UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Heidi Urmet

A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation

Bachelor's Thesis (9 EAP)

Supervisor: Dietmar Alfred Paul Kurt Pfahl

Tartu 2017

Süstemaatiline ülevaade ja empiiriline uurimus otsingupõhiste algoritmide kasutusest testide loomise

Lühikokkuvõte:

Otsingupõhine tarkvara testimine kasutab metaheuristilisi algoritme, et automatiseerida testide genereerimist. Selle töö eesmärgiks on osaliselt taasluua 2010. aastal kirjutatud Ali et al. artikkel, et uurida, kuidas on aastatel 2008-2015 kasutatud metaheuristilisi algoritme testide loomiseks. See töö analüüsib, kuidas on antud artiklid koostatud ning kuidas neis on algoritmide maksumust ja efektiivsust hinnatud. Kogutud tulemusi võrreldakse Ali et al. tulemustega.

Võtmesõnad:

Tarkvara testimine, otsingupõhine tarkvara testimine, süstemaatiline ülevaade
CERCS: P175

A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation

Abstract:

Search based software testing uses metaheuristic algorithms to automate the generation of test cases. This thesis partially replicates a literature study published in 2010 by Ali et al. to determine how studies published in 2008-2015 use metaheuristic algorithms to automate the generation of test cases. The thesis analyses how these studies were conducted and how the cost-effectiveness is assessed in these papers. The trends detected in the new publications are compared to those presented in Ali et al.

Keywords:

Software testing, search based software testing, systematic review
CERCS: P175

Table of Contents

1. Introduction

Software testing is an integral part of software engineering. Testing helps to determine the strengths and vulnerabilities of a software program. According to Anand et al. [1] testing can take more than 50% of the total cost during software development. Thus, it is beneficial to automate parts of this process to reduce the cost.

The software testing life cycle has many phases: requirements/design review; test planning; test design; test environment setup; test execution; test reporting [2]. Of all these phases, test design is the costliest. Manually coming up with all the different inputs can take a lot of time and there is always the possibility of human error e.g. forgetting one set of inputs. Thus, in order to speed up this phase and make it more efficient, automated test data generation comes into play. Automatically generating test data helps create the test cases that are in minimum needed to achieve the test goal.

Search based software testing (SBST) is a part of search based software engineering. It aims at automating the process of test data generation with the use of metaheuristic search (MHS) algorithms. Search based algorithms used in SBST use a fitness function to guide the search of the test inputs [1].

Focussing on the years 1996 – 2007, Ali et al. [3] give an overview of the state of how SBST has been applied and also about its cost-effectiveness. They also provide a framework on how a proper survey study in this field should be conducted. Since their paper was published in 2010 and it only analysed papers published until 2007, a similar study could be conducted covering the subsequent time period (2008 – 2015) to analyse whether new trends have emerged within search based software testing. This thesis aims at replicating some elements of the study conducted by Ali et al. and to compare the findings.

The paper by Ali et al. [3] has three research questions. Two of them had three respectively five sub-questions. Thus, if only counting the questions on the lowest level of the hierarchy, nine research questions were investigated. Of these nine research questions investigated in [3], the following five research questions are addressed in this thesis:

RQ1. For which metaheuristic search algorithms, test levels, and fault types is there credible evidence of cost-effectiveness in the literature?

RQ2. How convincing is the evidence of cost and effectiveness of search-based software testing techniques, based on empirical studies that report credible results?

RQ3. How well is the random variation, inherent in search-based software testing, accounted for in the design of empirical studies?

RQ4. What are the most common alternatives to which SBST techniques are compared?

RQ5. What are the measures used for assessing cost and effectiveness of search-based software testing?

The thesis is structured as follows. Chapter 2 gives a brief introduction into software testing and SBST. Chapter 3 details the research method used in this thesis. Chapter 4 reports the results of the research. Chapter 5 analyses the results and compares them to those of Ali et al. [3]. Chapter 5 also discusses threats to validity. Chapter 6 summarizes the study and presents the conclusions.

2. Background

This chapter gives a brief introduction to software testing, SBST and a framework for conducting an empirical study in SBST, used by Ali et al. [3].

2.1. Software Testing

Software testing is the process of running a programme with the intent to find errors and bugs so that it could be fixed and the best version of the programme could be released for use. It is a part of the Software Development Life Cycle.

The Software Testing Life Cycle (STLC) usually contains the steps requirements/design review, test planning, test designing, test environment setup, test execution and test reporting [2].

Of all these steps, the test design step is the costliest. During this step, the tests that are going to be used to assess the programme are created based on the requirements or code. Test design takes the most effort from the tester during this life cycle. There are a lot of possibilities for simple errors to occur (e.g. assembling a set of inputs for testing, but a wrong output). Thus, it is beneficial to come up with ways to automate this step and make it as effective as possible.

The tests that are created during test design are grouped together into test scripts, test suites or test cases. A test script is a set on instructions that are performed to determine whether the system under test performs these actions. A test suite is a collection of test cases. A test case contains the input data and the expected output data and the conditions under which tests should be executed. Test data is data that is meant to be used in test cases.

An oracle is a mechanism for determining whether a program passes or fails a test [4]. A complete oracle should provide the predicted and expected result of the test, compare the expected result and the actual result, and determine if the expected result and actual result are similar enough for the test to pass. A test verdict is what decides if the test is marked as pass/fail after the test has been executed.

In this thesis, test cases are the point of focus. As in the paper by Ali et al. [3], the expected output is not of interest but generating the input data is.

2.2. Search Based Software Testing

Search based software testing (SBST) is a part of search based software engineering (SBSE). In SBST, metaheuristic search (MHS) algorithms are used to generate test data. The goal is to make finding test data efficient while minimizing the cost of doing so. To guide the selection of this, fitness functions are used. Fitness functions help evaluate if the test data is good or not and guide the search to areas that might produce even better results from the fitness function.

There are many metaheuristic algorithms, for example: genetic algorithm (GA), simulated annealing (SA), particle swarm optimization (PSO), ant colony optimization (ACO), tabu search (TS), scatter search (SS), and hill climbing (HC) among others. All the mentioned algorithms are briefly introduced.

Genetic algorithm. Genetic algorithm is inspired by the process of natural selection. First a population is created randomly and then each individual's fit is assessed by the fitness function. The best individuals are selected and they are used to create a new population. This is an iterative process so this is repeated until a certain number or until the user deems necessary [5].

Genetic algorithm was first introduced to software testing in 1975 by Holland. Since then it has been used in software testing a lot. For example it has been used in test case/test data automation (e.g. [6-9]), selection (e.g. [6]), and optimization [7].

Simulated annealing. Simulated annealing takes its idea from heating a material and them cooling it to get rid of defects [8]. Simulated annealing is an iterative process. In each iteration a new point is chosen from its neighbours and then the algorithm either stays in the state it was or moves to the neighbouring state. Simulated annealing can be used to generate test data [9] and test cases [10], or for test selection and optimization [11].

Particle swarm optimization. PSO is inspired by the swarming behaviour like bee schooling or fish schooling. In each iteration the particles (candidate solutions) move around influenced by the local and global optima. When a better solution is found, these will become the new local or global optima. This makes the swarm move toward a better solution. PSO can be used to generate test cases [12], test case selection [13].

Ant colony optimization. Like with the previous metaheuristics, this also has been inspired by nature. In particular how ants find the shortest path to food by taking advantage of the pheromones deposited on the ground by others. Paths that did not lead to

any food and are so less used and the pheromones on the ground start to evaporate. As an optimization technique the algorithm is an iterative process, during each iteration an ant moves to the next state. When all ants have completed their trail, the trails are marked with pheromone in each state and it evaporates after each iteration. This way the most used path will be the result. ACO has been used to generate test cases [14], tackle optimization problems.

Tabu search. Tabu search moves iteratively from one solution to the neighbouring solutions hoping they are better. TS uses memory to memorize previous search results, this prevents the algorithm from going back to evaluate a solution more than once.

Scatter search. Scatter search was first introduced in 1977 by Glover [15]. SS works with a set of solutions (the reference set), then combines the solutions to create better ones. The combined sets that are created and evaluated whether they could replace some of the solutions in the reference set. SS has been used for scheduling problems, in neural networks and for test case generation [16].

Hill climbing. Hill climbing is a local search optimization algorithm. It is an iterative algorithm that starts off with a random individual, then chooses to move on to neighbours if they are deemed better by the algorithm and then repeats this process at the new spot. HC has the threat to get stuck in local maxima, so to avoid this, the algorithm can start the process again at a random place. HC can be used to generate test data [17] and for test case prioritization [18].

2.3. Framework

In their paper, Ali et al. [3] presented a framework for carrying out empirical studies on SBST. To be an empirical study was one of the inclusion criteria in the literature survey conducted by Ali et al. Therefore, this framework was used as a guideline in this thesis as well. The framework consists of four main elements. Firstly, it checks how clearly the test problem has been stated. Secondly, it checks how clearly, the MHS algorithms have been described. Thirdly, and most importantly, it checks the adequacy of the research design used. This part details what exactly should be described in the paper so that its validity can be assessed. Lastly, the results must be clear and reproducible.

3. Method

To find relevant literature, first the snowballing method was tried. However, for reasons described in the following sub-section, this approach was later discarded. In its place, a standard systematic literature review was conducted (cf. Section 3.2 and following).

3.1. Snowballing

Snowballing is a systematic literature review process that starts with a start set of papers. Then the papers that cite these are gathered in what is called forward snowballing. Then the papers that the start set references are also gathered in what is called backwards snowballing. According to the exclusion or inclusion criteria, the desired papers are chosen. Then the process is repeated with the new set of papers until no new papers are found.

At the beginning of this thesis project, the idea was to use the papers analysed in [3] as the starting set for snowballing following the guidelines provided by Wohlin [19]. However, it became apparent that not all the papers Ali et al. [3] analysed were mentioned explicitly by every RQ and it would take too much time and resources to do snowballing on the papers that were. Therefore, it was decided to do what was done in the original paper, i.e. to search various literature repositories.

3.2. Repository Selection and Search String Definition

The paper by Ali et al. [3] used the repositories *IEEE Xplore, The ACM Digital Library, Science Direct (including Elsevier Science), Wiley Interscience, Springer*, and *MIT Press*. To get similar results the same repositories were used in this thesis, with the exception of MIT Press due to unavailability via the library of the University of Tartu.

The search string used in the Ali paper [3] was as follows:

(((“software” AND “test”) OR "test case generation") AND ("evolutionary algorithm" OR "hill climbing" OR "metaheuristic" OR "meta-heuristic" OR "genetic algorithm" OR "optimization algorithm" OR "search based" OR "search-based" OR "simulated annealing" OR "ant colony”)) <in abstract, keywords, and title> OR "evolutionary testing” <in abstract, keywords, title, and whole content>

First it was tried to recreate the results of the original paper. Although 68 papers were analysed and used to answer the research questions, only 19 of these were mentioned in

name and author in [3]. Thus, it was checked whether these 19 papers could be found. 18 were found, one was not. The missing one could be from the fact that access to all repositories was not available or within certain repositories Ali et al. [3] added synonyms to the search string but these were not mentioned in the paper (e.g. in IEEE Xplore).
As this thesis has a smaller scope than the original study and to minimize the number of search results, an AND operator was added to the search string that contained the phrases "cost", "effective", "cost-effectiveness" all connected with an OR operator. The final string used in this paper is:

((((“software” AND “test”) OR "test case generation") AND ("evolutionary algorithm" OR "hill climbing" OR "metaheuristic" OR "meta-heuristic" OR "genetic algorithm" OR "optimization algorithm" OR "search based" OR "search-based" OR "simulated annealing" OR "ant colony")) <in abstract, keywords, and title> OR "evolutionary testing” <in abstract, keywords, title, and whole content>) AND (“cost” OR “effective” OR “cost-effective”)

Using the search string in different repositories requires adjustments. For example, since the "evolutionary testing" part of the search string also looks at the entire content of the paper and is connected to the rest of the string with an OR operator, it was sometimes easier to split the string in half at the OR operator. Later the results could be mixed together and duplicates could be removed. The split was done in ACM Digital Library and IEEE Xplore.
When using the search string in the Springer repository, it became apparent that there were too many results that were irrelevant to this research. To filter the search more, the search string was modified. The modified search string was as follows (the modification has been underlined):

((("software" AND "test case generation") AND ("evolutionary algorithm" OR "hill climbing" OR "metaheuristic" OR "meta-heuristic" OR "genetic algorithm" OR "optimization algorithm" OR "search based" OR "search-based" OR "simulated annealing" OR "ant colony")) OR "evolutionary testing") AND ("cost" OR "effective" OR "cost-effective")

Similarly, with the Wiley repository, there were too many out of context results (a lot of medical articles). In order to filter between these a NOT operator was added in front of the

terms “medical”, “medicine”, “biology” (all of them connected with an OR operator). This decreased the results from about 180 to 48, a considerable amount.

3.3. Paper Selection

All of the results from the repositories were run through a small self-made program that removed all duplicates. After duplicate removal, 597 papers were left for further analysis. Firstly, the title and abstract of these papers were read. As in [3] papers that

- had abstracts or titles that did not discuss test case generation were excluded
- had abstracts or titles that did not discuss the application of any MHS algorithm to automate test case generation were excluded

After applying these exclusion criteria 108 papers were left for further analysis.
The 108 papers were read in full and based on the second set of exclusion criteria either excluded or included. The second set of criteria used in [3] was as follows:

- The papers had to automate test case generation were excluded.
- The papers had to report an empirical study were excluded. (see 3.2.)
- Posters, extended abstracts, technical reports, PhD dissertations, and papers with less than three pages were excluded.

Table 1. Number of papers after applying exclusion criteria

Repository	Search string results (after removing duplicates)	Number of papers left after applying the 1st set? Of exclusion criteria	Number of papers left after applying the 2nd set on exclusion criteria
IEEE Explore	361	73	23
ACM	134	24	8
Wiley	43	4	1
ScienceDirect	55	8	6
Springer	4	0	0
Number of papers	597	109	38

The two rounds of exclusion criteria left a total on 38 papers. Among the 38 papers, there was one paper that was published at a conference and later in a journal. Since the context is the same, the journal was chosen to be analysed in this thesis, thus reducing the total

number of papers analysed to 37. The numbers of papers found in each repository are shown in Table 1.

3.4. Data Extraction

To gather all the relevant information from each paper an Excel spreadsheet for each set of inclusion/exclusion criteria was created. For the first set of criteria only the title and author(s) of the paper were gathered. The papers that were chosen were marked and transferred to another spreadsheet.

For the second set of criteria a more extensive spreadsheet was created. This spreadsheet[1] already contained the selected papers from the previous step, so the author(s) and title were already present. The same type of data was collected in this thesis as in the paper by Ali et al. [3]. This data includes the following:

- Test level
- Fault type
- MHS algorithm
- Test purpose
- Comparison baseline
- Cost and effectiveness results
- Cost measures
- Effectiveness measures
- Random variation (accounted for or not)

[1] The spreadsheet is accessible at: https://www.dropbox.com/s/pynrbn1r3w4h3ar/heidi.xlsx?dl=0

4. Results

This chapter outlines the results found for each research question. After each research question there is also a discussion subsection in which the results of this thesis are compared to the results found by Ali et al. [3]. 37 papers are used to answer the research questions. Table 2 summarises which paper was used to answer which research question. The check indicates that a paper was used to answer the corresponding research question. All the papers answered RQ3-5, 15 papers were used to answer RQ1 and 6 to answer RQ2.

Table 2. Table indicating which papers answers which research questions

ID	Title	Authors	RQ1	RQ2	RQ3	RQ4	RQ5
1	A First Approach to Test Case Generation for BPEL Compositions of Web Services Using Scatter Search	Blanco et al. [20]	✓		✓	✓	✓
2	A tabu search algorithm for structural software	Díaz et al. [21]	✓		✓	✓	✓
3	Adapting ant colony optimization to generate test data for software structural testing	Mao et al. [14]			✓	✓	✓
4	An approach to generate software test data for a specific path automatically with genetic algorithm	Cao et al. [22]	✓		✓	✓	✓
5	An Improved Memetic Algorithm with Method Dependence Relations (MAMDR)	Aburas and Groce [23]	✓		✓	✓	✓
6	Application of Genetic Algorithm and Tabu Search in Software Testing	Rathore et al. [24]			✓	✓	✓
7	Automated test data generation using a scatter search approach	Blanco et al. [16]	✓	✓	✓	✓	✓
8	Automatic Generating All-Path Test Data of a Program Based on PSO	Li and Zhang [12]	✓		✓	✓	✓
9	Automatic generation of software test cases based on improved genetic algorithm	Dong and Peng [25]			✓	✓	✓

10	Automatic generation of software test data based on hybrid particle swarm genetic algorithm	Ding et al. [26]			✓	✓	✓
11	Automatic generation of test data for path testing by adaptive genetic simulated annealing algorithm	Zhang and Wang [27]			✓	✓	✓
12	Automatic Path-Oriented Test Data Generation Using a Multi-population Genetic Algorithm	Chen and Zhong [28]			✓	✓	✓
13	Automatic program instrumentation in generation of test data using genetic algorithm for multiple paths coverage	Maragathavalli et al. [29]			✓	✓	✓
14	Automatic test case generation for unit software testing using genetic algorithm and mutation analysis	Khan and Amjad [30]			✓	✓	✓
15	Automatic Test Data Generation Based on SAMPSO Algorithm	Wei and Jiang [31]			✓	✓	✓
16	Automatic Test Data Generation for Software Path Testing Using Evolutionary Algorithms	Latiu et al. [32]			✓	✓	✓
17	Combining Genetic Algorithms and Constraint Programming to Support Stress Testing of Task Deadlines	Di Alesio et al. [33]	✓	✓	✓	✓	✓
18	Comparing algorithms for search-based test data generation of Matlab® Simulink® models	Ghani et al. [35]			✓	✓	✓
19	Comparison of Two Fitness Functions for GA-Based Path-Oriented Test Data Generation	Chen et al. [36]			✓	✓	✓
20	Critical Components Testing Using Hybrid Genetic Algorithm	Jeya Mala et al. [37]			✓	✓	✓
21	Diversity oriented test data generation using metaheuristic search techniques	Bueno et al. [38]	✓	✓	✓	✓	✓
22	Enhanced Genetic Algorithm For MC/DC Test Data Generation	El-Serafy et al. [39]			✓	✓	✓
23	Evolutionary Algorithms for Object-Oriented Test Data Generation	Suresh et al. [40]			✓	✓	✓

24	Evolutionary Testing of Object-Oriented Software	Silva and van Someren [41]			✓	✓	✓
25	GA-based multiple paths test data generator	Ahmed and Hermadi [42]	✓		✓	✓	✓
26	Generating combinatorial test cases using Simplified Swarm Optimization (SSO) algorithm for automated GUI functional testing	Ahmed et al. [43]	✓		✓	✓	✓
27	Generating Test Data for Structural Testing Based on Ant Colony Optimization	Mao et al. [44]			✓	✓	✓
28	Hybridizing Evolutionary Testing with Artificial Immune Systems and Local Search	Liaskos and Roper [45]			✓	✓	✓
29	Multi-Objective Test Generation for Software Product Lines	Henard et al. [46]	✓	✓	✓	✓	✓
30	Orthogonal Exploration of the Search Space in Evolutionary Test Case Generation	Kifetew et al. [47]			✓	✓	✓
31	PWiseGen: Generating test cases for pairwise testing using genetic algorithms	Flores and Cheon [48]			✓	✓	✓
32	Reformulating Branch Coverage as a Many-Objective Optimization Problem	Panichella et al. [50]			✓	✓	✓
33	Search Based Testing of Embedded Systems Implemented in IEC 61131-3: An Industrial Case Study	Doganay et al. [51]	✓	✓	✓	✓	✓
34	Search-based testing using constraint-based mutation	Malburg and Fraser [34]	✓	✓	✓	✓	✓
35	Test Data Generation Approach for Basis Path Coverage	Jiang et al. [52]			✓	✓	✓
36	Test Data Generation for Multiple Paths Based on Local Evolution	Xiangjuan et al. [49]	✓		✓	✓	✓
37	Test Data Generation From Hibernate Constraints	Marin and Doungsa-ard [53]	✓		✓	✓	✓

4.1. RQ1 – For Which Metaheuristic Search Algorithms, Test Levels, and Fault Types Is There Credible Evidence for the Study of Cost-Effectiveness?

The papers that are qualified to answer this RQ must comply to the following criteria set by Ali et al. [3]:

- Must account for random variation
- Comparison baseline must be a local SBST technique or a simpler non-SBST technique

After going through all 37 papers and applying the criteria mentioned above 15 papers were left. The details of these papers can be seen in Table 3. Among these 15 papers using GA was the most popular with 8 uses (although memetic algorithm is an extension of GA). SS was used twice, TS, HC, memetic algorithm (MA), PSO, SA, simulated repulsion (SR) and simplified swarm optimization (SSO) were all used once. The testing level for all papers but one was unit, the exception being system. None of the papers focused on finding fault types.

Table 3. List of papers with the MHS algorithm used, test level and fault type used to answer the first research question.

ID	Authors	MHS algorithm used	Test level	Fault type
1	Blanco et al. [20]	SS	unit	-
2	Díaz et al. [21]	TS	unit	-
4	Cao et al. [22]	GA	unit	-
5	Aburas and Groce [23]	MA	unit	-
7	Blanco et al. [16]	SS	unit	-
8	Li and Zhang [12]	PSO	unit	-
17	Di Alesio et al. [33]	GA	system	-
21	Bueno et al. [38]	SA, GA, SR	unit	-
25	Ahmed and Hermadi [42]	GA	unit	-
26	Ahmed et al. [43]	SSO	unit	-
29	Henard et al. [46]	GA	unit	-
33	Doganay et al. [51]	HC	unit	-
34	Malburg and Fraser [34]	GA	unit	-

36	Xiangjuan et al. [49]	GA	unit	-
37	Marin and Doungsa-ard [53]	GA	unit	-

4.2. RQ2 - How Convincing Is The Evidence of Cost and Effectiveness of Search-Based Software Testing Techniques, Based on Empirical Studies That Report Credible Results?

According to Ali et al. [3] for a paper to have convincing evidence, it has to comply to the criteria set in RQ1 and in addition to that:

- Studies must report proper descriptive statistics or statistical hypothesis testing results

After applying this to the 15 papers from the previous RQ, there were only 6 left. Of these 6 the most common comparison baseline was RS, all but one used this as a baseline. Three papers also used GA as a baseline. In addition half of the papers used more than one comparison baseline. When looking at the result highlights in 5 cases the proposed technique was superior to the baseline(s). In only one case the proposed algorithm did not prove to be better than RS. The test purpose varies in the papers, not all of them have the same goal. Two papers were focused on maximising the coverage result, one looked to find worst case scenarios, one looked to improve a MHS algorithm, one paper introduced a new method to generate test cases, and one focused on testing software product lines.

Table 4. Papers that provide convincing evidence for cost and effectiveness

ID	Authors	Test purpose	Comparison baseline	Result highlights
7	Blanco et al. [16]	Applying SS to automated test case generation to achieve high branch coverage	RS, GA (different versions), SS (different versions), TS	The authors found that the proposed algorithm generates fewer test cases to reach the same or better percentage of coverage

17	Di Alesio et al. [33]	Use GA and CP to automate the generation of test cases to search for worst case scenarios where tasks are likely to miss deadlines	GA, CP	The technique was tested on five different systems and the study concluded that the proposed technique (GA+CP) is nearly as efficient as GA and practically as effective as CP
21	Bueno et al. [38]	Present a new testing technique that can be applied to automated test data generation	RS	The proposed algorithm performed better (in terms of coverage) in 10 out of the 12 cases. The statistical results point out that the algorithm is more effective than RTS while only in one pair of coverage values is RTS more effective than the proposed algorithm. The proposed algorithm did not perform as well when test set sizes were defined smaller.
29	Henard et al. [46]	Minimize the cost (number of tests) and maximize pairwise coverage	RS	For the same (or higher) pairwise coverage the algorithm requires less products making the cost lower.
33	Doganay et al. [51]	Automatically generate test data to maximize MC/DC coverage for embedded control software	RS	The authors found that RS is more efficient in the majority of cases (HC outperforms RS in only about 30% of the cases). The results did not show a clear winner between the two approaches, but on average RS performed better.

34	Malburg and Fraser [34]	To overcome the disadvantages a MHS algorithm might possess with the help of constraint solving	RS, GA, DSE	The proposed method surpassed GA and RS significantly in branch coverage. When comparing to DSE, in 9 out of 20 examples the proposed algorithm achieved higher branch coverage, for the remaining 11 branch coverage was the same (no statistically significant difference)

RS – Random search, CP – constraint programming, DSE – dynamic symbolic execution, RTS – random test sets

4.3. RQ3 - How Well Is The Random Variation Inherent in Search-Based Software Testing, Accounted for in the Design of Empirical Studies?

To assess how well random variation is present in the analysed papers, there are first two categories: one where random variation is accounted for and the other is random variation is not accounted for. Like Ali et al. [3], a paper is considered to be in the first category if the number of runs is presented (and it is more than 10), sufficient evidence that the runs are independent, and the data analysis method used to compare MHS algorithms and the baseline is reported. Among the first category (random variation is accounted for) the papers are further divided into three categories:

1. Poor descriptive statistics – only the average of the result is reported
2. Good descriptive statistics – levels of variation or central tendencies are reported
3. Statistical data analysis – in addition to the previous category's ('Good') demands the paper had to report the results of a statistical hypothesis test and establish the statistical significance of differences

The second category (random variation not accounted for) is divided into two sub-categories: random variation not discussed or accounted of, insufficient number of runs.

Among the 37 papers analysed 8 did not account for random variation. All 8 of these were in the first sub-category – random variation not discussed or accounted for. In these papers most often the number of runs was not mentioned.

Of the remaining 29 that accounted for random variation, 16 were categorized under 'poor descriptive statistics'. These papers only described the average of their results. Three

papers were categorized as ‘Good descriptive statistics’. These papers brought out more than the ones in the previous category and analysed their techniques further. Statistical data analysis was found to be used in 10 papers. Some of the statistical test methods used were Mann-Whitney U test and t-test.

The distribution of the papers can also be seen in Table 5.

Table 5. Distribution of how random variation is reported in 37 papers

Random Variation Accounted For			Random Variation not accounted for	
Poor descriptive statistics	Good descriptive statistics	Statistical data analysis	Not discussed or accounted for	Insufficient number of runs
16	3	10	8	0

4.4. RQ4 - What Are the Most Common Alternatives to Which SBST Techniques Are Compared?

Baselines are important to show how the proposed algorithm works better and also justify why the proposed algorithm is needed.

In order for the results to be comparable, the alternative techniques are categorised the same way as Ali et al. [3], divided into four categories. These four categories are:

1. Baseline of comparison is a global MHS algorithm
2. Baseline of comparison is a local MHS algorithm
3. Baseline of comparison is not a SBST technique
4. Baseline of comparison was not discussed

Of the 37 papers only 3 did not have a comparison baseline. These papers proposed a new method for software test case generation, but did not compare it to anything or compared two or more MHS algorithms to each other.

A global MHS baseline was used in 31 papers. Out of these 31, using a genetic algorithm and its extensions was the most popular with 24 uses. In some papers the proposed algorithm was even compared to different versions of genetic algorithms. Particle swarm optimization was used in 5 papers and simulated annealing and its extensions in 2. Both of these were used significantly less than genetic algorithms.

Local MHS algorithms, TS, were used only once. Making this option (using local SBST techniques as a baseline) the least popular.

Using techniques that were not SBST techniques was almost as popular as using global MHS algorithms. These techniques were used in 21 papers. A random generator was used in 14. Constraint solving was used once. Other techniques were used in 6 papers. Among the non-SBST techniques using a random generator was the most popular.

All of these results can be seen in Table 6. In addition, a table containing all the papers, the MHS technique and the baseline that was used can be seen in the appendix 8.1.

Table 6. Comparison baselines

Global SBST technique			Local SBST technique	Non-SBST technique			Not discussed
GA+	SA+	PSO	TS	Random	Constraint	Others	
24	2	5	1	14	1	6	3

GA+ – genetic algorithm and its extensions, SA+ – simulated annealing and its extensions

4.5. RQ5 - What Are the Measures Used for Assessing Cost and Effectiveness of Search-Based Software Testing?

The cost and effectiveness can be measured in different ways, so the answer to this RQ is divided into two parts, first cost and then effectiveness.

4.5.1. Cost

Cost is one of the main factors driving the purpose to find better ways to automate test case generation.

Like in the paper by Ali et al. [3] the cost measures were divided into two categories:

1. Cost of finding the target (the cost of automating test case generation)
2. Cost of executing the generated test suite

In the category “cost of executing the generated test suite” the size of the test suite was the sub-category used by Ali et al. [3] and the same category was used here. The category “cost of finding the target” has the subcategories test case generation time, number of fitness evaluations, number of iterations (e.g. number of generations in genetic algorithms), and number of individuals (test cases). The same division was used here.

Of the 37 papers analysed 15 papers did not report using any cost measures, these papers oftentimes focused on the effectiveness of the proposed technique. Cost measures in the first category (cost of finding the target) were used 39 times. Among these 39, the number of iterations was used 13 times. The number of individuals was used 8 times and the number of fitness evaluations was used once. The most popular measure in this category was test time generated with 18 uses. In the second category, size of the test suite was used twice. The distribution of cost measures can be seen in Table 7.

Table 7. Measures of cost used in 37 papers

Cost of finding the target				Cost of executing the final suite	No cost measure
Number of iterations	Number of individuals	Number of fitness evaluations	Test case generation time	Size of test suite	
13	8	1	18	2	15

4.5.2. Effectiveness

In the paper by Ali et al. [3], the measures for effectiveness are divided into four categories. These categories are:

1. Coverage-based measures
2. Fault based measures
3. Others
4. No cost measure

The first category, coverage-based measures, is divided into three sub-categories: control-flow coverage criteria (branch, statement, path, condition and condition-decision coverage), data-flow coverage criteria (all-DU coverage), and n-wise coverage criteria (for MHS algorithms in combinatorial testing). There are no sub-categories for fault based measures. Under fault based measures mutation analysis was the main strategy used, thus mutation score was classified under here. A few papers used the number of faults found to assess the effectiveness, so these papers were also classified ‘fault based measures’. The other measures used in the analysed papers that did not qualify under the first two

categories were classified under the third category, ‘Others’. Within in this, ‘Time based measures’, ‘The fitness value of individuals’ and ‘Misc.’ are three subcategories identified by Ali et al. [3] and also used here. Papers that did not assess the effectiveness of the proposed technique were categorised under the last category (No cost measure).

Table 8. Distribution of effectiveness measures in 37 papers

Coverage-based measures			Fault based measures	Others			No effectiveness measure
Control-flow	Data flow	n-wise		Time based measures	Fitness value of individuals	Misc.	
17	2	1	5	1	2	6	5

Of the collected 37 papers there were 5 that did not report the effectiveness of the proposed technique at all. 22 papers used coverage based measures to assess the effectiveness of the proposed technique. Of those 22 papers, 17 used control-flow criteria, making this the most popular option within coverage based measures. Data-flow coverage criteria were used 2 times and n-wise coverage criteria were used only once. 5 papers used fault based measures to measure the effectiveness. Only one paper that was analysed used n-wise coverage criteria to evaluate the effectiveness. 8 papers were classified under using other effectiveness measures. Of these 8 papers one used ‘Time-based measures’ and two used ‘Fitness value of individuals’. 6 papers used other measures that could not be classified to any other category, but were still relevant to the paper. The distribution of effectiveness measures can also be seen in Table 8.

5. Discussion

In this chapter the results will be analysed and compared to those of the paper by Ali et al. [3].

5.1. RQ1

According to the results presented in the previous chapter (chapter 4.1), there are only 15 papers among 37 (40%) that provide credible evidence for assessing cost-effectiveness. In the paper by Ali et al. [3] this percentage was only 28%. According to these percentages the amount of papers that provide good evidence in the study of cost-effectiveness is rising. Based on Table 3 using GA is the most popular choice. When comparing this to the result found by Ali et al. [3] it can be seen that in their research GA was also the most popular choice (12 out of 18 papers used GA), this was also the most popular MHS algorithm used to generate test cases both in this thesis (26 times) and in the paper by Ali et al. [3].

5.2. RQ2

There were 6 papers out of 37 that have sufficient criteria to be categorized as having credible evidence. That is only 16% when in comparison Ali et al. [3] found that 14% (8 papers out of 64) had enough evidence. Percentage-wise the difference is not very large, meaning that during the past 8 (2008-2015) years not much has changed and the level of providing convincing evidence of cost-effectiveness has not changed. As in the paper by Ali et al. [3] the test purposes vary, meaning that MHS algorithms can be applied to different problems.

5.3. RQ3

As shown in Table 5, 78% of the papers accounted for random variation. Because random variation helps convince the reader of the stability of the results of the proposed technique, it is encouraging that this number is 78%. In comparison Ali et al. found that 39 papers of 64 (61%) accounted for random variation, meaning this number has risen in the past years. The percentage of papers in the poor descriptive statistics column of Table 5 has remained fairly similar with a slight increase in the recent years, in this thesis the percentage is 43% (of all the papers) and in the paper by Ali et al. [3] it was 38%. The number of papers in the good descriptive statistics category is alarmingly small, only 8% of all the papers qualified for this. The number of papers in the statistical data analysis category was higher

than expected in this paper, 27% of the papers qualified under this, meaning that researchers are putting more emphasis on confirming their proposed techniques efficiency or effectiveness by statistical methods. In the paper by Ali et al. [3] only 11% on the papers could be classified as having done statistical data analysis. This means that more researchers are inclined to do statistical data analysis now than before. Among the papers analysed in this thesis 22% did not account for random variation at all, whereas in then the paper by Ali et al. [3] 39% did not account for variation.

5.4. RQ4

Based on the numbers in Table 6 it can be said that using GA is the most popular choice to compare the different MHS algorithms to. Since this is also one of the most popular algorithms to use to automate test data generation, it makes sense to compare a new enhanced GA to a previous, maybe even dated version of GA. The second most popular comparison baseline is random search, with 14 uses. When introducing a new approach or applying a MHS algorithm to a new aspect in test case generation, it is difficult to compare the algorithm to anything if the area doesn't have any previous research done. In this case it is beneficial to use random search. When comparing the results shown in Table 6 to those of the paper by Ali et al. [3] it can be seen that using GA is not the most popular option there (22 uses out of 70), but using random search is (24/70). Although the two most popular has remained the same, they have switched places. The difference between GA and random is not that large in the paper by Ali et al. [3] (only 2 uses), whereas in this thesis the difference is more noticeable (8 uses). This might indicate that using GA as a baseline has become more prominent and could act as a better baseline.

5.5. RQ5

59% on the papers analysed in this thesis used cost measures of some kind. 22 papers used 1.91 cost measures on average per paper meaning that the tendency is rather to use more than one cost measure in an empirical review. Among the results using the test case generation time was the most popular. The number of iterations (most often the number of test cases) was the second most popular cost measure, but this measure is also one of the least precise. When looking at the results found by Ali et al. [3], they found that the most popular cost measure was the number of iterations. In second place was test case generation time followed closely by the number of fitness evaluations. The two most popular cost measures in this thesis and the paper by Ali et al. [3] are the same, but in

reverse order. The number of iterations was used 5 times less than test case generation time, whereas in Ali et al. [3] the difference was 12, a much larger difference. According to this the type of measures used has not changed, the scales have tipped more in the favour of using the number of iterations as a cost measure.

Based on the data in Table 8 it can said that using control-flow based coverage criteria is the most popular choice, it precedes the other options by a large margin (no other measure reaches double digits). As the paper by Ali et al. [3] stated, using control-flow based coverage has been researched a lot and is thus a widely accepted standard to use in fitness functions. Like in this paper, the paper by Ali et al. [3] found that the most popular effectiveness measure is control-flow coverage criteria, meaning that using this measure was a popular choice, still is and probably will continue to be common practice. Using fault based measures was not very popular among the 37 papers analysed, only 5 uses. Similarly in the paper by Ali et al. [3] using fault based measures was not very popular. No effectiveness measure was used only 5 times, in these papers the main focus was on the cost of the effectiveness. Ali et al. [3] found that no effectiveness measures were used 3 times (out of 64 papers). In percentages, this thesis no effectiveness measure was used in 14% of the papers and in 5% according to Ali et al. [3]. This could mean that measuring the effectiveness has become less popular.

5.6. Threats to Validity

The original search string used by Ali et al. [3] was not modified in this thesis, it is possible that some relevant papers used other synonyms and did not turn up in the search. The paper selection step was conducted only by one person, meaning that when in doubt whether to exclude or include a paper, the paper was not discussed with anybody else and so some relevant papers may have been discarded and mistakes could have been made when extracting data without noticing. In addition the papers that were analysed and named in the paper by Ali et al. [3] influenced the selection process.

6. Conclusions

The aim of this thesis was to partially replicate a literature survey conducted by Ali et al. [3] in 2010. Ali et al.'s study reviewed empirical studies conducted in the realm of search-based software testing from 1997 to 2007. To complement the original study, in this thesis, the timeframe 2008-2015 was used.

Based on the results found in this thesis and the discussion it can be deducted that most points of interest looked at remained fairly the same as in Ali et al. [3] or had a slight fluctuation to the positive or negative side. RQ1 and RQ2 produced similar results to the paper by Ali et al. [3] in this thesis. The number of papers that could qualify for these two questions was higher in this thesis than in Ali et al. [3]. The most popular MHS algorithm that was used in the papers selected for answering these two research questions was GA and the most popular baseline RS. This was also the case in the paper by Ali et al. [3]. The most difference could be found when looking at the data for RQ3, accounting for random variation. In this thesis 27% of the papers provided statistical data analysis, whereas only 11% did this in the paper by Ali et al. [3]. Baselines to which the proposed algorithms were compared to in this thesis and in the paper by Ali et al. [3] followed very similar tendencies. The only difference was that the two most popular choices had switched places. In this paper the most popular was GA, followed by RS, whereas in the paper by Ali et al. [3] the order was reversed, meaning that GA has become more popular for comparing new approaches. The measures for cost found in the papers analysed in this thesis were categorised the same way as in [3] and the results indicated that the two most popular measures (test case generation time and number of iterations) are the same as in Ali et al. [3] although the scales have tipped in favour or using test case generation. The distribution of measures of effectiveness found in this paper and the results from Ali et al. [3] indicate that the most popular effectiveness measure in both papers remains the same - control-flow coverage criteria. The only downside was that more papers did not use any effectiveness measures in the 37 papers analysed, than in the paper by Ali et al. [3].

7. Bibliography

[1] S. Anand, E. K. Burke , T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold and P. McMinn, An orchestrated survey of methodologies for automated software test case generation, *Journal of Systems and Software,* vol. 86, no. 8, pp. 1978-2001, 2013.

[2] Software Testing Life Cycle (STLC), http://softwaretestingfundamentals.com/software-testing-life-cycle/. (17 March 2017).

[3] S. Ali, L. C. Briand, H. Hemmati and R. K. Panesar-Walawege, A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation, *IEEE Transactions on Software Engineering,* vol. 36, no. 6, pp. 742-762, 2010.

[4] A Course in Black Box Software Testing - Examples of Test Oracles, http://www.testingeducation.org/k04/OracleExamples.htm. (22 April 2017).

[5] What Is the Genetic Algorithm?, https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html (30 April 2017).

[6] A. C. Monção, G. C. Camilo-Jr, L. T. Queiroz, C. L. Rodrigues, P. de Sá Leitão-Jr and A. M. Vincenzi, Applying genetic algorithms to data selection for SQL mutation analysis, in *GECCO '13 Companion Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, Amsterdam, 2013.

[7] D. Jeya Mala and V. Mohan, *Quality improvement and optimization of test cases: a hybrid genetic algorithm based approach,* vol. 35, 2010, pp. 1-14.

[8] What Is Simulated Annealing?, https://www.mathworks.com/help/gads/what-is-simulated-annealing.html (30 April 2017).

[9] K. Hou, J. Huang and X. Bai, Geographical Test Data Generation by Simulated-Annealing, in *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, Taichung, 2013.

[10] B. L. Li, S. Li, J. Y. Zhang and J. R. Sun, An Automated Test Case Generation Approach by Genetic Simulated Annealing Algorithm, in *Third International Conference on Natural Computation (ICNC 2007)*, Haikou, 2007.

[11] J. Qiu, X. Tan, G. Liu and K. Lü, Test selection and optimization for PHM based on failure evolution mechanism model, *Journal of Systems Engineering and Electronics,* vol. 24, no. 5, pp. 780-792, 2013.

[12] A. Li and Y. Zhang, Automatic Generating All-Path Test Data of a Program Based on PSO, in *2009 WRI World Congress on Software Engineering*, Xiamen, 2009.

[13] L. S. d. Souza, R. B. C. Prudêncio and F. d. A. Barros, A Hybrid Binary Multi-objective Particle Swarm Optimization with Local Search for Test Case Selection, in *2014 Brazilian Conference on Intelligent Systems*, Sao Paulo, 2014.

[14] C. Mao, L. Xiao, X. Yu and J. Chen, Adapting ant colony optimization to generate test data for software structural testing, *Swarm and Evolutionary Computation,* vol. 20, pp. 23-36, 2015.

[15] F. Glover, Heuristics for integer programming using surrogate constraints, in *Decision Sciences*, 1977, pp. 159-166.

[16] R. Blanco, T. Javier and B. Adenso-Diaz, Automated test data generation using a

scatter search approach, *Information and Software Technology,* vol. 51, no. 4, pp. 708-720, 2009.

[17] F. C. M. Souza, M. Papadakis, Y. Le Train and M. E. Delamaro, Strong Mutation-Based Test Data Generation Using Hill Climbing, in *2016 IEEE/ACM 9th International Workshop on Search-Based Software Testing (SBST)*, Austin.

[18] N. Sharma, Sujata and G. N. Purohit, Test case prioritization techniques an empirical study, in *2014 International Conference on High Performance Computing and Applications (ICHPCA)*, Bhubaneswar, 2014.

[19] C. Wohlin, Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering, *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering,* p. ??, 2014.

[20] R. Blanco, J. Garcia-Fanjul and J. Tuya, A First Approach to Test Case Generation for BPEL Compositions of Web Services Using Scatter Search, in *IEEE International Conference on Software Testing Verification and Validation Workshops*, Denver, 2009.

[21] E. Díaz, J. Tuya, R. Blanco and J. J. Dolado, A tabu search algorithm for structural software testing, *Computers & Operations Research,* vol. 35, no. 10, pp. 3052-3072, 2008.

[22] Y. Cao, C. Hu and L. Li, An approach to generate software test data for a specific path automatically with genetic algorithm, in *2009 8th International Conference on Reliability, Maintainability and Safety*, Chengdu, 2009.

[23] A. Aburas and A. Groce, An Improved Memetic Algorithm with Method Dependence Relations (MAMDR), in *2014 14th International Conference on Quality Software*, Dallas, 2014.

[24] A. Rathore, A. Bohara, R. G. Prashil, T. S. L. Parashatnth and P. R. Srivastava, Application of Genetic Algorithm and Tabu Search in Software Testing, in *COMPUTE '11 Proceedings of the Fourth Annual ACM Bangalore Conference*, Bangalore, 2011.

[25] Y. Dong and J. Peng, Automatic generation of software test cases based on improved genetic algorithm, in *2011 International Conference on Multimedia Technology*, Hangzhou, 2011.

[26] R. Ding, X. Feng, S. Li and H. Dong, Automatic generation of software test data based on hybrid particle swarm genetic algorithm, in *2012 IEEE Symposium on Electrical & Electronics Engineering (EEESYM)*, Kuala Lumpur, 2012.

[27] B. Zhang and C. Wang, Automatic generation of test data for path testing by adaptive genetic simulated annealing algorithm, in *2011 IEEE International Conference on Computer Science and Automation Engineering*, Shanghai, 2011.

[28] Y. Chen and Y. Zhong, Automatic Path-Oriented Test Data Generation Using a Multi-population Genetic Algorithm, in *2008 Fourth International Conference on Natural Computation*, Jinan, 2008.

[29] P. Maragathavalli, S. Kanmani, J. S. Kirubakar, P. Sriraghavendrar and A. Sai Prasad, Automatic program instrumentation in generation of test data using genetic algorithm for multiple paths coverage, in *IEEE-International Conference On Advances In Engineering, Science And Management (ICAESM -2012)*, Nagapattinam, Tamil Nadu, 2012.

[30] R. Khan and M. Amjad, Automatic test case generation for unit software testing using genetic algorithm and mutation analysis, in *2015 IEEE UP Section Conference on*

Electrical Computer and Electronics (UPCON), Allahabad, 2015.

[31] F. q. Wei and S. j. Jiang, Automatic Test Data Generation Based on SAMPSO Algorithm, in *2009 International Conference on Computational Intelligence and Software Engineering*, Wuhan, 2009.

[32] G. I. Latiu, O. A. Cret and L. Vacariu, Automatic Test Data Generation for Software Path Testing Using Evolutionary Algorithms, in *2012 Third International Conference on Emerging Intelligent Data and Web Technologies*, Bucharest, 2012.

[33] S. Di Alesio, L. C. Briand, S. Nejati and A. Gotlieb, Combining Genetic Algorithms and Constraint Programming to Support Stress Testing of Task Deadlines, *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 25, no. 1, 2015.

[34] J. Malburg and G. Fraser, Search-based testing using constraint-based mutation, *Software Testing, Verification and Reliability,* vol. 24, no. 6, pp. 472-495, 2014.

[35] K. Ghani, J. A. Clark and Y. Zhan, Comparing algorithms for search-based test data generation of Matlab® Simulink® models, in *2009 IEEE Congress on Evolutionary Computation*, Trondheim, 2009.

[36] Y. Chen, Y. Zhong, T. Shi and J. Liu, Comparison of Two Fitness Functions for GA-Based Path-Oriented Test Data Generation, in *2009 Fifth International Conference on Natural Computation*, Tianjin, 2009.

[37] D. Jeya Mala, K. Sabari Nathan and S. Balamurugan, Critical Components Testing Using Hybrid Genetic Algorithm, *ACM SIGSOFT Software Engineering Notes,* vol. 38, no. 5, pp. 1-13, 2013.

[38] P. M. S. Bueno, M. Jino and W. E. Wong, Diversity oriented test data generation using metaheuristic search techniques, *Information Sciences,* vol. 259, pp. 490-509, 2014.

[39] A. El-Serafy, G. El-Sayed, C. Salama and A. Wahba, Enhanced Genetic Algorithm for MC/DC test data generation, in *2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA)*, Madrid, 2015.

[40] Y. Suresh and S. K. Rath, Evolutionary Algorithms for Object-Oriented Test Data Generation, *ACM SIGSOFT Software Engineering Notes,* vol. 39, no. 4, pp. 1-6, 2014.

[41] L. S. Silva and M. van Someren, Evolutionary Testing of Object-Oriented Software, in *SAC '10 Proceedings of the 2010 ACM Symposium on Applied Computing*, Sierre, 2010.

[42] M. A. Ahmed and I. Hermadi, GA-based multiple paths test data generator, *Computers & Operations Research,* vol. 35, no. 10, pp. 3107-3124, 2008.

[43] B. S. Ahmed, M. A. Sahib and M. Y. Potrus, Generating combinatorial test cases using Simplified Swarm Optimization (SSO) algorithm for automated GUI functional testing, *Engineering Science and Technology, an International Journal,* vol. 17, no. 4, pp. 218-226, 2014.

[44] C. Mao, X. Yu, J. Chen and J. Chen, Generating Test Data for Structural Testing Based on Ant Colony Optimization, in *2012 12th International Conference on Quality Software*, Xi'an, Shaanxi, 2012.

[45] K. Liaskos and M. Roper, Hybridizing Evolutionary Testing with Artificial Immune Systems and Local Search, in *2008 IEEE International Conference on Software Testing Verification and Validation Workshop*, Lillehammer, 2008.

[46] C. Henard, M. Papadakis, G. Perrouin, J. Klein and Y. Le Traon, Multi-objective Test Generation for Software Product Lines, in *SPLC '13 Proceedings of the 17th International Software Product Line Conference*, Tokyo, 2013.

[47] F. M. Kifetew, A. Panichella, A. De Lucia, R. Oliveto and P. Tonella, Orthogonal Exploration of the Search Space in Evolutionary Test Case Generation, in *ISSTA 2013 Proceedings of the 2013 International Symposium on Software Testing and Analysis*, Lugano, 2013.

[48] P. Flores and Y. Cheon, PWiseGen: Generating test cases for pairwise testing using genetic algorithms, in *2011 IEEE International Conference on Computer Science and Automation Engineering*, Shanghai, 2011.

[49] Y. Xiangjuan, G. Dunwei and W. Wenliang, Test Data Generation for Multiple Paths Based, *Chinese Journal of Electronics,* vol. 24, no. 1, pp. 46-51, 2015.

[50] A. Panichella, F. M. Kifetew and P. Tonella, Reformulating Branch Coverage as a Many-Objective Optimization Problem, in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, Graz, 2015.

[51] K. Doganay, M. Bohlin and O. Sellin, Search Based Testing of Embedded Systems Implemented in IEC 61131-3: An Industrial Case Study, in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, Luxembourg, 2013.

[52] S. Jiang, Y. Zhang and D. Yi, Test Data Generation Approach for Basis Path Coverage, *ACM SIGSOFT Software Engineering Notes,* vol. 37, no. 3, pp. 1-7, 2012.

[53] K. Marin and C. Doungsa-ard, Test data generation from Hibernate constraints, in *The 8th International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2014)*, Dhaka, 2014.

[54] D. M. Cohen, S. R. Dalal, M. L. Fredman and G. C. Patton, The AETG system: an approach to testing based on combinatorial design, *IEEE Transactions on Software Engineering,* vol. 23, no. 7, pp. 437-444, 1997.

[55] A. W. Williams and R. L. Probert, A practical strategy for testing pair-wise coverage of network interfaces, in *Software Reliability Engineering, 1996. Proceedings., Seventh International Symposium on*, White Plains, 1996.

[56] A. Hartman and L. Raskin, Problems and algorithms for covering arrays, *Discrete Mathematics,* vol. 284, no. 1-3, pp. 149-156, 2004.

8. Appendix

8.1. Algorithms and Baselines

Table 9. Table Containing All MHS Algorithms and Baselines Used

ID	Title	Authors	MHS algorithm used	Comparison baseline
1	A First Approach to Test Case Generation for BPEL Compositions of Web Services Using Scatter Search	Blanco et al. [20]	SS	RS
2	A tabu search algorithm for structural software	Díaz et al. [21]	TS	RS
3	Adapting ant colony optimization to generate test data for software structural testing	Mao et al. [14]	ACO	SA, GA, PSO
4	An approach to generate software test data for a specific path automatically with genetic algorithm	Cao et al. [22]	GA	RS, GA
5	An Improved Memetic Algorithm with Method Dependence Relations (MAMDR)	Aburas and Groce [23]	GA, HC	GA
6	Application of Genetic Algorithm and Tabu Search in Software Testing	Rathore et al. [24]	GA, TS	GA
7	Automated test data generation using a scatter search approach	Blanco et al. [16]	SS	RS, GA, SS, TS
8	Automatic Generating All-Path Test Data of a Program Based on PSO	Li and Zhang [12]	PSO	PSO
9	Automatic generation of software test cases based on improved genetic algorithm	Dong and Peng [25]	GA	GA
10	Automatic generation of software test data based on hybrid particle swarm genetic algorithm	Ding et al. [26]	PSO, GA	GA
11	Automatic generation of test data for path testing by adaptive genetic simulated annealing algorithm	Zhang and Wang [27]	GA, SA	GA

12	Automatic Path-Oriented Test Data Generation Using a Multi-population Genetic Algorithm	Chen and Zhong [28]	GA	GA
13	Automatic program instrumentation in generation of test data using genetic algorithm for multiple paths coverage	Maragathavalli et al. [29]	GA	GA
14	Automatic test case generation for unit software testing using genetic algorithm and mutation analysis	Khan and Amjad [30]	GA	Mutation testing
15	Automatic Test Data Generation Based on SAMPSO Algorithm	Wei and Jiang [31]	PSO	GA, BPSO
16	Automatic Test Data Generation for Software Path Testing Using Evolutionary Algorithms	Latiu et al. [32]	GA, SA, PSO	-
17	Combining Genetic Algorithms and Constraint Programming to Support Stress Testing of Task Deadlines	Di Alesio et al. [33]	GA	GA, CP
18	Comparing algorithms for search-based test data generation of Matlab® Simulink® models	Ghani et al. [35]	GA	RS, DSE, GA
19	Comparison of Two Fitness Functions for GA-Based Path-Oriented Test Data Generation	Chen et al. [36]	GA, SA	-
20	Critical Components Testing Using Hybrid Genetic Algorithm	Jeya Mala et al. [37]	GA	-
21	Diversity oriented test data generation using metaheuristic search techniques	Bueno et al. [38]	MA	GA
22	Enhanced Genetic Algorithm For MC/DC Test Data Generation	El-Serafy et al. [39]	SA, GA, SR	RS
23	Evolutionary Algorithms for Object-Oriented Test Data Generation	Suresh et al. [40]	GA	GA
24	Evolutionary Testing of Object-Oriented Software	Silva and van Someren [41]	BPSO, ABC	Selection algorithm
25	GA-based multiple paths test data generator	Ahmed and Hermadi [42]	GA	RS
26	Generating combinatorial test cases using Simplified Swarm Optimization (SSO) algorithm for automated GUI functional testing	Ahmed et al. [43]	GA	GA
27	Generating Test Data for Structural Testing Based on Ant Colony Optimization	Mao et al. [44]	SSO	PIST, TVG, CTE-XL, ITCH, IPOG, PSO

28	Hybridizing Evolutionary Testing with Artificial Immune Systems and Local Search	Liaskos and Roper [45]	ACO	SA, GA
29	Multi-Objective Test Generation for Software Product Lines	Henard et al. [46]	GA	GA
30	Orthogonal Exploration of the Search Space in Evolutionary Test Case Generation	Kifetew et al. [47]	GA	RS
31	PWiseGen: Generating test cases for pairwise testing using genetic algorithms	Flores and Cheon [48]	GA	GA
32	Reformulating Branch Coverage as a Many-Objective Optimization Problem	Panichella et al. [50]	GA	GA, AETG [54], IPO, TConfig [55], CTS [56]
33	Search Based Testing of Embedded Systems Implemented in IEC 61131-3: An Industrial Case Study	Doganay et al. [51]	GA	RS, GA
34	Search-based testing using constraint-based mutation	Malburg and Fraser [34]	GA	GA
35	Test Data Generation Approach for Basis Path Coverage	Jiang et al. [52]	HC	RS
36	Test Data Generation for Multiple Paths Based on Local Evolution	Xiangjuan et al. [49]	GA	GA
37	Test Data Generation From Hibernate Constraints	Marin and Doungsa-ard [53]	GA	RS

RS – random search, BPSO - , CP – constraint programming, DSE – dynamic symbolic execution, SR - simulated repulsion, ABC - artificial bee colony algorithm , SSO – simplified swarm optimization, PIST, TVG – Test Vector Generator, CTE-XL – Classification-Tree Editor eXtended Logics, ITCH - Intelligent Test Case Handler, IPOG/IPO – In Parameter Order Generator

8.2. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, Heidi Urmet,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation,

supervised by Dietmar Alfred Paul Kurt Pfahl,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **10.05.2017**