State of Art in the field of Search-based Mutation Testing

Nishtha Jatana1    Shweta Rani2   Bharti Suri3

1 Maharaja Surajmal Institute of Technology, JanakPuri, New Delhi, India
email : nishtha.jatana@gmail.com
2 USICT, GGS Indraprastha University, Dwarka, New Delhi, India
email : shweta2610@gmail.com
3 USICT, GGS Indraprastha University, Dwarka, New Delhi, India
email : bhartisuri@gmail.com

Abstract-- Search-based Mutation Testing makes use of Meta-heuristic optimization techniques like Genetic Algorithm, Hill Climbing and other such Evolutionary Approaches for the process of mutation testing. This paper presents the opportunities and challenges faced in applying Search-based techniques to mutation testing. Enormous amount of research has been done in the recent years in this area. The major challenge is the storage and computational cost for a large set of mutants. This paper also lists the latest research, available tools and updates to SBMT.

Index Term- Meta-heuristics, Mutation testing, Search-based, Testing Tools

I. INTRODUCTION

Software testing requires validating and verifying the software product against the specified requirements and the major intent for applying software testing is to find the faults in the product. In order to extensively test the product, huge amount of test data is required. Search Based Software Testing (SBST) applies meta-heuristic or optimization techniques, used for test case selection, prioritization, minimization and predominantly for test data generation. History of SBST can be traced in a recent work by Harman et al. [1]. SBST works by formulating the testing problem as search problem by representing candidate solutions and then applying optimization algorithms to evaluate these candidate solutions to find near optimal solutions guided by a specified fitness function.

Mutation Testing[2][3] is a fault based testing technique which works by first injecting faults into a program and then a set of test cases (test suite) are applied to the faulty programs (Mutants). These mutants are created by simple syntactic changes to the original program. The results corresponding to the test cases applied to the mutants are evaluated against the result of the original program. If any of the mutant yields a different result; the mutant is set to be killed.

For some mutants, it is not possible to find a test case that can distinguish the behaviour of mutants from that of original program. Such mutants are called equivalent mutants. Adequacy of the test suite is measured by the percentage of the killed mutants. If a program has \( \alpha \) mutants, \( \beta \) of which are equivalent, and a set of test cases \( \mu \) kills \( \pi \) mutants then the mutation score (MS) is given by:

\[
MS = \frac{\pi}{\alpha-\beta}
\]  

(1)

Mutation testing can be applied to different levels of testing like unit testing [4][5][6], integration testing[7], specification testing and also at design level to verify models of a program. Mutation testing has been applied to various programming languages like FORTRAN, ADA, C [4] [8][9] [10], C#, Java[11][12] [13][14][15][16][17][18] [19], Python[20], PHP [21], JavaScript[13][22] and even to SQL[23], AspectJ programs [24][18][19] and XML[25]. Mutation testing has been increasingly widened in the past few years. A few literature surveys have been done on mutation testing. Other related surveys are [27][28][29][30].

Search-based Mutation Testing (SBMT) combines meta-heuristic techniques and Mutation Analysis to solve optimization problems. Meta-heuristic techniques are applied for optimized generation of test cases, test-case prioritization, selection and minimization and/or mutant optimization; thus reducing the cost of Mutation Testing.

In SBMT, a meta-heuristic optimization algorithm and its corresponding fitness function is chosen on the basis of characteristics of the problem. Initial population of test input is then taken randomly and evaluated for fitness. Fitness is estimated based on the number of mutants killed by test data out of total mutants generated. If fitness is greater than a threshold value; then the population is accepted, else it is subjected to further improvement using the optimization algorithm. The iterative process continues, thus yielding optimised test suite.

In order to find the papers relevant for the work conducted, we searched the online repositories of the key
technical publishers, including IEEE explore, Springer Online Library, Elsevier Online Library, ACM Portal and Wiley having both “Mutation testing” and “Search based” keywords in their title or abstract. We have also searched for recent advancements in mutation testing and the publicly available tools for mutation testing that came into existence after the year 2009. Author search for researchers, working in the area was also done to find the relevant papers.

The contribution of the present paper can be summarized as:

i. Lists the recent advancements in the area of SBMT.
ii. Lists various challenges in SBMT.
iii. Lists various tools for mutation testing released in year between 2010 -2014.

The rest of the paper is organised as follows: Section II presents the research work carried on SBMT. Section III lists the challenges in SBMT. The major and the latest developments and the automated tools in the field are studied and listed in section IV. Section V concludes.

II. RELATED WORK

Usage of optimization algorithms to kill mutants was first suggested in [31] by Bottaci. A new fitness function was described based on three conditions: reachability, necessity and sufficiency for killing the mutants. A model [32] was proposed to reveal faults and to kill the mutant using Genetic Algorithms. Genetic Programming was used to derive alternatives for testing program under test and introduced two GP-based procedures for selection and evaluation of test data [33]. They presented a tool based on Genetic Programming named GPTesT. Evolutionary Genetic Algorithm was introduced in [34] for mutation testing by selection of test cases and mutated programs using a new representation and implementing new genetic operators to yield better mutation score and reduction in computational cost. A multi-objective ‘Pareto optimal technique’ with ‘Monte Carlo sampling’, Genetic Algorithm and Genetic Programming [35] was implemented to search for higher order mutants. Harman et al. [36] introduced the concept of strongly subsuming higher order mutants by using three algorithms: Greedy Approach, Genetic Algorithm and Hill Climbing. A technique for automatic generation of test inputs for killing higher-order mutants was introduced in [35] based on Genetic Algorithm. Adamopoulos et al. [37] worked on the equivalent mutant problem and introduced a new fitness function based on co-evolution to overcome the equivalent mutant problem and also evolved the mutants and test data; thus reducing the cost of SBMT. [38] presented Genetic Algorithm based test suite aiming to produce a set of mutations having high impact, and were wide-spread throughout the code and were not easily detectable by the test suites. [39] presented an approach(μ test) for unit-test and oracle generation using Genetic Algorithm. Other work related to SBMT is discussed in [34][40].

Baudry et al. in their publications[41] [42] [43][44] applied Bacteriological Algorithm to Mutation Testing. Akkari et al. [45] combined Genetic Algorithm and Bacteriological Algorithm for optimizing the test data in SBMT.

May et al. in their publications [46] [47] [48] applied AIS to Mutation Testing aiming to act as a vaccination to software development process in order to save the software from errors. A comparative study [47] demonstrates an Elitist Genetic Algorithm to an Immune Inspired Algorithm for software test data evolution.

Ayari et al. [49] proposed Ant Colony Optimization for automatic generation of test input data as optimization technique for reduction in the cost of mutation testing.

Harman et al. have worked on higher order mutation testing with search based techniques for test case generation and mutant optimization. Relevant publications include [35] [36] [50] [51][52]. [53] also used higher order mutants for SBMT.

Papadakis & Malevris [54] developed framework using alternating variable method and mutant schemata approach for SBMT.

III. CHALLENGES IN SEARCH-BASED MUTATION TESTING

This section lists the limitation of mutation testing in general but these challenges are also applicable to SBMT. Limitations associated with GA and other evolutionary approaches are also enumerated here which are also appropriate to SBMT challenges.

Nguyen & Madeysky [55] in their study pointed out the following challenges in mutation testing: (i) Huge amount of mutants generated corresponding to program under test, (ii) Very high execution cost in terms of time and effort, (iii) The problems seeded in mutation analysis are not as complex as real faults, (iv) Detecting equivalent mutants requires manual attention.

Reales et. al. [56] stated issues in mutation analysis as: (i) high cost in terms of storage as all mutants need to be stored, (ii) high computational cost for generation and running testsuites, (iii) great amount of human effort.

As per our research, the following challenges were encountered: (i) It is challenging to decide the criteria for selection of appropriate Mutation Testing tool, (ii) The mutation testing tool (Jester, [57]) that mutates the test data along with the program is quite slow to work with, (iii) Another freeware tool (Jumble,[5]) gives the mutation score and the line numbers for mutants which are not killed but does not provide the mutants for further analysis, (iv) Equivalent Mutants which are not easily detectable, affect the mutation score by acting as false positives, (v) It is difficult to decide which test programs to choose as benchmark for evaluation of research on Mutation Analysis. (iv) MuClipse, an eclipse based plugin for mutant generation and execution, provides too many mutants with mutant viewer interface that requires a lot of storage.

Limitations while applying GA in software testing can be found in [58]. The general problem is to select the initial population and its size. GA can not be applied on all datatypes too.
IV. OPPORTUNITIES IN SEARCH-BASED MUTATION TESTING

In this section the latest trends and opportunities in SBMT have been listed as per our research.

Jia and Harman[26] listed extensively the research work carried out till 2009 in Mutation Testing. The further developments in Mutation Testing are stated in Table I.

As per our findings, major advancements in mutation testing since the year 2010 are: i) development of new delete mutation operators for C and Java, ii) mutation testing tools for JavaScript, logic formulas, web browser using grammar and event processing queries. (Refer Table I).

TABLE I: Major and Recent Developments in Mutation Testing (2010 onwards)

<table>
<thead>
<tr>
<th>Year</th>
<th>Description of Advancement to Mutation Analysis</th>
<th>Aim/Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>Statement deletion operator was introduced [8]</td>
<td>Reduction in execution cost</td>
</tr>
<tr>
<td>2010</td>
<td>Mutation testing on Web Application [11]</td>
<td>Solution to problems related Integration testing of Web Applications by Mutation Analysis</td>
</tr>
<tr>
<td>2010</td>
<td>Semantic Mutation Testing [4]</td>
<td>Changing semantics of the language seeds different class of faults to be captured by mutants</td>
</tr>
<tr>
<td>2010</td>
<td>Binary Mutation Testing[59]</td>
<td>Mutants are generated through binary mutation at run-time using dynamic translation</td>
</tr>
<tr>
<td>2013</td>
<td>JavaScript Mutation Testing [13]</td>
<td>To guide the mutation generation process towards parts of the code that are prone to errors or possibly influence the JavaScript program’s output</td>
</tr>
<tr>
<td>2013</td>
<td>GramFuzz: testing of web browsers based on grammar and mutation analysis [22].</td>
<td>Fuzzy testing of web browsers contents(HTML, CSS, JavaScript files) by building grammar trees and mutation test cases with structural strategies</td>
</tr>
<tr>
<td>2014</td>
<td>Mutation on Logic Formulas [60]</td>
<td>Evaluation of quality of a test suite with respect to logic constraints using Mutation Analysis</td>
</tr>
</tbody>
</table>

A Survey on Mutation Testing [26] lists tools for Mutation Testing developed until the year 2009. Table II lists the various tools available for Automated Mutation Testing 2010 onwards. As per our research, recently developed tools for mutation testing are mostly built for Java language. These Tools are easy to use, save time and effort for mutation testing. A few more surveys exist on mutation testing tools [61][62].

TABLE II: Tools launched for Mutation Testing (2010 onwards)

<table>
<thead>
<tr>
<th>Year</th>
<th>Name</th>
<th>Application</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>MuTMuT [14]</td>
<td>Java</td>
<td>Mutation testing of multi-threaded code</td>
</tr>
<tr>
<td>2011</td>
<td>MAJOR [16]</td>
<td>Java</td>
<td>Tool for Mutation Analysis in a Java Compiler</td>
</tr>
<tr>
<td>2011</td>
<td>EvoSuite [63]</td>
<td>Java</td>
<td>Search-based, Assertion and Mutation based test suite generation tool</td>
</tr>
<tr>
<td>2012</td>
<td>SMT-C [10]</td>
<td>C</td>
<td>Tool for Semantic Mutation Testing for C-Programs (SMT-C)</td>
</tr>
<tr>
<td>2014</td>
<td>MuCheck [64]</td>
<td>Haskell</td>
<td>Mutation testing tool for Haskell programs</td>
</tr>
<tr>
<td>2014</td>
<td>MutaTesting [21]</td>
<td>PHP</td>
<td>Mutation testing tool for PHP programs</td>
</tr>
<tr>
<td>2014</td>
<td>MutaLog [60]</td>
<td>Logic Formulas</td>
<td>Tool for Mutating Logic Formulas developed in Java</td>
</tr>
</tbody>
</table>

Figure 1 and figure 2 pictorially represent the trends of the recent developments in the field of SBMT as listed in the tables above. The line graph in Figure 1 depicts the publication trends of research in the field during the past years. The chart in Figure 2 signifies that mutation testing tools are being developed mostly for Java language in the recent years as Java is still one of the most preferred language by the software developers as stated by a recent survey[65].
Further research can be carried out in the field by using a variety of search-based techniques, including immune-inspired algorithms, ant colony optimization, and elitist genetic algorithms with mutation testing for test data generation. Other search-based techniques, such as Bacteriological Algorithm, are also being used.

### References


