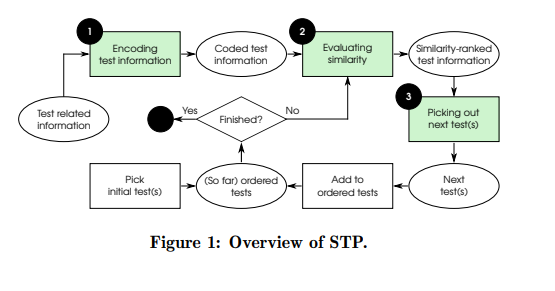
FAST Approaches to Scalable Similarity-based Test Case Prioritization

Objective : For real-world software the size of a test suite can often exceed the size of the system under test. Test suite sizes grow at fast pace, and existing techniques and tools for regression testing become inadequate.

Approach : FAST uses the coded test information T to create the minhash signatures M.

WB the code coverage information can be directly represented as sets(regardless of the coverage criterion), for BB the string representation of the test cases need to be preprocessed into k-shingles. Used a number of hash functions h = 10, which guarantees an expected error not greater than 0.32 in the estimation of the Jaccard similarity(and distance) between two signatures.

Even if the error in the estimation is high, the choice of the next test case is performed over a subset of tests that are all dissimilar from the so-far-prioritized ones.



Evaluation Metrics : APFD, assess the investigated TCP approaches in terms of preparation time and prioritization time.

Strength point : effectiveness, efficiency(total time and preparation time each) in bigger test more effective.

Provide greater benefits if it is adopted in an environment where the test case signatures can be reused as new test cases are added.

Weakness: it cost quite large preparation time. And more precise result, more time consume.

Every process used in similar projects and may have bias for randomly generated test information could reproduce unusual situation. it could used generally?

Scalable Approaches for Test Suite Reduction (ICSE2019)

Objective : reduction test suite techniques based on fast++ techniques(minhashing and locality-sensitive) applies the k-means algorithm and FAST-CS construct coreset,a clustering technique that scales up to massive dataset.

Enhance the scalability of both approaches by applying the random projection technique, reduces the space dimensionality while preserving the pairwise distances of the points.

Approaches : Using Euclidean distance. Transforms test cases into points in the Euclidean space via the vector-space model : The textual representation of each test case(test source code or command line) is mapped into an n-dimensional point where each dimension corresponds to a different term of the source code and n is equal to the total number of terms used in the whole test suite.  
Using sparse random projection, reduce the dimension.

FAST++ reduce test suites based on k-means++ as greedy reduction strategy. The test suite T, mapping each test case into a vector according to the vector-space model and then lowering its dimensionality via random projection.

FAST cs simplify the clustering problem is that of using coreset. Given a set of points S, a coreset is a small subset of S that well approximates the geometric feature of S.

Strength : Preparation time and Prepared data time is lower than above paper.

Weakness : de facto standard metrics,

Evaluation metrics : Fault detect loss, test suite reduction.

QTEP : Quality-Aware Test Case prioritization(ESEC/FSE 2017)

Objective : find code fault-prone for information to prioritize test cases.

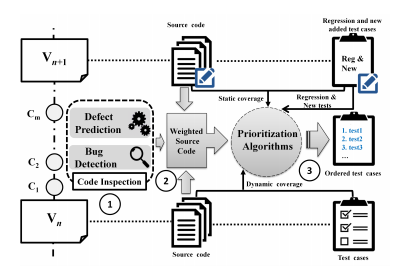
Approach : QTEP gives more weight to fault-prone source code so test cases that cover the fault-prone code have a higher priority to be executed.

With two code inspection approaches.

1. Static bug finders (perform FindBugs on its source code(without test code) to detect potential bugs.)
2. Defect prediction models(predict potential faults in the source code snapshot)

Leverage the detection results from the two code inspection approaches to weight the fault-prone source code units.

1. If class Method or class contains code unit is detected as buggy - > weight of this code unit will be calculated by accumulating the weights of the buggy method
2. If the class or the method that contains this code unit is identified as clean -> its weight will not be updated
3. Code units that are not covered by any buggy class or method will be assigned the default weight.,



Evaluation metrics : APFD

Strength : Using static bug finders and statistical defect prediction models to improve testing efficiency.

Can check more suspicious method.

Weakness : all bugs cannot be detected by FindBugs. So this research effectiveness may vary for using other techniques. And APFD can not reflect time and space costs or the severity of faults. Not general commercial projects or other languages.

Optimizing Test Prioritization via Test Distribution Analysis

Objective : To achieve the optimal prioritization effectiveness for any given project in practice. Learning based Predictive Test Prioritization.

Approach : PTP builds a predictive model via XGBoost by collecting three groups of features(distribution of test coverage, testing time, and coverage per time unit) on existing projects and labeling which prioritization technique performs optimal on the training data. (Cost-awareness,Cost-unware,and cost-only techniques)

In the training process, PTP collects the test distribution features and label information(e.g., which prioritization technique performs optimal) for each training project, and performs feature normalization and over-sampling to build the predictive model.

Then, given any new project, PTP can predict its optimal test prioritization technique based on its test distribution features.

For each project with a test suite, extract the three group of features

1. The number of program elements covered by each test,
2. The testing time of each test,
3. The number of program elements covered by each test per unit time.

Evaluation metrics APFDc

Strength : result from practical testing infrastructure for Baidu(industrial subjects) Differently from other works, PTP aims at generating the optimal prioritization results for each project through the selection of prioritization techniques.

Weakness : use the mutation faults. So There are threats from mutations.

Assessing Test Case prioritization on Real Faults and Mutants.

Objective : investigated (1) whether mutants are as difficult to detect as real faults (2) whether mutant detection correlates with real fault detection (3) whether mutants can be used to guide test case generation (4) whether tokens contained in patches for real-world faults can be expressed in terms of mutants

Our study is, in essence, aimed at investigating how test case ordering(and prioritization) can impact the representativeness of mutants in terms of fault detection rates.

RQ

1. How effective are TCP techniques when applied to detecting real faults?
2. Is the performance of TCP techniques on mutants representative of performance on real faults?
3. How do the properties of real faults and mutants affect the performance of TCP techniques?

So Objective is to determine how well mutation-based measures of TCP effectiveness reflect the performance of these techniques on real faults.

Approaches : create one mutated program instance by seeding a randomly selected mutant into the latest corresponding program version, and then repeat this process 100 times.

Result

1) Real faults, all techniques tend to perform better when APFD compared to APFDc(for execution cost)

2) APFD and APFDc metrics calculated using the full mutant set generally tend to overestimate absolute performance compared to real faults by 20% on average. Mutation -based TCP performance APFDc is more correlate to performance in terms of APFDc on real faults.

3) This investigation suggests that TCP performance correlations between real faults and mutants is low when

seeded mutants do not properly reflect faults occurring in a given domain or a program.

Strength : explain correlation between mutant and real fault in terms of empirical study.

Weakness : mutation tool`s variation. Different affects from mutants seeded in the same code from the real fault. This result are representative of a certain set of mutants.

Evaluation metrics : APFD and APFDc(but more consider for APFDc and same severity)