the model transformation execution. For example, Jouault presented a method that can be used to automatically obtain a trace model from a transformation execution for ATL programs (Jouault, 2005).

The objective of the presented TCP techniques is to maximize rule coverage. The use of a "maximizing rule coverage" heuristic implies that a TCP technique must order test cases in such a way that a high coverage of rules is achieved as early as possible when the test cases are executed. The intuition behind the use of this heuristic is that by maximizing rule coverage, the test cases execute as much of the rules as possible and thus the chances of fault detection become better.

The first TCP technique applies a greedy approach for maximizing coverage. This is the simplistic coverage-based TCP approach. It uses a Greedy algorithm which ranks the test cases with the most rule coverage first. Although the Greedy technique is pretty simple and practical, it suffers from the fact that it does not take into consideration the current rule coverage while selecting the next best test case. In other words, the Greedy technique makes the locally optimal choice at each step, and in many cases it does not produce an optimal solution.

To escape from the local optima, we present two global search techniques which are based on using metaheuristic search algorithms. The first one uses Genetic Algorithm (GA), and the second one uses Ant Colony Optimization (ACO). Both techniques use a fitness function based on rule coverage. We use the area under a curve (AUC) metric, where the curve represents the total number of test cases covered when the test cases are executed following the order of test cases given by a TCP technique. More formally, we use the following metric:

$$APRC = 100 * \left(1 - \frac{TR_1 + \ldots + TR_m}{nm} + \frac{1}{2n}\right) \quad (1)$$

Here, APRC stands for "Average Percentage of Rules Covered". The APRC is motivated by APFD which is another metric, to be discussed later, that is a common metric used to measure the effectiveness of a test case prioritization technique. APRC is the weighted average of the percentage of rules covered when the test cases are executed following the specified order. In Eq. (1), n denotes the number of test cases and m denotes the number of rules. TR_i is the number of test cases which must be executed before rule i is covered. APRC corresponds to the AUC where the x-axis is the portion of the test suite executed in the given order and the y-axis is the portion of rules covered by the test executions. The maximum value of APRC occurs when all rules are covered by the first test case and the minimum APRC occurs when no rule is covered by all test cases.

Consider the following example whose details are shown in Fig. 5. Assume that there are four test cases in the test suite to be ranked. The table in the figure shows the rule coverage information for these test cases. For example, test case T1 covers rules R1, R2, and R3. This means that when test case T1 is executed, it results in executing the three aforementioned rules. Consider the following test case orderings O1: ⟨T1, T2, T3, T4⟩ and O2: ⟨T1, T3, T4, T2⟩. The test case ordering O1 is an example of an ordering which is obtained by a Greedy algorithm. The first test case in the ordering results in 60% right away. The second test case does not add to coverage, however, after that each test case results in 20% extra

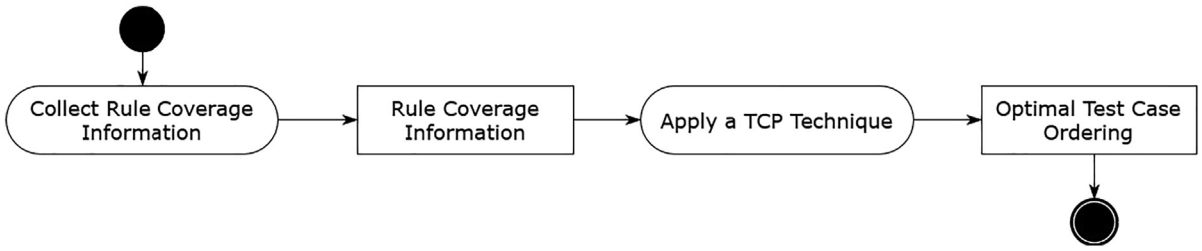


Fig. 6. An overview of our test case prioritization approach.

coverage until all rules are covered. The test case ordering O2 starts by covering 60% by the end of the first test case, but then every test case results in additional 20% coverage. Thus, O2 results in 100% coverage after executing the first three test cases.

Fig. 5 shows the AUCs for both O1 and O2. The actual AUC values are 0.625 for O1 and 0.725 for O2. Note that these values are the APRC values for the corresponding orderings. Hence, O2 achieves full coverage faster than O1, and for this reason it should be preferred. Since the APRC is used as the fitness function for a solution (ordering), O2 is considered to have a higher fitness value than O1.

To evaluate test case prioritization techniques, we use the "Average Percentage of Fault-Detection" (APFD) which is a common effectiveness metric used in test case prioritization literature. This metric was initially proposed by Rothermel et al. (2001). APFD measures how fast an ordering returned by a TCP technique detects faults. Higher APFD values mean faster fault detection rates. The following formula is used to calculate APFD values:

$$APFD = 100 * \left(1 - \frac{TF_1 + \ldots + TF_m}{nm} + \frac{1}{2n}\right) \quad (2)$$

Here, TF_i is the number of test cases which must be executed before fault i is detected. To compute the TF_i's for a given test case ordering, the test cases are executed in order and for each fault the number of the first test case which results in the fault detection is recorded. Conceptually, APFD represents the AUC where the x-axis is the portion of the test suite executed in the given order and the y-axis is the portion of unique faults detected by the test executions. This is quite similar to the use of AUC as a measure of coverage in the case of the APFD metric explained earlier.

To ease the work of the tester during the regression testing of model transformations, we have developed a tool to automate the test case prioritization process. The tool is a Java program available on GitHub at https://github.com/ialazzon/TestCasePrioritizationTool. The tool only requires two inputs from the user: the list of the rule names in the transformation and the location of a folder containing the trace models of the previous runs of the test cases. These trace models are automatically obtained from the transformation executions. A trace model is composed of traces, one for each rule execution. A trace captures the name of the applied rule, the source element, and the target elements which were created by applying the rule. Our tool utilizes this trace model to build a rule coverage information model which is used later by the TCP techniques. As an output, the tool displays the test case orderings returned by all algorithms presented in this paper. In addition, the tool displays the APRC value corresponding to each ordering.

Fig. 6 presents an overview of our test case prioritization approach. Using UML activity diagram notation, it shows the main steps involved in applying our approach. The steps can be fully automated by using our tool. To collect the rule coverage information, the test cases need to be executed at least once in order to create the execution trace model. Note that the tool implements all TCP techniques presented in the paper.

For the implementation of the GA, we used the Java-based framework presented in Jacobson and Kanber (2015). GA is an evolutionary algorithm that maintains a population of solutions that keep evolving by applying genetic operators such as mutations and crossovers. The population is initially populated by the specified number of orderings. An ordering is represented as an array of length n, where n is the size of the ranked list representing the ordering. The elements in the array have values in the range 0 to $m-1$, where m is the number of test cases to be prioritized. An element in the array at index i with value j means assigning the test case $i+1$ to the $(j+1)th$ position in the ordering. For mutation, we used the swap algorithm, and for crossover we used the 2-point crossover algorithm. For termination, we used the condition that the specified number of generations is reached. We used the cost function APRC in comparing the orderings.

The GA's setup and parameters are obtained using Design of Experiments (DoE) (Talbi, 2009) which is a popular approach for the parameters setting of evolutionary algorithms. The parameters include the mutation rate, the crossover rate, the number of generations, the population size, and the elitism count. We divide the domain of each parameter into several intervals and we choose a random value from each interval. The combination of values that achieve the best performance are chosen for the parameters.

For the implementation of the ACO algorithm, we used Isula. Isula is a Java framework for ACO algorithms presented in Gavidia-Calderon and Castañon (2020). ACO algorithms, originally proposed by Dorigo et al. (1996), emulate the behavior of ants in nature to solve combinatorial optimization problems (Blum, 2005). The ACO algorithm implemented is the Ant System (AS). The AS's setup and parameters are shown in Table 1. These values are based on a trial-and-error approach where the ACO algorithm achieved its best performance on several selected test case prioritization problem instances. To store the pheromone values, we used an $n \times m$ two-dimensional array where n is the size of the ranked list representing the ordering and m is the number of test cases to be prioritized.

5. Empirical study

In this section, we present our empirical study and discuss the results of our experiments. First, we list the main research questions to be investigated. Then, we present different case studies to be used in investigating the research questions. Third, we dis-

Table 1
AS Settings.

Number of Iterations	10
Number of Ants	2000
Evaporation Ration (ρ)	0.4
Pheromone Importance (α)	1
Heuristic Importance (β)	1

cuss how relevant data is collected. The evaluation metric is presented next, followed by a summary of results and discussion. Finally, threats to the validity of the conclusions are presented.

5.1. Research questions

There are two research questions that we want to investigate:

1. How do the proposed techniques in Section 4 perform compared to random search? Investigating this question serves as a sanity check step. If the random search technique outperforms the other techniques, then this implies that there is no need to use a metaheuristic search algorithm or a greedy-based approach.
2. Can a global search technique outperform the greedy technique? Here, we compare the proposed GA and ACO techniques with the greedy one.

5.2. Subjects under study

We have used five different case studies to evaluate our research questions. Four cases are taken from previous work on spectrum-based fault detection in model transformations (Troya et al., 2018a). The test suites used in these four cases have been semi-automatically created using model generation scripts. We obtained the test suites from ImplementationSpectrumBased (2021). The fifth case was created specifically to address the second research question and will be presented and discussed in subsequent sections. The first four case studies are as follows:

- **BibTeX2DocBook:** This transformation generates a DocBook composed document from a BibTeXML model. This transformation is available in the ATL Zoo (ATLZoo, 2021).
- **UML2ER:** This transformation generates an Entity Relationship (ER) diagram from a UML Class diagram.
- **CPL2SPL:** This is a relatively complex case which handles several aspects of model transformations between two modeling languages in the domain of Internet telephony: SPL and CPL (Jouault et al., 2006). This transformation is available in the ATL Zoo (ATLZoo, 2021).
- **Ecore2Maude:** This transformation generates Maude specifications (Clavel et al., 2007) from Ecore metamodels.

These case studies come from several application domains and they differ regarding the size of the transformation and its structure. Table 2 summarizes some information regarding the transformations. For instance, the size of the transformations ranges from 8 to 39 rules and the lines of code (LOC) from 53 to 1055. The table also shows the number of input test models and the number of mutants for each case study. These test models and mutants were created for evaluating different spectrum-based fault detection techniques for model transformations in the work of Troya et al. (2018a). Each mutant is a faulty variation of the original model transformation and is obtained by applying one or more of the mutation operators for model transformations presented in Troya et al. (2015).

In our experiments, mutants represent faults that need to be detected by the test cases. To mimic real faults, some mutants in the case studies are created by applying more than one mutation operator. In addition, a mutant can have more than a single faulty rule. When a mutant is executed, an output model is successfully generated. However, one or more of the OCL assertions fail on the output model. This allows to focus on the semantic errors made by the transformation writer, rather than the syntactic errors. Each case study has its own OCL assertions that pass on the unchanged, original transformation, but at least one OCL assertion fails on each mutant. In a transformation, we consider that a test case detects a fault when at least one of the OCL assertions fails on the output model generated by the test case's input model when the fault's corresponding mutant is executed.

Table 2
Information of the Case Studies.

Case Study	#Rules	#LOC	#Mutants	#Test Input Models
BibTeX2DocBook	9	393	40	100
UML2ER	8	53	18	100
CPL2SPL	19	503	50	100
Ecore2Maude	39	1055	50	100

Table 2 shows also that there are 100 available test cases in each case study. Each test case has a corresponding input model which was semi-automatically created in the work of Troya et al. (2018a). When a mutant of a transformation is executed on an input model, one or more output models are created. Then, the OCL assertions are evaluated against the input and its corresponding output models. If at least one assertion fails, then the test case is said to fail, and thus the fault in the corresponding mutant is considered to be detected by the test case. In our experiments, in each case study we select a number of test cases at random from the available test cases.

5.3. Data collection

We have modified the program in ImplementationSpectrumBased (2021) to obtain the rule coverage information for each of the 100 test cases available for each case study. In addition, we have extended the program to report the test case results when executing a particular mutant for each case study. We have created a separate program that calculates APFD and APRC values for different test case orderings based on the corresponding rule coverage information and the test case results on a particular mutant.

For each case study, we randomly select a number of test cases that are to be ordered. In addition, we randomly select a number of mutants that need to be killed. When we report the results for a case study, we first note the number of test cases to be ordered (n) followed by the number of mutants that need to be killed (m). For example, Fig. 7 shows the APFD results for the BibTeX2DocBook case study. Each sub-figure shows the results for a particular pair of values for the number of test cases to be ordered and the number of mutants to be killed.

5.4. Evaluation metric

To evaluate a test case ordering, we use the APFD metric. Since a test case prioritization technique can produce a different test case ordering every time it is run, it becomes necessary to repeat each experiment a minimum number of times and to perform a statistical analysis of the measured APFD values. In our case, each experiment is repeated 30 times. In each experiment, the measured APFD values are represented using a box plot where the higher median line indicates a better test case ordering.

To compare two distributions of the results, we use the non-parametric Mann–Whitney U-test for detecting statistical differences (Hollander et al., 2013). To carry a U-test, we used the function *wilcox.test* in R for a Wilcoxon Rank Sum test (R Core Team, 2019). In addition to statistical significance, we use the Vargha and Delaney's $\widehat{A}_{12}$ statistics as a non-parametric effect size measure (Arcuri and Briand, 2011; Vargha and Delaney, 2000). Effect size measures are useful to assess the magnitude of the differences

in the observed performance of two different techniques. In our context, considering APFD as the performance measure, the $\widehat{A}_{12}$ statistics measures the probability that a TCP technique X yields higher APFD values than another technique Y. When the two techniques are equivalent, then $\widehat{A}_{12} = 0.5$. If, for example, $\widehat{A}_{12} = 0.7$, then this entails that we would obtain higher APFD results in 70% of the time with technique X compared to technique Y.

5.5. Results and discussion

Figs. 7–10 show the results for the BibTeX2DocBook, UML2ER, CPL2SPL, and Ecore2Maude case studies, respectively.

Concerning the first research question, we want to compare the performance of the random search technique against each of the other three techniques: Greedy, GA, and ACO techniques. When statistically comparing two techniques, the null hypothesis (H_0) states that there is not a statistically significant difference between the APFD results obtained by the two techniques, while the alternative hypothesis (H_1) states that the difference is statistically significant. By applying a proper statistical test, we obtain the p-value which is the probability of obtaining test results at least as extreme as the results actually observed, under the assumption that the null hypothesis is correct. P-values are in the range [0,1], and it is a common convention that p-values under 0.05 represent statistical significant values and hence the null hypothesis can be rejected in such cases.

In all the cases presented except two cases, the p-values were under 0.05 and hence each of Greedy, GA, and ACO significantly outperforms the random search technique. The exception cases are UML2ER ($n = 5, m = 10$) and BibTeX2DocBook ($n = 15, m = 20$). In the UML2ER case (Fig. 8a), all the techniques achieved poor APFD results. The reason is that the test cases in the test suite only cover two rules in the transformation program. In the BibTeX2DocBook case (Fig. 7d), the observation that the differences were not significant can be justified by the nature of the test suite. The BibTeX2DocBook transformation is composed of 9 rules only and each of the test cases covers 5 to 7 rules. In such case, the test cases quickly cover the rules. In fact, the average APRC of the test case orderings returned by the random search technique is greater than 91.04% which is very close to the APRC averages of the other techniques (94.40% for Greedy and 95.19% for GA and ACO).

Concerning the second research question, Table 3 shows the $\widehat{A}_{12}$ statistics for the four case studies. Each cell shows the $\widehat{A}_{12}$ value obtained when comparing the TCP technique in the row against the technique in the column. We use the thresholds suggested by Vargha and Delaney (2000) for interpreting the effect size: 0.5 means no difference; values below 0.5 indicate a small (0.44–0.5), medium (0.36–0.43), large (0.29–0.35), or very large difference (0.0–0.28) in favor of the technique in the column; values above 0.5 indicate a small (0.5–0.56), medium (0.57–0.64), large (0.65–0.71), or very large difference (0.71–1.0) in favor of the technique in the row. In the table, cells are shaded in light gray, gray, and dark gray if the differences are in the medium, large, and very large ranges, respectively. In such cells, and to help the reader, we add the upwards arrow if the results are in favor of the technique in the column and a leftwards arrow if the results are in favor of the technique in the row. The values in bold are those where the hypothesis test indicated statistical differences (*i.e.,* the p-value is less than 0.05).

It can be seen from the table that in the majority of cases there is no significant difference between Greedy, GA, and ACO techniques. In some cases, GA and ACO significantly outperform Greedy with large and very large difference such as CPL2SPL ($n = 5, m = 10$) and Ecore2Maude ($n = 15, m = 10$). In other cases, Greedy significantly outperforms GA and ACO with large and very large difference such as CPL2SPL ($n = 15, m = 20$) and Ecore2Maude ($n = 15, m = 10$).

We can make the following observations concerning the results. Greedy technique is less random in terms of the returned test case

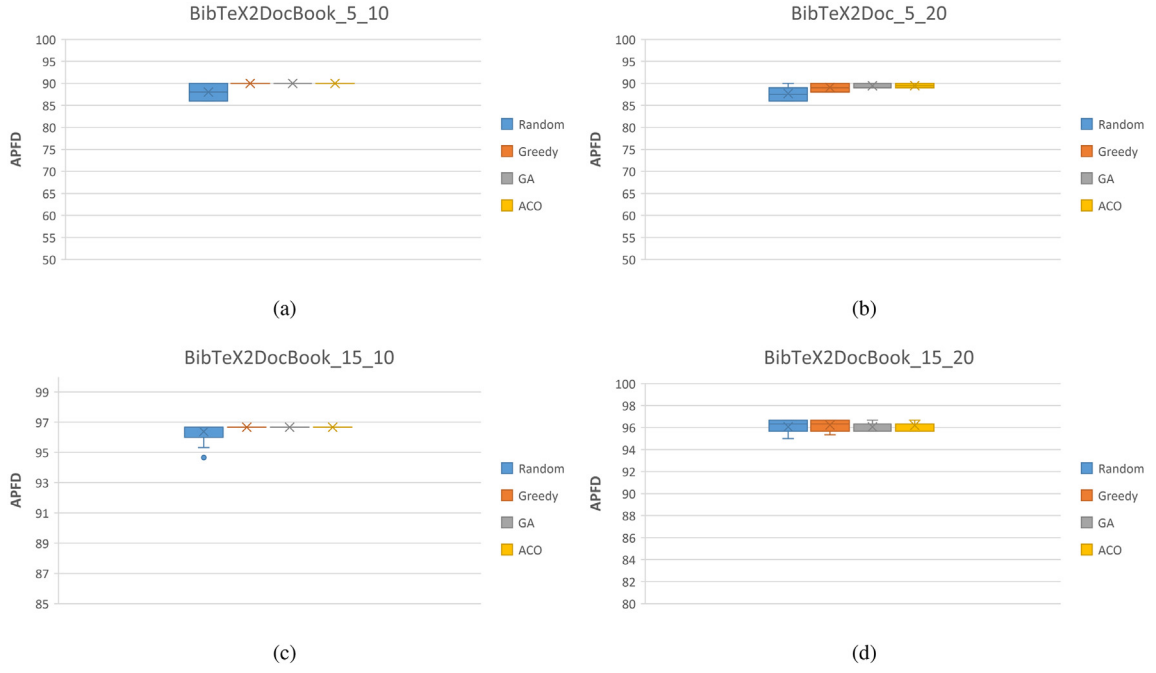


Fig. 7. BibTeX2DocBook results.

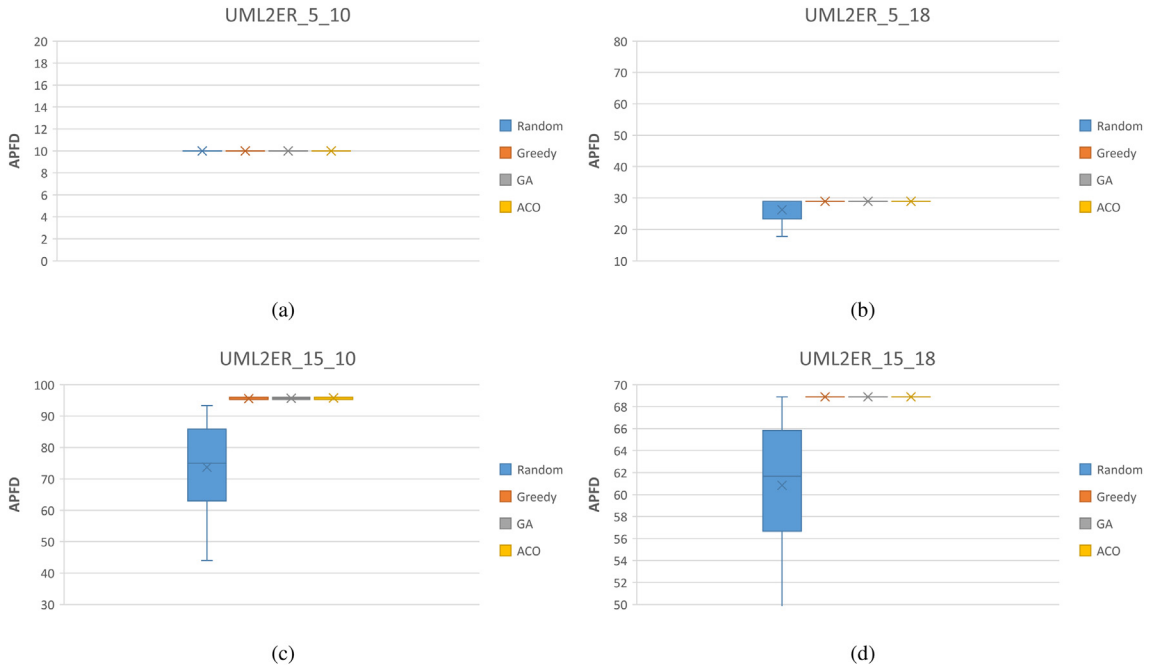


Fig. 8. UML2ER results.

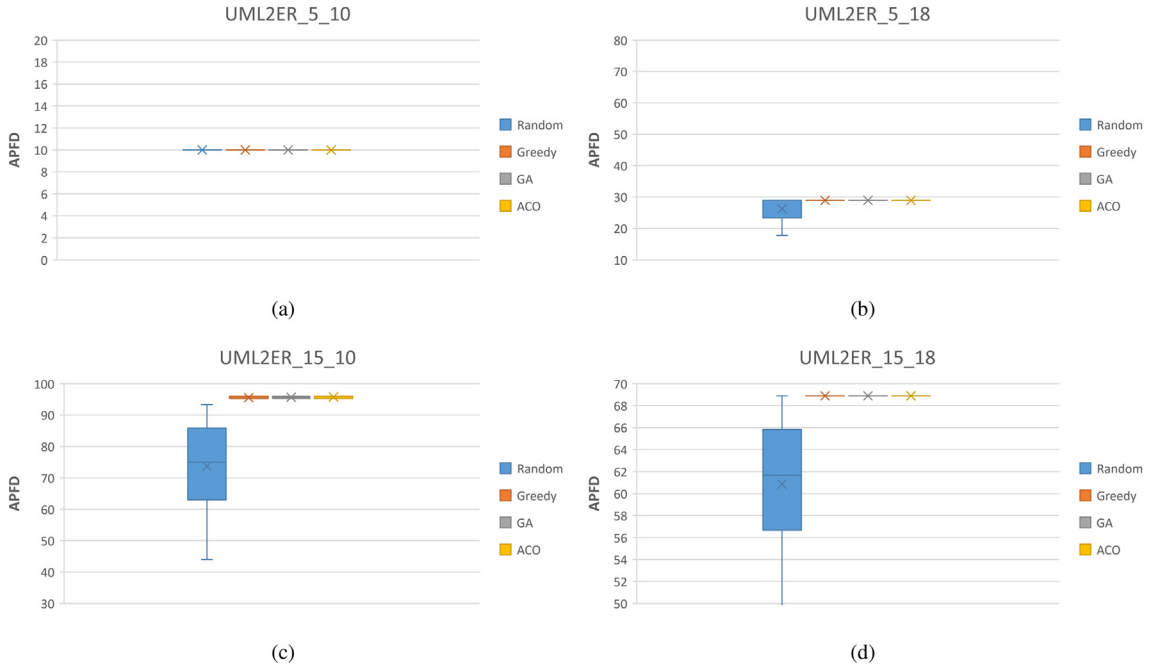


Fig. 9. CPL2SPL results.

orderings than GA and ACO. This explains the small widths of the boxes in the box plots corresponding to Greedy technique. Furthermore, in many cases Greedy finds the test case ordering with the highest APRC compared with GA and ACO. Given that in such orderings it is more likely that the same rule is covered by multiple test cases than in the orderings returned by GA and ACO, this would explain the advantage of Greedy in several of the cases presented above. This is especially true if the size of the test suite (n) is

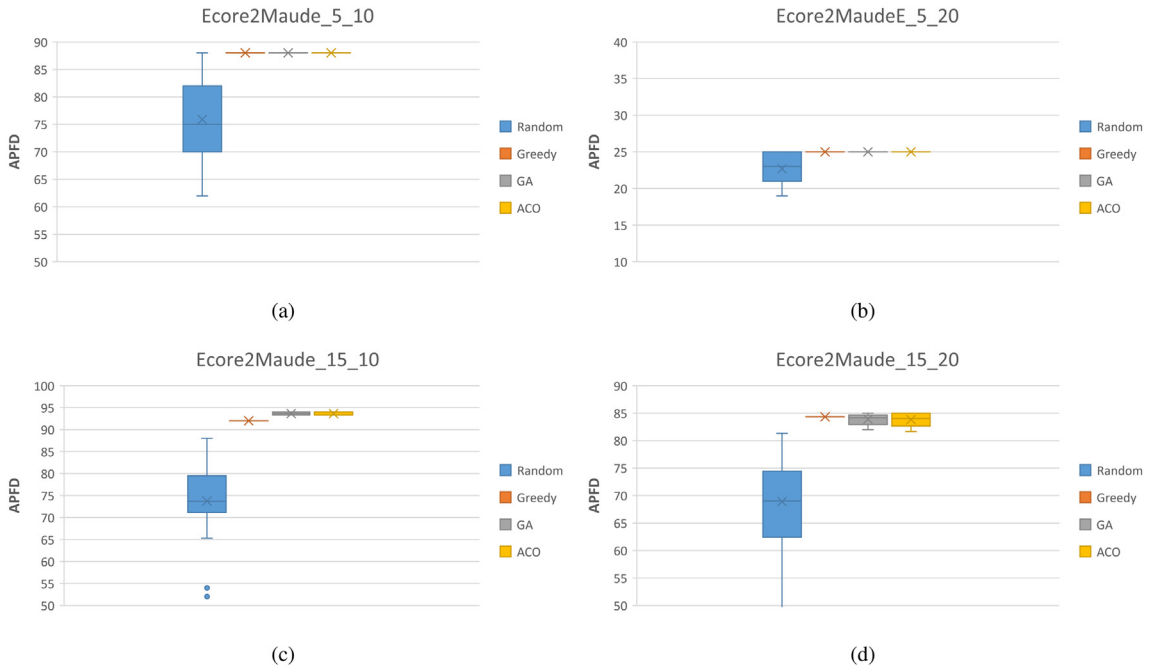


Fig. 10. Ecore2Maude results.

Table 3
$\widehat{A}_{12}$ Statistics for Effect Size Estimation.

BibTeX2DocBook

5_10

	GA	ACO
Greedy	0.500	0.500
GA	0.500	0.500

15_10

	GA	ACO
Greedy	0.500	0.500
GA	0.500	0.500

5_20

	GA	ACO
Greedy	0.387↑	0.375↑
GA	0.500	0.483

15_20

	GA	ACO
Greedy	0.622←	0.559
GA	0.500	0.413↑

UML2ER

5_10

	GA	ACO
Greedy	0.500	0.500
GA	0.500	0.500

15_10

	GA	ACO
Greedy	0.400↑	**0.317↑**
GA	0.500	0.417↑

5_18

	GA	ACO
Greedy	0.500	0.500
GA	0.500	0.500

15_18

	GA	ACO
Greedy	0.500	0.500
GA	0.500	0.500

CPL2SPL

5_10

	GA	ACO
Greedy	**0.14↑**	**0.19↑**
GA	0.500	0.583←

15_10

	GA	ACO
Greedy	**0.817←**	**0.728←**
GA	0.500	0.508

5_20

	GA	ACO
Greedy	0.556	0.622←
GA	0.500	0.548

15_20

	GA	ACO
Greedy	**1.000←**	**1.000←**
GA	0.500	0.503

Ecore2Maude

5_10

	GA	ACO
Greedy	0.500	0.500
GA	0.500	0.500

15_10

	GA	ACO
Greedy	**0.000↑**	**0.000↑**
GA	0.500	0.517

5_20

	GA	ACO
Greedy	0.500	0.500
GA	0.500	0.500

15_20

	GA	ACO
Greedy	0.567←	0.600←
GA	0.500	0.510

big and the number of mutants (m) are big. Nevertheless, Greedy technique may perform too badly in some cases as discussed next.

Table 4 presents the test case coverage information for an arbitrary model transformation testing experiment. There are 15 test cases in the test suite and 10 rules in the model transformation program. We make the following two assumptions. The first one is that each rule has a fault. The second assumption is that if a test case covers a rule (as indicated in the table by the X mark), then the test case discovers (kills) the fault in that rule. We compare Greedy, GA, and ACO techniques in terms of the APFD values of their returned test case orderings.

The results for this arbitrary case show that GA and ACO significantly outperform Greedy. The $\widehat{A}_{12}$ value is 0.0 indicating a very large effect size in favor of GA and ACO. In this case, the average APFD is 48.31% for Greedy and 88.67% for GA and ACO. This can be explained by the nature of the test case coverage shown in Table 4. Greedy would select test cases $T6, \ldots, T15$ before the other test cases, however, these ones only cover four out of 10 rules in

Table 4
Test Case Coverage Information for an Arbitrary Model Transformation Testing Experiment.

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15
R1	X														
R2	X														
R3		X													
R4		X													
R5			X												
R6			X												
R7				X		X	X	X	X	X	X	X	X	X	X
R8				X		X	X	X	X	X	X	X	X	X	X
R9					X	X	X	X	X	X	X	X	X	X	X
R10					X	X	X	X	X	X	X	X	X	X	X

the model transformation. GA and ACO do not suffer from this and in this case produce orderings that cover rules much faster than Greedy resulting in better APFD results. In fact, the APRC averages of the three techniques resemble the APFC averages. The average APRC for Greedy is much smaller than that for GA and ACO. This would explain the poor performance of Greedy in this case.

The main conclusion of the empirical study is that there is no technique that is guaranteed to outperform the other techniques on all model transformation cases. Hence, our recommendation is to obtain the test case orderings of both Greedy and a global search technique such as GA or ACO. In this case, the tester should choose the ordering returned by Greedy technique, unless the APRC of the Greedy ordering is significantly less than the APRC of the ordering returned by GA or ACO. For example, if the APRC of greedy ordering is less than the APRC of GA or ACO ordering by 10 or more, then the tester is advised to use the ordering by GA or ACO. A threshold for what represents a significant difference can be set by the tester. It could depend on several factors, such as the number of rules in the transformation and the size of the test suite. .

5.6. Threats to validity

In the following, we discuss possible threats to the validity of the conclusions of our experiments. There are four basic types of validity threats:

1. *Conclusion Validity:* Threats to the conclusion validity are concerned with factors that affect the ability to draw the correct conclusions based on the observed data. To address this threat, we applied proper statistical tests. We used the Mann–Whitney U-test with a 95% confidence level. All p-values were considered when making any conclusion. When comparing any pair of techniques, we considered APFD measurements for 30 independent runs under each technique. We also reported the effect size measures.
2. *Construct Validity:* This validity is concerned with the relationship between theory and observation. To address this threat, we used APFD which is a well-known metric for evaluating test case prioritization techniques.
3. *Internal Validity:* This validity is concerned with establishing a causal relationship between the treatment (in this case, the application of TCP techniques) and the observed results. Threats to this validity include any disturbing factor that might influence the results. To address this threat, we only used standard libraries and tools for collecting the coverage information for test cases, executing the test cases, and obtaining the test case results. These include OCL Checker (MDTOCLChecker, 2021), ATL Transformation Engine (ATL, 2021), and Eclipse Modeling Framework (EMF, 2021). In the process, we extended the Java framework used in the work of Troya et al. (2018a) to collect the necessary data for our experiments.
4. *External Validity:* This validity is concerned with generalization. In the experiments, we compared several TCP techniques using four transformation programs written in ATL. To model faults, we reused the mutants in the work of Troya et al. (2018a). These mutants had been created by applying different mutation operators that aim to mimic real-world faults made by the transformation programmers. We also reused the test cases created in the work of Troya et al. (2015). Although the empirical study considered several transformation cases with different kinds of mutants, we cannot firmly conclude that the results can be generalized for all model transformations. More experiments are needed in the future involving a wider range of model transformations, including different languages, mutation types, and approaches for creating and defining the test cases.

6. Related work

In this section we present related work in the fields of regression testing for model transformations and test case prioritization model-based testing.

6.1. Regression testing in model transformation

Testing is a crucial step in model-transformation due to the automated nature of transformation of one model to another. Testing can validate the correctness of the target models by checking if the rules have been implemented without errors and the transformation is correct. The crucial nature of testing in efficient model-transformation has been highlighted by a number of researchers (Bryant et al., 2011; Gómez et al., 2012; Van Der Straeten et al., 2008). An automated approach for generating test models based on certain criteria was proposed by Brottier et al. (2006), which was adopted as a foundation approach in several research work (Almendros-Jiménez and Becerra-Terón, 2016; Baudry et al., 2010; Lamari, 2007). Some, however, exploited techniques based on GAs to generate more relevant test cases, such as the work by Gómez et al. (2012), Shelburg et al. (2013), Wang et al. (2013) and Sahin et al. (2015). There are some other researchers who used various techniques to generate efficient test cases for testing the validity of model-transformations such as Vallecillo et al. (2012) who used formal specification based technique, Rose and

Poulding (2013) who exploited probabilistic testing using search, and Shelburg et al. (2013) who used automatic derivation of well-formedness rules.

Regression testing of the model transformation ensures that the progression of the testing process does not harm previously tested correctness of the models. Regression testing is categorized under three classifications, minimization techniques, selection techniques and prioritization techniques. As a minimization technique, Farooq and Lam (2009) reduced the test size of the models by considering test case reduction as a combinatorial optimization problem. Hemmati et al. (2010) proposed a selection technique based on a GA to remove similar test cases from the test sets. The prioritization-based techniques are discussed in the following section.

Model-based regression testing has been implemented on the development of autonomous system by Honfi et al. (2017). They exploited optimization techniques to generate test cases for the regression testing of changed components. A similar application of model-based regression testing is demonstrated by Majzik et al. (2019) on autonomous vehicles. Troya et al. (2018b) also propose a model-based testing method that automatically generates ATL transformations using a technique called metaphoric testing. The technique generates metaphoric relations, which are very much similar to OCL assertions and are used to detect faults in different types of testing including regression testing.

6.2. TCP techniques

TCP techniques have gained a considerable amount of interest and a number of thorough literature surveys have been conducted in this domain, such as the one by Yoo and Harman (2012). The advantage of TCP approaches is that they do not exclude test cases in order to minimize the set, rather they prioritize test cases so that an abrupt end of testing would still yield maximized results. Many approaches based on structural coverage have also been proposed for the TCP (Do et al., 2006; Jeffrey and Gupta, 2005; Zhang et al., 2009). Several code-based TCP techniques have been proposed to prioritize test cases. Elbaum et al. (2004) proposed a technique based on Average Probability of Fault Detection (APFD) for the evaluation of the test cases. Luo et al. (2018) proposed static and dynamic TCP approaches, which are in the form of an open source Java-based system. McMinn and Kapfhammer (2016) developed an Alternating Variable Method (AVM) to prioritize test cases. Pradhan et al. (2016) used a search-based algorithm to minimize the test set based on priority. Arrieta et al. (2016) exploited GA and AVM to prioritize test cases.

In model-based mutation testing, the mutants are introduced to models in order to design optimized and robust test cases. Aichernig and Tappler (2019) have proposed an effective test-selection technique that uses active automata learning technique on Mealy machines. Belli et al. (2016) used model-based techniques primarily based on finite state machines and state charts in order to prioritize test cases. A similar UML-based approach has been proposed by Aichernig et al. (2014) who used activity and state diagrams to prioritize test cases. Another model-based test generation approach has been presented by Samuel et al. (2008) who proposed a predictive AVM that forms full path coverage using fewer test cases. Shin and Lim (2018) also applied state diagrams to generate test cases. They generated test cases in a model-based environment using custom parser with a satisfiability modulo theories (SMT)-based solver. Choi and Lim (2019) have also exploited UML-based techniques to generate test cases for fault localization problems.

7. Conclusion and future work

In this paper, we have presented an approach for test case prioritization in the regression testing of model transformations. The approach exploits the rule coverage information of each test case in the test case suite. In the evaluation, we applied the approach using several techniques on model transformation examples and compared the fault detection effectiveness of these techniques. We also presented a tool that implements the proposed techniques and can automatically generate test case orderings, and thus facilitate the job of a tester during the regression testing of model transformations. The major contribution of the paper is an approach to apply test case prioritization especially in the regression testing of model transformations. The secondary contribution is the provision of a tool that helps in automatic test case prioritization based on the proposed approach.

There are several avenues for future research. First, the proposed approach and tool need to be integrated with existing model transformation tools and technologies. For instance, this can be realized by extending an Eclipse plug-in for model-based development to incorporate our tool and hence be of a useful feature to testers of model transformations. Also, it is recommended to apply the proposed approach to industrial case studies involving larger and more complex model transformations. Furthermore, this paper uses APFD as a performance metric to evaluate the TCP techniques. It is of interest to explore other metrics that have been used in the regression testing of conventional software systems. Several such metrics are highlighted by Hemmati (2019) and Lou et al. (2019). For example, the cost of each test case execution and the severity of each fault can be taken into account by the performance evaluation metric. In this case, the TCP techniques need to be adapted to achieve better performance on these new metrics.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Appendix A. ATL BibTeX2DocBook code

APPENDIX: ATL BIBTEX2DOCBOOK CODE

```
module BibTeX2DocBook;
create OUT : DocBook from IN : BibTeX;

helper def: authorSet : Sequence(BibTeX!Author) = BibTeX!Author.allInstances()->
  iterate(e; ret : Sequence(BibTeX!Author) = Sequence {} |
  if ret->collect(e | e.author)->includes(e.author) then ret
  else ret->including(e) endif
  )->sortedBy(e | e.author);

helper def: titledEntrySet : Sequence(BibTeX!TitledEntry) = BibTeX!TitledEntry.allInstances()->
  iterate(e; ret : Sequence(BibTeX!TitledEntry) = Sequence {} |
  if ret->collect(e | e.title)->includes(e.title) then ret
  else ret->including(e) endif
  )->sortedBy(e | e.title);

helper def: articleSet : Sequence(BibTeX!Article) = BibTeX!Article.allInstances()->
  iterate(e; ret : Sequence(BibTeX!Article) = Sequence {} |
  if ret->collect(e | e.journal)->includes(e.journal) then ret
  else ret->including(e) endif
  )->sortedBy(e | e.journal);

helper context BibTeX!BibTeXEntry def: buildEntryPara() : String =
  '[' + self.id + ']'
  + ' ' + self.oclType().name
  + (if self.oclIsKindOf(BibTeX!TitledEntry) then ' ' + self.title else  endif)
  + (if self.oclIsKindOf(BibTeX!AuthoredEntry)
  then self.authors->iterate(e; str : String =  | str + ' ' + e.author)
  else
  endif)
  + (if self.oclIsKindOf(BibTeX!DatedEntry) then ' ' + self.year else  endif)
  + (if self.oclIsKindOf(BibTeX!BookTitledEntry) then ' ' + self.booktitle else  endif)
  + (if self.oclIsKindOf(BibTeX!ThesisEntry) then ' ' + self.school else  endif)
  + (if self.oclIsKindOf(BibTeX!Article) then ' ' + self.journal else  endif)
  + (if self.oclIsKindOf(BibTeX!Unpublished) then ' ' + self.note else  endif)
  + (if self.oclIsKindOf(BibTeX!Book) then ' ' + self.publisher else  endif)
  + (if self.oclIsKindOf(BibTeX!InBook) then ' ' + self.chapter.toString() else  endif);


rule Main {
  from bib : BibTeX!BibTeXFile
  to doc : DocBook!DocBook (books <- boo),
     boo : DocBook!Book (articles <- art),
     art : DocBook!Article (
     title <- 'BibTeXML to DocBook',
     sections_1 <- Sequence{se1, se2, se3, se4},
     se1 : DocBook!Sect1 (
       title <- 'References List',
       paras <- BibTeX!BibTeXEntry.allInstances()->sortedBy(e | e.id)),
     se2 : DocBook!Sect1 (
       title <- 'Authors list',
       paras <- thisModule.authorSet),
     se3 : DocBook!Sect1 (
       title <- 'Titles List',
       paras <- thisModule.titledEntrySet->collect(e | thisModule.resolveTemp(e, 'title_para'))),
     se4 : DocBook!Sect1 (
       title <- 'Journals List',
       paras <- thisModule.articleSet->collect(e | thisModule.resolveTemp(e, 'journal_para')))
}


rule Author {
  from a : BibTeX!Author (thisModule.authorSet->includes(a))
  to p1 : DocBook!Para (content <- a.author)
}


rule UntitledEntry {
  from e : BibTeX!BibTeXEntry (not e.oclIsKindOf(BibTeX!TitledEntry))
  to p : DocBook!Para (content <- e.buildEntryPara())
}


rule TitledEntry_Title_NoArticle {
  from e : BibTeX!TitledEntry (
    thisModule.titledEntrySet->includes(e) and
    not e.oclIsKindOf(BibTeX!Article))
  to entry_para : DocBook!Para (content <- e.buildEntryPara()),
     title_para : DocBook!Para (content <- e.title)
}


rule TitledEntry_NoTitle_NoArticle {
  from e : BibTeX!TitledEntry (
    not thisModule.titledEntrySet->includes(e) and
    not e.oclIsKindOf(BibTeX!Article))
  to entry_para : DocBook!Para (content <- e.buildEntryPara())
}


rule Article_Title_Journal {
  from e : BibTeX!Article (
    thisModule.titledEntrySet->includes(e) and
    thisModule.articleSet->includes(e))
  to entry_para : DocBook!Para (content <- e.buildEntryPara()),
     title_para : DocBook!Para (content <- e.title),
     journal_para : DocBook!Para (content <- e.journal)
}


rule Article_NoTitle_Journal {
  from e : BibTeX!Article (
    not thisModule.titledEntrySet->includes(e) and
    thisModule.articleSet->includes(e))
  to entry_para : DocBook!Para (content <- e.buildEntryPara()),
     journal_para : DocBook!Para (content <- e.journal)
}

rule Article_Title_NoJournal {
  from e : BibTeX!Article (
    thisModule.titledEntrySet->includes(e) and
    not thisModule.articleSet->includes(e))
  to entry_para : DocBook!Para (content <- e.buildEntryPara()),
     title_para : DocBook!Para (content <- e.title)
}

rule Article_NoTitle_NoJournal {
  from e : BibTeX!Article (
    not thisModule.titledEntrySet->includes(e) and
    not thisModule.articleSet->includes(e))
  to entry_para : DocBook!Para (content <- e.buildEntryPara())
}
```

References

Aichernig, B.K., Auer, J., Jöbstl, E., Korošec, R., Krenn, W., Schlick, R., Schmidt, B.V., 2014. Model-based mutation testing of an industrial measurement device. In: Proceedings of the International Conference on Tests and Proofs, pp. 1–19.

Aichernig, B.K., Tappler, M., 2019. Efficient active automata learning via mutation testing. J. Automated Reason. 63, 1103–1134.

Alkhazi, B., Abid, C., Kessentini, M., Leroy, D., Wimmer, M., 2020. Multi-criteria test cases selection for model transformations. Automated Software Eng. 27, 91–118. https://doi.org/10.1007/s10515-020-00271-w.

Almendros-Jiménez, J.M., Becerra-Terón, A., 2016. Automatic generation of Ecore models for testing ATL transformations. In: Proceedings of the International Conference on Model and Data Engineering, pp. 16–30.

Arcuri, A., Briand, L., 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of the International Conference on Software Engineering, pp. 1–10. https://doi.org/10.1145/1985793.1985795.

Arrieta, A., Wang, S., Sagardui, G., Etxeberria, L., 2016. Test case prioritization of configurable cyber-physical systems with weight-based search algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1053–1060.

Atenea, Atenea BibTeXML2DocBook. http://atenea.lcc.uma.es/projects/MTB/BibTeXML2DocBook.html (Accessed March 2021)..

ATL, ATL. https://www.eclipse.org/atl/ (Accessed March 2021)..

ATLTransformationExample, ATL Transformation Example: BibTeXML to DocBook. https://www.eclipse.org/atl/atlTransformations/BibTeXML2DocBook/ExampleBibTeXML2DocBook[v00.01].pdf (Accessed March 2021)..

ATLZoo, ATL Transformations Zoo.https://www.eclipse.org/atl/atlTransformations/ (Accessed March 2021)..

Baudry, B., Ghosh, S., Fleurey, F., France, R., Le Traon, Y., Mottu, J.M., 2010. Barriers to systematic model transformation testing. Commun. ACM 53, 139–143.

Belli, F., Budnik, C.J., Hollmann, A., Tuglular, T., Wong, W.E., 2016. Model-based mutation testing—approach and case studies. Sci. Comput. Program. 120, 25–48.

Blum, C., 2005. Ant colony optimization: introduction and recent trends. Phys. Life Rev. 2, 353–373. https://doi.org/10.1016/j.plrev.2005.10.001.

Brambilla, M., Cabot, J., Wimmer, M., 2017. Model-Driven Software Engineering in Practice. Morgan & Claypool.

Brottier, E., Fleurey, F., Steel, J., Baudry, B., Le Traon, Y., 2006. Metamodel-based test generation for model transformations: an algorithm and a tool. In: Proceedings of the International Symposium on Software Reliability Engineering, pp. 85–94.

Brottier, E., Fleurey, F., Steel, J., Baudry, B., Traon, Y., 2006. Metamodel-based test generation for model transformations: an algorithm and a tool. In: Proceedings of the International Symposium on Software Reliability Engineering, pp. 85–94. https://doi.org/10.1109/ISSRE.2006.27.

Bryant, B., Gray, J., Mernik, M., Clarke, P., France, R., Karsai, G., 2011. Challenges and directions in formalizing the semantics of modeling languages. Comput. Sci. Inf. Syst. 8, 225–253. https://doi.org/10.2298/csis110114012b.

Chittimalli, P.K., Harrold, M.J., 2009. Recomputing coverage information to assist regression testing. IEEE Trans. Software Eng. 35, 452–469. https://doi.org/10.1109/TSE.2009.4.

Choi, Y.M., Lim, D.J., 2019. Model-based test suite generation using mutation analysis for fault localization. Appl. Sci. 9, 3492.

Cicchetti, A., Di Ruscio, D., Eramo, R., Pierantonio, A., 2011. JTL: a bidirectional and change propagating transformation language. In: Malloy, B., Staab, S., van den Brand, M. (Eds.), Software Language Engineering. Springer, pp. 183–202.

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L. (Eds.), 2007. All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic. Springer. doi: 10.1007/978-3-540-71999-1..

Csertan, G., Huszerl, G., Majzik, I., Pap, Z., Pataricza, A., Varro, D., 2002. VIATRA – visual automated transformations for formal verification and validation of UML models. In: Proceedings of the International Conference on Automated Software Engineering, pp. 267–270. https://doi.org/10.1109/ASE.2002.1115027.

Czarnecki, K., Helsen, S., 2006. Feature-based survey of model transformation approaches. IBM Syst. J. 45, 621–645. https://doi.org/10.1147/sj.453.0621.

Do, H., Rothermel, G., Kinneer, A., 2006. Prioritizing JUnit test cases: an empirical assessment and cost-benefits analysis. Empirical Software Eng. 11, 33–70.

Dorigo, M., Maniezzo, V., Colorni, A., 1996. Ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybern. Part B 26, 29–41. https://doi.org/10.1109/3477.484436.

Elbaum, S., Rothermel, G., Kanduri, S., Malishevsky, A.G., 2004. Selecting a cost-effective test case prioritization technique. Software Qual. J. 12, 185–210.

Elbaum, S., Rothermel, G., Penix, J., 2014. Techniques for improving regression testing in continuous integration development environments. In: Proceedings of the International Symposium on Foundations of Software Engineering, pp. 235–245. https://doi.org/10.1145/2635868.2635910.

EMF, Eclipse Modeling Framework. https://www.eclipse.org/modeling/emf/ (Accessed March 2021)..

Farooq, U., Lam, C.P., 2009. Evolving the quality of a model based test suite. In: Proceedings of the International Conference on Software Testing, and Validation, pp. 141–149.

Gavidia-Calderon, C., Castañon, C.B., 2020. Isula: a java framework for ant colony algorithms. SoftwareX 11. https://doi.org/10.1016/j.softx.2020.100400.

Gómez, J.J.C., Baudry, B., Sahraoui, H., 2012. Searching the boundaries of a modeling space to test metamodels. In: Proceedings of the International Conference on Software Testing, Verification and Validation, pp. 131–140.

Greenyer, J., Kindler, E., 2010. Comparing relational model transformation technologies: implementing query/view/transformation with triple graph grammars. Software Syst. Model. 9, 21–46. https://doi.org/10.1007/s10270-009-0121-8.

Harrold, M.J., Gupta, R., Soffa, M.L., 1993. A methodology for controlling the size of a test suite. ACM Trans. Software Eng. Methodol. 2, 270–285. https://doi.org/10.1145/152388.152391.

Harrold, M.J., Jones, J.A., Li, T., Liang, D., Orso, A., Pennings, M., Sinha, S., Spoon, S.A., Gujarathi, A., 2001. Regression test selection for Java software. ACM SIGPLAN Notices 36, 312–326. https://doi.org/10.1145/504311.504305.

Hemmati, H., 2019. Chapter four - advances in techniques for test prioritization, in: Memon, A.M. (Ed.), Advances in Computers. Elsevier. volume 112, pp. 185–221. doi: 10.1016/bs.adcom.2017.12.004..

Hemmati, H., Briand, L., Arcuri, A., Ali, S., 2010. An enhanced test case selection approach for model-based testing: an industrial case study. In: Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 267–276.

Hollander, M., Wolfe, D.A., Chicken, E., 2013. Nonparametric Statistical Methods. Wiley.

Honfi, D., Molnár, G., Micskei, Z., Majzik, I., 2017. Model-based regression testing of autonomous robots, in: Proceedings of the International System Design Language Forum, pp. 119–135. doi: 10.1007/978-3-319-68015-6_8..

IEEEStd2010, 2010. ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary. ISO/IEC/IEEE 24765:2010(E), 1–418doi: 10.1109/IEEESTD.2010.5733835..

ImplementationSpectrumBased, Implementation of the Spectrum-Based Fault Localization in Model Transformations. https://github.com/javitroya/SBFL_MT (Accessed March 2021)..

Jacobson, L., Kanber, B., 2015. Genetic Algorithms in Java Basics. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-0328-6.

Jeffrey, D., Gupta, N., 2005. Test suite reduction with selective redundancy. In: Proceedings of the International Conference on Software Maintenance, pp. 549–558.

Jézéquel, J.M., Barais, O., Fleurey, F., 2011. Model driven language engineering with kermeta, in: Fernandes, J.M., Lämmel, R., Visser, J., Saraiva, J. (Eds.), Generative and Transformational Techniques in Software Engineering III: International Summer School, GTTSE 2009, Braga, Portugal, July 6–11, 2009. Revised Papers, Springer. pp. 201–221. doi: 10.1007/978-3-642-18023-1_5..

Jouault, F., 2005. Loosely coupled traceability for ATL, in: Proceedings of the European Conference on Model Driven Architecture Workshop on Traceability, pp. 29–37..

Jouault, F., Bézivin, J., Consel, C., Kurtev, I., Latry, F., 2006. Building DSLs with AMMA/ATL, a Case Study on SPL and CPL Telephony Languages, in: Proceedings of the ECOOP Workshop on Domain-Specific Program Development..

Jouault, F., Kurtev, I., 2006. Transforming models with ATL. In: Bruel, J.M. (Ed.), Satellite Events at the MoDELS 2005 Conference. Springer, pp. 128–138.

Kelly, S., Tolvanen, J.P., 2008. Domain-Specific Modeling: Enabling Full Code Generation. Wiley.

Korel, B., Tahat, L.H., Vaysburg, B., 2002. Model based regression test reduction using dependence analysis. In: Proceedings of the International Conference on Software Maintenance, pp. 214–223. https://doi.org/10.1109/ICSM.2002.1167768.

Lamari, M., 2007. Towards an automated test generation for the verification of model transformations. In: Proceedings of the ACM symposium on Applied computing, pp. 998–1005.

Lara, J.d., Vangheluwe, H., 2002. AToM^3: A tool for multi-formalism and meta-modelling, in: Kutsche, R.D., Weber, H. (Eds.), Fundamental Approaches to Software Engineering, Springer. pp. 174–188..

Lou, Y., Chen, J., Zhang, L., Hao, D., 2019. Chapter one - A survey on regression test-case prioritization, in: Memon, A.M. (Ed.), Advances in Computers. Elsevier. volume 113, pp. 1–46. doi: 10.1016/bs.adcom.2018.10.001..

Lúcio, L., Amrani, M., Dingel, J., Lambers, L., Salay, R., Selim, G.M., Syriani, E., Wimmer, M., 2016. Model transformation intents and their properties. Software Syst. Model. 15, 647–684. https://doi.org/10.1007/s10270-014-0429-x.

Luo, Q., Moran, K., Zhang, L., Poshyvanyk, D., 2018. How do static and dynamic test case prioritization techniques perform on modern software systems? an extensive study on github projects. IEEE Trans. Software Eng. 45, 1054–1080.

Majzik, I., Semeráth, O., Hajdu, C., Marussy, K., Szatmári, Z., Micskei, Z., Vörös, A., Babikian, A.A., Varró, D., 2019. Towards system-level testing with coverage guarantees for autonomous vehicles. In: Proceedings of the International Conference on Model Driven Engineering Languages and Systems, pp. 89–94. https://doi.org/10.1109/MODELS.2019.00-12.

Malishevsky, A.G., Ruthruff, J.R., Rothermel, G., Elbaum, S., 2006. Cost-cognizant test case prioritization. Technical Report TR-UNL-CSE-2006-0004. University of Nebraska-Lincoln.

Marijan, D., Gotlieb, A., Sen, S., 2013. Test case prioritization for continuous regression testing: an industrial case study. In: Proceedings of the International Conference on Software Maintenance, pp. 540–543. https://doi.org/10.1109/ICSM.2013.91.

Mazón, J.N., Trujillo, J., 2007. A model driven modernization approach for automatically deriving multidimensional models in data warehouses. In: Parent, C., Schewe, K.D., Storey, V.C., Thalheim, B. (Eds.), Conceptual Modeling – ER 2007. Springer, pp. 56–71.

McMinn, P., Kapfhammer, G.M., 2016. AVMf: An open-source framework and implementation of the Alternating Variable Method. In: Proceedings of the International Symposium on Search Based Software Engineering, pp. 259–266.

MDTOCLChecker, MDT OCL Checker. https://wiki.eclipse.org/MDT_OCL/Ocl_Checker (Accessed March 2021)..

Moreno-Delgado, A., Durán, F., Zschaler, S., Troya, J., 2014. Modular DSLs for flexible analysis: an e-Motions reimplementation of Palladio, in. In: Proceedings of the European Conference on Modelling Foundations and Applications, pp. 132–147. https://doi.org/10.1007/978-3-319-09195-2_9.

Mottu, J., Baudry, B., Traon, Y.L., 2008. Model transformation testing: oracle issue. In: Proceedings of the International Conference on Software Testing Verification and Validation Workshop, pp. 105–112. https://doi.org/10.1109/ICSTW.2008.27.

Mussbacher, G., Amyot, D., Breu, R., Bruel, J.M., Cheng, B.H.C., Collet, P., Combemale, B., France, R.B., Heldal, R., Hill, J., Kienzle, J., Schöttle, M., Steimann, F., Stikkolorum, D., Whittle, J., 2014. The relevance of model-driven engineering thirty years from now. In: Dingel, J., Schulte, W., Ramos, I., Abrahão, S., Insfran, E. (Eds.), Model-Driven Engineering Languages and Systems. Springer, pp. 183–200.

Pradhan, D., Wang, S., Ali, S., Yue, T., 2016. Search-based cost-effective test case selection within a time budget: an empirical study. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1085–1092.

R Core Team, 2019. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing. Vienna, Austria. URL: https://www.R-project.org/..

Ramón, Ó.S., Cuadrado, J.S., Molina, J.G., 2014. Model-driven reverse engineering of legacy graphical user interfaces. Automated Software Eng. 21, 147–186. https://doi.org/10.1007/s10515-013-0130-2.

Rose, L.M., Poulding, S., 2013. Efficient probabilistic testing of model transformations using search. In: Proceedings of the International Workshop on Combining Modelling and Search-Based Software Engineering, pp. 16–21.

Rothermel, G., Harrold, M.J., 1994. Selecting regression tests for object-oriented software. In: Proceedings of the International Conference on Software Maintenance, pp. 14–25. https://doi.org/10.1109/ICSM.1994.336793.

Rothermel, G., Untch, R.J., Chu, C., 2001. Prioritizing test cases for regression testing. IEEE Trans. Software Eng. 27, 929–948. https://doi.org/10.1109/32.962562.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., 1991. Object-Oriented Modeling and Design. Prentice-Hall, USA.

Sahin, D., Kessentini, M., Wimmer, M., Deb, K., 2015. Model transformation testing: a bi-level search-based software engineering approach. J. Software: Evol. Process 27, 821–837.

Samuel, P., Mall, R., Bothra, A.K., 2008. Automatic test case generation using unified modeling language (uml) state diagrams. IET software 2, 79–93.

Sendall, S., Kozaczynski, W., 2003. Model transformation: the heart and soul of model-driven software development. IEEE Softw. 20, 42–45. https://doi.org/10.1109/MS.2003.1231150.

Shelburg, J., Kessentini, M., Tauritz, D.R., 2013. Regression testing for model transformations: a multi-objective approach. In: Proceedings of the International Symposium on Search Based Software Engineering, pp. 209–223.

Shin, K.W., Lim, D.J., 2018. Model-based automatic test case generation for automotive embedded software testing. Int. J. Automotive Technol. 19, 107–119.

Srivastava, A., Thiagarajan, J., 2002. Effectively prioritizing tests in development environment, in: Proceedings of the International Symposium on Software Testing and Analysis, pp. 97–106. doi: 10.1145/566172.566187..

Taentzer, G., 2004. AGG: a graph transformation environment for modeling and validation of software. In: Pfaltz, J.L., Nagl, M., Böhlen, B. (Eds.), Applications of Graph Transformations with Industrial Relevance. Springer, pp. 446–453.

Talbi, E., 2009. Metaheuristics – From Design to Implementation. Wiley.

Troya, J., Bergmayr, A., Burgueño, L., Wimmer, M., 2015. Towards systematic mutations for and with ATL model transformations. In: Proceedings of the International Conference on Software Testing, Verification and Validation Workshops, pp. 1–10.

Troya, J., Segura, S., Parejo, J.A., Ruiz-Cortés, A., 2018a. Spectrum-based fault localization in model transformations. ACM Trans. Software Eng. Methodol. 27. https://doi.org/10.1145/3241744.

Troya, J., Segura, S., Ruiz-Cortés, A., 2018b. Automated inference of likely metamorphic relations for model transformations. J. Syst. Software 136, 188–208. https://doi.org/10.1016/j.jss.2017.05.043.

Troya, J., Vallecillo, A., 2014. Specification and simulation of queuing network models using domain-specific languages. Comput. Stand. Interfaces 36, 863–879. https://doi.org/10.1016/j.csi.2014.01.002.

Vallecillo, A., Gogolla, M., Burgueño, L., Wimmer, M., Hamann, L., 2012. Formal specification and testing of model transformations, in: Bernardo, M., Cortellessa, V., Pierantonio, A. (Eds.), Formal Methods for Model-Driven Engineering: 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2012, Bertinoro, Italy, June 18–23, 2012. Advanced Lectures, Springer, Berlin Heidelberg. pp. 399–437. doi: 10.1007/978-3-642-30982-3_11..

Van Der Straeten, R., Mens, T., Van Baelen, S., 2008. Challenges in model-driven software engineering. In: Proceedings of the International Conference on Model Driven Engineering Languages and Systems, pp. 35–47.

Vargha, A., Delaney, H.D., 2000. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. J. Educ. Behav. Stat. 25, 101–132. https://doi.org/10.3102/10769986025002101.

Wang, W., Kessentini, M., Jiang, W., 2013. Test cases generation for model transformations from structural information. In: Proceedings of the International Conference on Model Driven Engineering Languages and Systems, pp. 42–51.

Yoo, S., Harman, M., 2012. Regression testing minimization, selection and prioritization: a survey. Software Test. Verification Reliab. 22, 67–120. https://doi.org/10.1002/stv.430.

Yoo, S., Harman, M., 2012. Regression testing minimization, selection and prioritization: a survey. Software Test. Verification Reliab. 22, 67–120.

Zhang, J., Lou, Y., Zhang, L., Hao, D., Zhang, L., Mei, H., 2016. Isomorphic regression testing: executing uncovered branches without test augmentation. In: Proceedings of the International Symposium on Foundations of Software Engineering, pp. 883–894. https://doi.org/10.1145/2950290.2950313.

Zhang, L., Hou, S.S., Guo, C., Xie, T., Mei, H., 2009. Time-aware test-case prioritization using integer linear programming. In: Proceedings of the International Symposium on Software Testing and Analysis, pp. 213–224.