Test Cases Generation and Optimization Using Ant Colony and Decision Tree Algorithm

Manisha Verma, Priya Arora
1Research Scholar, 2Assistant Professor,
1, 2Baddi University of Emerging Science and Technologies Baddi, Himachal Pradesh, India

Abstract: UML stands for Unified Modeling Language. It’s a widely-used modeling language in the field of software engineering. Experts use UML to analyze, design, and implement software-based systems, along with other business processes. The name says it all—Unified Modeling Language is a modeling language that combines various approaches in a single design language, which is used to plan and create computer applications. UML is actually a combination of several notations: Object-Oriented Design, Object Modeling Technique, and Object-Oriented Software Engineering. The Unified Modeling Language uses the strengths of these approaches to present a more consistent methodology that’s easier to use. UML represents best practices for building and documenting the facets of software and business system modeling.

Keywords: UML, XMI, AFT, CCG

I. INTRODUCTION

Software testing is a process of executing a program or application with the intent of finding the software bugs. It can also be stated as the process of validating and verifying that a software program or application or product: Meets the business and technical requirements that guided it's design and development. In software testing process here we examine the efficiency of developed software means is it according to user requirement. IT company spend 50% of time, cost, efforts on testing module. It's more difficult, challenging, labour intensive and time consuming task. In testing process here we compare the actual results with expected result to get how efficient it is.

II. WHAT IS TESTING?

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements. According to ANSI/IEEE 1059 standard, Testing can be defined as: A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/ errors/ bugs) and to evaluate the features of the software item.

III. LITERATURE SURVEY

1. P. Samuel R. Mall A.K. Bothra UML is widely accepted and used by industry for modelling and design of software systems. A novel method to automatically generate test cases based on UML state models is presented. In the present approach, the control and data flow logic available in the UML state diagram to generate test data are exploited. The state machine graph is traversed and the conditional predicates on every transition are selected. Then these conditional predicates are transformed and function minimisation technique is applied to generate test cases. The present test data generation scheme is fully automatic and the generated test cases satisfy transition path coverage criteria. The generated test cases can be used to test class as well as cluster-level state-dependent behaviours.

2. Y.G.Kim, H.S.Hong, D.H.Bae and S.D.Cha The paper discusses the application of state diagrams in UML to class testing. A set of coverage criteria is proposed based on control and data flow in UML state diagrams and it is shown how to generate test cases satisfying these criteria from UML state diagrams. First, control flow is identified by transforming UML state diagrams into extended finite state machines (EFSMs). The hierarchical and concurrent structure of states is flattened and broadcast commu- nications are eliminated in the resulting EFSMs. Second, data flow is identified by transforming EFSMs into flow graphs to which conventional data flow analysis techniques can be applied.

3. Anneliese von Mayrhauser, Senior Member, Black-box test-generation requires a model of the system under test to describe what is to be tested. Testing criteria and test objectives define how it is to be tested. This paper describes an approach to black-box test-generation in which an AI (artificial intelligence) planner is used to generate test cases from test objectives derived from UML (Unified Modeling Language) Class Diagrams. The UML Class Diagrams are conceptual models of the systems under test. They differ from traditional design and requirements models in that they include information pertinent to test case generation.

4. Vahid Garousi, Member, IEEE Genetic algorithms (GAs) have been applied previously to UML-driven stress test requirements generation with the aim of increasing chances of discovering faults relating to network traffic in distributed...
real-time systems. However, since evolutionary algorithms are heuristic, their performance can vary across multiple executions, which may affect robustness and scalability. To address this, we present the design and technical detail of a UML-driven, GA-based stress test requirements generation tool, together with its empirical analysis. The main goal is to analyze and improve the applicability, efficiency, and effectiveness and also to validate the design choices of the GA used in the tool. Findings of the empirical evaluation reveal that the tool is robust and reasonably scalable when it is executed on large-scale experimental design models. The study also reveals the main bottlenecks and limitations of the tools, e.g., there is a performance bottleneck when the system under test has a large number of sequence diagrams which could be triggered independently from each other. In addition, issues specific to stress testing, e.g., the impact of variations in task arrival pattern types, reveal that the tool generally generates effective test requirements, although the features of those test requirements might be different in different runs (e.g., different stress times from the test start time might be chosen). While the use of evolutionary algorithms to generate software test cases has been widely reported, the extent, depth, and detail of the empirical findings presented in this paper are novel and suggest that the proposed approach is effective and efficient in generating stress test requirements. It is hoped that the findings of this empirical study will help other SBSE researchers with the empirical evaluation of their own techniques and tools.

5. Debasish Kundu1, Debasis Samanta1, Rajib Unified modelling language (UML) is a visual modelling language, which has gained popularity among software practitioners. In a model-driven software development environment, the existing UML tools mainly support automatic generation of structural code from UML class diagrams. However, the code generation from UML diagrams such as statechart, activity, collaboration and sequence diagrams (SDs) are not supported by most UML tools and also have scarcely been reported in the literatures. This work proposes an approach to automatic generation of code from UML 2.x SDs of use cases. From the XML metadata interchange (XMI) representation of an SD of a use case, the authors construct a graph model called sequence integration graph (SIG). The SIG encapsulates information related to messages, control flow and method scope of interactions. These information are then used to generate code. The proposed approach has been tested using a number of real-life application systems and the results substantiate the efficacy of the approach to synthesise the code for controller classes. The authors observe that approximately 48% of the total lines of code within controller class methods can be generated with the proposed approach. The proposed approach can be easily extended to other behavioural UML models such as interaction-overview diagrams, communication diagrams and activity diagrams.

6. Simon Pickin, Claude Jard, Thierry Je ron , Jean-Marc Je ze quel, Member, IEEE, and Yves Le Traon The object-oriented software development process is increasingly used for the construction of complex distributed systems. In this context, behavior models have long been recognized as the basis for systematic approaches to requirements capture, specification, design, simulation, code generation, testing, and verification. Two complementary approaches for modeling behavior have proven useful in practice: interaction-based modeling (e.g., UML sequence diagrams) and state-based modeling (e.g., UML statecharts). Building on formal V&V techniques, in this article we present a method and a tool for automated synthesis of test cases from scenarios and a state-based design model of the application, remaining entirely within the UML framework. The underlying “on the fly” test synthesis algorithms are based on the input/output labeled transition system formalism, which is particularly appropriate for modeling applications involving asynchronous communication. The method is eminently compatible with classical OO development processes since it can be used to synthesize test cases from the scenarios used in early development stages to model global interactions between actors and components, instead of these test cases being derived manually. We illustrate the system test synthesis process using an air traffic control software example.

7. Ajay Kumar Jena, Santosh Kumar: Software testing approaches are mainly divided into three types i.e. code based testing, specification based testing and model based testing. In model based testing, the testing can be started from the design process at the beginning phase. So, early detection of faults can be achieved by using this approach. An approach for the generation of test cases from UML (Unified Modelling Language) activity diagram using genetic algorithm has been presented in this paper. Early detection of faults can be achieved by this approach and will certainly reduce the time, cost and effort of the developer to a large extent. We propose a model to generate an Activity Flow Table (AFT) from an Activity Diagram and then convert it to Activity Flow Graph (AFG). Coverage criteria are very important in test case generation. By using activity coverage criterion we traverse the AFG and the test paths are generated. Finally, we generate the test cases from these paths. Genetic Algorithm has been applied to generate test cases and also to optimise them. The model is implemented on a case study of ATM Withdrawal system.

8. Hyungchoul Kim, Sungwon Kang, Jongmoon Baik UML activity diagram is a notation suitable for modeling a concurrent system in which multiple objects interact with each other. This paper proposes a method to generate test cases from UML activity diagrams that minimizes the number of test cases generated while deriving all practically useful test cases. Our method first builds an I/O explicit Activity Diagram from an ordinary UML activity diagram and then transforms it to a directed graph, from which test cases for the initial activity diagram are derived. This conversion is performed based on the single stimulus principle, which helps avoid the state explosion problem in test generation for a concurrent system.

9. Durga Prasad Mohapatra, Bhubaneswar Testing concurrency is difficult yet important. Because of arbitrary interference of concurrent objects, test case explosion becomes a major problem in testing concurrent systems. Synchronization and Deadlock being the two key features of concurrent systems make the systematic testing of concurrent systems a tedious task. In this paper we present a novel approach of generating test cases for concurrent systems with the help of UML Sequence Diagram. Our approach consists of transferring the Sequence Diagram into a Concurrent Composite Graph (CCG). The CCG is traversed by an effective graph traversing technique like BFS (Breath-
First-Technique) and DFS (Depth-First-search) using message sequence path criteria to generate the test cases for concurrent systems. The proposed approach is applied to concurrent systems for test case generation and found to be very effective in controlling the test case explosion problem. The generated test cases are useful to detect interaction, scenario, as well as operational faults in case of concurrent systems.

10. Andreas Heinecke, Tobias Brückmann, Tobias Griebe, Volker GruhnThe Unified Modeling Language (UML) is the standard to specify the structure and behaviour of software systems. The created models are a constitutive part of the software specification that serves as guideline for the implementa- tion and the test of software systems. In order to verify the functionality which is defined within the specification doc- uments, the domain experts need to perform an acceptance test. Hence, they have to generate test cases for the accep- tance test. Since domain experts usually have a low level of software engineering knowledge, the test case generation process is challenging and error-prone. In this paper we propose an approach to generate high-level acceptance test plans automatically from business processes. These processes are modeled as UML Activity Diagrams (ACD). Our method enables the application of an all-path coverage criterion to business processes for testing software systems.

IV. PROBLEM FORMULATION

In our Research we will work on future work to make an Automated tool system where we upload activity digarme xmi it will generate test case and also give their prioritisation. in our Research we will make test case prioritisation using hybrid algo which will be the combination of Ant colony and decision tree algorithm etc.

Decision Tree:-

Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. It is one of the predictive modelling approaches used in statistics ,data mining and machine learning. Tree models where the target variable can take a finite set of values are called classification trees. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining , a decision tree describes data but not decisions; rather the resulting classification tree can be an input for decision making . This page deals with decision trees in data mining.

Decision tree learning is a method commonly used in data mining. The goal is to create a model that predicts the value of a target variable based on several input variables. An example is shown below. Each interior node corresponds to one of the input variables; there are edges to children for each of the possible values of that input variable. Each leaf represents a value of the target variable given the values of the input variables represented by the path from the root to the leaf.

A decision tree is a simple representation for classifying examples. For this section, assume that all of the features have finite discrete domains, and there is a single target feature called the classification. Each element of the domain of the classification is called a class. A decision tree or a classification tree is a tree in which each internal (non-leaf) node is labeled with an input feature. The arcs coming from a node labeled with a feature are labeled with each of the possible values of the feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes.

A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node has all the same value of the target variable, or when splitting no longer adds value to the predictions. This process of top-down induction of decision trees (TDIDT) is an example of a greedy algorithm, and it is by far the most common strategy for learning decision trees from data.

In data mining, decision trees can be described also as the combination of mathematical and computational techniques to aid the description, categorisation and generalisation of a given set of data.

Data comes in records of the form:

\[
Y, x_1, x_2, x_3, \ldots, x_n\]

The dependent variable, \(Y\), is the target variable that we are trying to understand, classify or generalize. The vector \(x\) is composed of the input variables, \(x_1, x_2, x_3\) etc., that are used for that task.

**Types**

Decision trees used in data mining are of two main types:

- **Classification tree** analysis is when the predicted outcome is the class to which the data belongs.
- **Regression tree** analysis is when the predicted outcome can be considered a real number (e.g. the price of a house, or a patient’s length of stay in a hospital).

The term **Classification And Regression Tree (CART)** analysis is an umbrella term used to refer to both of the above procedures, first introduced by Breiman et al. Trees used for regression and trees used for classification have some similarities - but also some differences, such as the procedure used to determine where to split.

Some techniques, often called **ensemble** methods, construct more than one decision tree:

- **Bagging** decision trees, an early ensemble method, builds multiple decision trees by repeatedly resampling training data with replacement, and voting the trees for a consensus prediction.
A Random Forest classifier uses a number of decision trees, in order to improve the classification rate. 

**Boosted Trees** can be used for regression-type and classification-type problems.

**Rotation forest**: - in which every decision tree is trained by first applying principal components component analysis (PCA) on a random subset of the input features.

A special case of a decision tree is a Decision List which is a one-sided decision tree, so that every internal node has exactly 1 leaf node and exactly 1 internal node as a child (except for the bottommost node, whose only child is a single leaf node). While less expressive, decision lists are arguably easier to understand than general decision trees due to their added sparsity, permit non-greedy learning methods and monotonic constraints to be imposed.

**Decision tree learning** is the construction of a decision tree from class-labeled training tuples. A decision tree is a flow-chart-like structure, where each internal (non-leaf) node denotes a test on an attribute, each branch represents the outcome of a test, and each leaf (or terminal) node holds a class label. The topmost node in a tree is the root node.

There are many specific decision-tree algorithms. Notable ones include:

- ID3 (Iterative Dichotomiser 3)
- C4.5 (successor of ID3)
- CART (Classification And Regression Tree)
- MARS: extends decision trees to handle numerical data better.
- Conditional Inference trees. Statistics-based approach that uses non-parametric tests as splitting criteria, corrected for multiple testing to avoid overfitting. This approach results in unbiased predictor selection and does not require pruning.

D3 and CART were invented independently at around the same time (between 1970 and 1980), yet follow a similar approach for learning decision tree from training tuples.

**Metrics**

Algorithms for constructing decision trees usually work top-down, by choosing a variable at each step that best splits the set of items. Different algorithms use different metrics for measuring "best". These generally measure the homogeneity of the target variable within the subsets. Some examples are given below. These metrics are applied to each candidate subset, and the resulting values are combined (e.g., averaged) to provide a measure of the quality of the split.

**Gini impurity**

Not to be confused with gini coefficient.

Used by the CART (classification and regression tree) algorithm, Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. Gini impurity can be computed by summing the probability of each item being chosen times the probability of a mistake in categorizing that item. It reaches its minimum (zero) when all cases in the node fall into a single target category.

To compute Gini impurity for a set of items, suppose, and let be the fraction of items labeled with value in the set.

**Information gain**

Main article: Info gain in decision trees

Used by the ID3, C4.5 and C5.0 tree-generation algorithms. Info gain is based on the concept of entropy from information theory.

Entropy is defined as below

\[
\text{Information Gain} = \text{Entropy(parent)} - \text{Weighted Sum of Entropy(Children)}
\]

**Variance reduction**

Introduced in CART, variance reduction is often employed in cases where the target variable is continuous (regression tree), meaning that use of many other metrics would first require discretization before being applied. The variance reduction of a node \( N \) is defined as the total reduction of the variance of the target variable \( x \) due to the split at this node:
where \( I \), \( J \), and \( K \) are the set of presplit sample indices, set of sample indices for which the split test is true, and set of sample indices for which the split test is false, respectively. Each of the above summands are indeed variance estimates, though, written in a form without directly referring to the mean.

**Decision tree advantages**

This section does not cite any sources. Please help improve this section by adding citations to reliable source. Unsourced material may be challenged and removed. (July 2015)

Amongst other data mining methods, decision trees have various advantages:

- **Simple to understand and interpret.** People are able to understand decision tree models after a brief explanation.
- **Requires little data preparation.** Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed.
- **Able to handle both numerical and categorical data.** Other techniques are usually specialised in analysing datasets that have only one type of variable. (For example, relation rules can be used only with nominal variables while neural networks can be used only with numerical variables.)
- **Uses a white box model.** If a given situation is observable in a model the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model, the explanation for the results is typically difficult to understand, for example with an artificial neural network.
- **Possible to validate a model using statistical tests.** That makes it possible to account for the reliability of the model.
- **Robust.** Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.
- **Performs well with large datasets.** Large amounts of data can be analysed using standard computing resources in reasonable time.

**Limitations**

- The problem of learning an optimal decision tree is known to be NP-Complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristics such as the greedy algorithm where locally-optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally-optimal decision tree. To reduce the greedy effect of local-optimality some methods such as the dual information distance (DID) tree were proposed.
- Decision-tree learners can create over-complex trees that do not generalise well from the training data. Mechanisms such as pruning are necessary to avoid this problem (with the exception of some algorithms such as the Conditional Inference approach, that does not require pruning).
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems. In such cases, the decision tree becomes prohibitively large. Approaches to solve the problem involve either changing the representation of the problem domain (known as propositionalisation) or using learning algorithms based on more expressive representations (such as statistical relational learning).
- For data including categorical variables with different numbers of levels, information gain in decision trees is biased in favor of those attributes with more levels. However, the issue of biased predictor selection is avoided by the Conditional Inference approach.

**Extensions**

**Decision graphs**

In a decision tree, all paths from the root node to the leaf node proceed by way of conjunction, or AND. In a decision graph, it is possible to use disjunctions (ORs) to join two more paths together using Minimum message length (MML).

Decision graphs have been further extended to allow for previously unstated new attributes to be learnt dynamically and used at different places within the graph. The more general coding scheme results in better predictive accuracy and log-loss probabilistic scoring. In general, decision graphs infer models with fewer leaves than decision trees.

**Alternative search methods**

Evolutionary algorithms have been used to avoid local optimal decisions and search the decision tree space with little a priori bias.

It is also possible for a tree to be sampled using MCMC.

The tree can be searched for in a bottom-up fashion.
VI. CONCLUSION AND FUTURE WORK

In our research work we have developed an algorithm with the help of this we can generate the test cases and then optimize the test cases based on shortest path come on top. our tool help for the developer to do white box testing for their module. But there is an limitation we cannot find out the hidden test cases in our research for this further work has been required.

REFERENCES

[1] P. Samuel R. Mall A.K. Bothra “Automatic test case generation using unified modeling language (UML) state diagrams” Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur 721302, West Bengal, India E-mail: philips@cusat.ac.in


[3] Anneliese von Mayrhauser, Senior Member, ”Generating Test-Cases from an Object-Oriented Model with an Artificial-Intelligence Planning System ”IEEE, Robert France, Member, IEEE, Michael Scheetz, and Eric Dahlman


[5] Debasish Kundu1, Debasis Samanta1, Rajib Mall2 “Automatic code generation from unified modelling language sequence diagrams ”


[9] Philippe Chevalley, Pascale Thkvenod-Fosse “Generation Test Case from UML Activity Diagram Based on AC Grammar”


[13] Chi-Kuang Chang and Nai-Wei Lin “UTGen: a Black-Box Method-Level Unit-Test Generator for JUnit Test-Platform”