A Systematic Review of the Application and Empirical Investigation of Evolutionary Testing

S. ALI, L. C. BRIAND, H. HEMMATI, and R. K. PANESAR-WALAWEGE

***Abstract*—Metaheuristic search techniques have been extensively used to automate the process of generating test cases and thus providing solutions for a more cost-effective testing process. This approach to test automation, often coined as "Evolutionary Testing" (ET), has been used for a wide variety of test case generation purposes, differing in terms of test objectives, test levels, and other characteristics. Since ET techniques are heuristic by nature, they must be empirically investigated in terms of how costly and effective they are at reaching their test objectives and whether they scale up to realistic development artifacts. However, approaches to empirically study ET techniques have shown wide variation in the literature.**

This paper presents the results of a systematic, comprehensive review that aims at characterizing how empirical studies have been designed to investigate ET cost-effectiveness and what empirical evidence is available in the literature regarding ET cost-effectiveness and scalability. We also provide a framework that drives the data collection process of this systematic review and can be the starting point of guidelines on how ET techniques can be empirically assessed.

***Index Terms*— Evolutionary computing and genetic algorithms, Frameworks, Heuristics design, Review and evaluation, Test generation, Testing strategies, Validation.**

I. INTRODUCTION

SOFTWARE is incorporated into ever increasing numbers of systems and hence it is becoming increasingly important to thoroughly test these systems. One challenge to testing software systems is the effort involved in creating test cases that will systematically test the system and reveal faults in an effective manner. The overall testing cost has been estimated at being almost fifty percent of the entire development cost [2], if not more. Thus, a logical response is to automate the testing process as much as possible, and test case generation is naturally a key part of this automation. A possible strategy which has drawn great interest in the automation of test case generation is the application and tailoring of metaheuristic search (MHS) algorithms [5]. The main reason for such an interest is that test case generation problems can often be re-expressed as optimization or search problems.

This work was supported by the Simula Research Laboratory, Norway.

S. Ali, L.C. Briand, H. Hemmati, R. K. Panesar-Walawege are with Simula Research Laboratory and University of Oslo, Norway, P.O. Box 134, 1325 Lysaker, Norway (telephone: +47 47466831, e-mail: {shaukat, briand, hemmati, rpanesar@simula.no}).

There has been a tremendous amount of research in applying MHS algorithms to test case generation and a large body of research exists: 121 articles were published over the years 1996-2007 on this specific topic, often referred to as evolutionary testing (ET) [7].

Seeing the amount of research activity in this field, it is at this point in time, highly important to characterize what type of research has been performed and how it has been conducted. Among other things, it is crucial to appraise how much empirical evidence there is regarding the cost-effectiveness of ET and to determine whether there is room for improvement in the way studies are performed and reported. The ultimate goal is to improve the quality of future research in this important, emerging field of research. In order to assess the current state of the art in ET, we decided to conduct a systematic, comprehensive review of the current literature, as this is commonly done in other scientific fields of research such as medicine [9] and social science [10]. The purpose of this systematic review is to collect, classify, and assess the empirical studies of ET in order to assess the current body of evidence regarding the cost and effectiveness of ET. By identifying the strengths and weaknesses of the current literature we hope to suggest improved research practices and relevant future research directions.

This paper is organized as follows: In Section II, we provide the background relevant to the material presented in this paper. Section III suggests a framework used to assess the empirical studies in ET and Section IV presents the method used to conduct this systematic review. In Section V, we present the results of our review whilst Section VI outlines its validity threats. The final conclusions that we can draw from this systematic review are presented in Section VII.

II. BACKGROUND

In this systematic review, we are analyzing what test case generation problems have been addressed by MHS algorithms and what body of evidence exists regarding their cost effectiveness. As a preliminary to the review itself, we introduce here the three main components involved in this paper: evolutionary testing, systematic reviews, and empirical studies.

A. Evolutionary Testing Techniques

The main aim of software testing is to detect as many faults as possible, especially the most critical ones, in the system under test (SUT). To gain sufficient confidence that most faults are detected, testing should ideally be exhaustive. Since in practice this is not possible, testers resort to test models and coverage/adequacy criteria to define systematic and effective test strategies that are fault revealing. A test case normally consists of test data and the expected output [13]. The test data can take various forms such as values for input parameters of a function, values of input parameters for a sequence of method calls, or seeding times to trigger task executions. In the context of this review, we are not dealing with the expected outputs, but focus exclusively on the generation of test data as this has been the objective of papers making use of ET. In order to perform test case generation, systematically and efficiently, automated test case generation strategies are employed. Bertolino [15] addresses the need for 100% automatic testing as a means to improve the quality of complex software systems that are becoming the norm of modern society. A comprehensive testing strategy must address many activities that should ideally be automated: the generation of test requirements, test case generation, test oracle generation, test case selection, or test case prioritization. In our current review, we are only dealing with test case generation. A promising strategy for tackling this challenge comes from the field of search-based software engineering [18].

Search-based software engineering attempts to solve a variety of software engineering problems by reformulating them as search problems [21]. A major research area in this domain is the application of MHS algorithms to test case generation. MHS algorithms are a set of generic algorithms that are used to find optimal or near optimal solutions to problems that have large complex search spaces [21]. There is a natural match between MHS algorithms and software test case generation. The process of generating test cases can be seen as a search or optimization process: there are possibly hundreds of thousands of test cases that could be generated for a particular SUT and from this pool we need to select, systematically and at a reasonable cost, those that comply to certain coverage criteria and are expected to be fault revealing, at least for certain types of faults. Hence, we can reformulate the generation of test cases as a search that aims at finding the required or optimal set of test cases from the space of all possible test cases. When software testing problems are reformulated into search problems, the resulting search spaces are usually very complex, especially for realistic or real-world SUTs. For example, in the case of white-box testing, this is due to the non-linear nature of software resulting from control structures such as if-statements and loops [7]. In such cases, simple search strategies may not be sufficient and MHS algorithms may, as a result, become a necessity. The use of MHS algorithms for test case generation is referred to as evolutionary testing [7]. McMinn [27] provides a survey of some of the MHS algorithms that have been used for test data generation. The most common MHS algorithms that have been employed for evolutionary testing are evolutionary algorithms, simulated annealing, ant colony optimization, and particle swarm optimization [29]. Among the MHS algorithms, hill climbing [29] is special as it is a very simple, local search algorithm. The ET techniques using complex MHS algorithms are usually compared with test case generation based on hill climbing and random search to determine whether their complexity is warranted to address a specific test case generation problem. For the simplicity of discussion, in the rest of the paper, we will refer to MHS algorithms in a slightly more restrictive way than usual by excluding hill climbing algorithms from this set.

B. Systematic Reviews

Systematic reviews are a means of synthesizing existing research regarding a specific research question [12]. They are usually performed to summarize the existing evidence for a particular topic and aid in the identification of gaps in the current research and thus can form the basis of new research activity. A review protocol is created at the beginning of the review, which lays out the research questions being answered and the methodology that will be used to answer these questions. The protocol specifies a specific search strategy that is used to select as much of the relevant literature as possible and provides justification for why studies are included or excluded from the systematic review. The data to be collected to answer the research questions is also presented in the protocol. All this information is published so that readers can judge the completeness of the systematic review, and if necessary replicate it. These features distinguish the systematic review from the usual literature review or survey that is usually conducted at the beginning of a research activity. A systematic review synthesizes the existing work in a systematic, comprehensive, and unbiased manner.

C. Empirical Studies for Evolutionary Testing

Kitchenham et al. [31] make the case for evidence-based software engineering that seeks to help practitioners make informed decisions related to software development and maintenance by integrating current best evidence from research with practical experience. Thus, to determine if ET techniques can be applied in practice, we need to conduct empirical studies to assess their cost-effectiveness and scalability. The cost-effectiveness of an ET technique is normally measured in terms of the ability of the technique to generate test cases that achieve a certain testing objective at a reasonable cost. The testing objective, as is the case with any test case generation technique, is to detect faults of a type that is explicitly defined or implicitly determined by the test model (e.g., state transition faults for a state machine model). In this review, we have focused on empirical studies of ET techniques in order to assess whether convincing evidence exists to show their cost-effectiveness and scalability. For this purpose, it was necessary to define what we mean by an empirical study in this context and what constitutes a well designed and reported empirical study. Empirical studies are usually divided into three different types: surveys, case studies or experiments [32]. For this review, we have used a broad definition of empirical study, to include any kind of empirical evaluation that has been done in the field of ET.

As to what constitutes a proper empirical study in ET, we looked at what advice has been given previously [32-34] for conducting empirical studies in software engineering and we have used the existing general guidelines to create a specific framework for conducting and reporting empirical studies in the domain of ET. This was necessary as ET studies involves the analysis of automation techniques in which no human subjects are involved and therefore presents many specific challenges. In addition, the fact that ET techniques are based on MHS algorithms makes it important to account for the inherent random variation that exists in the results. Furthermore, there should also be some means to show that an ET technique is really necessary for the context that it is being applied in. This can be done, for example, by showing that other simpler search techniques do not perform as well. The reason for doing this is that we want to ensure that the problems being tackled by the ET techniques do warrant their use.

The framework was created for a dual purpose. First, it was used in this systematic review to direct the collection of data that was used to assess the current state of empirical research in ET. Second, it can also be used as a set of guidelines for conducting and reporting future research in the field or at least as a starting point in the development of such guidelines. The next section will present the framework.

III. FRAMEWORK

As presented here, this framework is not intended to provide complete operational guidelines, but rather to justify the data collection that took place to perform the systematic review presented in the next sections and to highlight some of the most important concepts and issues.

The framework is divided into four parts. First, the test problem addressed must be clearly specified. Second, the MHS algorithms adopted must be clearly defined. Third, since any ET research should always include empirical studies aiming at assessing the cost and effectiveness of the proposed approaches, the design of such studies must be carefully described so that its validity can be assessed. Last, results must be carefully reported so as to be clearly interpretable.

A. Test Problem Specification

The test problem specification includes two main parts, the purpose of testing and the test strategy that will be employed. Each of these parts directly affects the form that the evolutionary testing strategy will take. Fig. 1 outlines the constituent parts of a test problem specification.

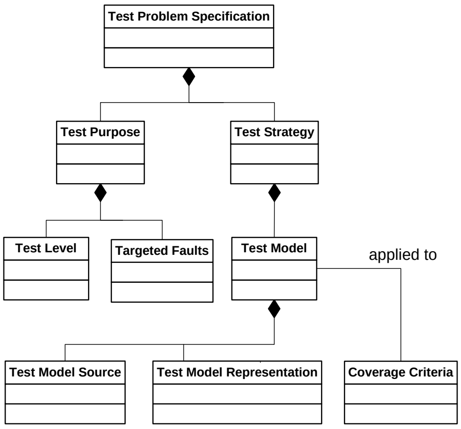

Fig. 1. : Concept diagram of test problem

1. *Test Purpose*

The general purpose of software testing is to gain sufficient confidence in the dependability of a software artifact by detecting as many (critical) faults as possible. However, testing strategies are usually not universal and normally target specific types of faults at different levels (such as unit, integration, and system testing).

a. *Test Level*

The Encyclopedia of Software Engineering [35] describes four major levels at which testing is conducted. These are unit testing, integration testing, system testing, and acceptance testing. Though standard definitions exist for the different levels of testing, clarifications should be provided where necessary. For example, the definition of what constitutes a ’unit’ in unit testing may vary from one context to another. For instance, typical examples of a unit include: a method in a procedural program, a class in an object oriented program, and a component in a component based system.

b. *Targeted Faults*

The targeted fault can be categorized in many ways depending on the view one takes of a system. At the highest level, one differentiates functional from non-functional faults, e.g., faults related to performance, security, robustness, and safety requirements.

2. *Test Strategy*

A testing strategy is defined by a model of the software under test (SUT) and some specific coverage criteria defined on that model. Such a model is typically referred to as a test model and the coverage criteria aim at systematically exercising the SUT based on the test model. This test model may be characterized by its source and representation (i.e., notation and semantics).

Coverage criteria definitions depend on the test model representation.

a. Test Model source

This defines the source used by a test strategy to create a test model. The source of the model implies constraints on the application of the test strategy as it depends on the availability and reliability of precise information in a specific form. Possible sources for a test model are briefly described below:

1. SUT specification

Black-box testing strategies use specifications as the model source and do not require knowledge about the internal structure of the SUT. For functional testing, these testing strategies are typically based on input-output relationships. Inputs are fed to the SUT and outputs are observed to determine success or failure of test cases. Inputs and outputs are selected from specifications in a systematic manner, following a strategy aiming at effectively detecting faults.

2. SUT design

Test models can be constructed from software design information, that is more precise information about the components of the SUT (i.e., subsystems, classes and their public interfaces), their state behavior (if any), and their interactions during the execution of use cases. For example, using a component diagram in component interaction testing or using a state machine for behavioral testing is considered using design information as a test model source.

3. SUT Source code

White-box testing strategies use source code as the source for the test model. White-box testing strategies (also known as structural testing techniques) aim at generating test cases to cover structural properties of source code such as statements, branches, or conditions. Most of these techniques involve constructing control or data flow graphs for test case generation. Mutation-based testing techniques can use either the source code or design artifacts as the model source depending on how the mutation operators are defined and where the faults are seeded.

b. Test model representation

Based on the model source (specification, design or source code), different types of test models can be constructed. An example of the test model representation from specifications is “Input domain representation”. This includes value ranges of relevant input variables and constraints on them. Black-box test strategies such as boundary value analysis and equivalence partitioning have this type of representation. Some examples of notations used for defining test models based on SUT design include state machines or Markov Usage models. Standard notations often have to be augmented to be test-ready, that is, to allow automated test case generation, e.g., guard conditions in a precise language for state machines. Typical examples of models derived from source code include control and data flow graphs. For mutation testing, the list of mutants generated from the mutation operators and to be detected by testing can be seen as the “test model”.

c. Coverage criteria

To be systematic, a test strategy generates test cases to cover certain features of the test model. For instance, in the case of state machines, typical coverage criteria include the coverage of all states or all transitions, the latter being a stronger requirement. In the case of control-flow graphs, a typical coverage criterion is branch coverage. For mutation testing, a typical coverage criterion is, after mutants are generated based on predefined operators, their detection by the test suite. It is important to clearly specify the coverage criteria as it is often used as an indirect way to measure the effectiveness of ET techniques regarding test case generation.

3. Automated test step

A comprehensive testing strategy must address many activities that should ideally be automated: the generation of test requirements, test case generation, test oracle generation, test case selection, or test case prioritization. Since no ET technique will address all of them at once, it is important to clearly highlight in a study which part of the testing process is being automated and what are the expected inputs and outputs. In our current review, we are only dealing with test case generation. However, this term is used to mean many different things in the literature. Amman and Offutt [36] define a test case to be composed of test case values, expected results, prefix values, and postfix values necessary for a complete execution and evaluation of the software under test. In their definition, the test case values are the input data necessary to complete the execution, the prefix values are the inputs necessary to put the software into the appropriate state to receive the test case values and the postfix values are either the values necessary to see the results of the test case execution or the values needed to terminate the program or return it to a stable state. In the literature, a test case can include any one of these. In the review, we have accepted the broad definition of test case and so a test case may refer, for example, to a set of program inputs values, a sequence of method calls, a state-transition path, or a task/thread schedule.

4. Why is this test problem addressed with an evolutionary testing technique?

It is very important that the authors describe why a specific testing problem warrants the use of an ET technique. In order to assess the relevance of an empirical study involving the application of a MHS algorithm, the following information should be provided for the study:

- Sound evidence that current non-ET techniques do not perform well. This can be done by showing scenarios in which state-of-the-art techniques either fail or show poor performance.
- The authors should mention some specific characteristics of the problem that justify the usage of the proposed evolutionary technique. Examples include a large number of possible test cases (possibly infinite), complex or multiple fitness/optimization objectives. Certain evolutionary techniques have some well known characteristics or limitations. For instance, multi-objective approaches have scalability concerns or simulated annealing techniques may get stuck in local optima. In these cases, the authors should mention why these techniques are expected to still be appropriate for the problem at hand and how they have dealt with the limitations.

B. Metaheuristic Search Algorithm Specification

MHS algorithms are general strategies that need to be adapted to the problem at hand. When reporting a study, this implies describing and justifying the customizations and parameter settings for each specific algorithm. This will be required for replicating the study and also for comparisons with other ET techniques and future studies. Each type of MHS algorithm has specific parameter settings to be reported, but the general idea is to report all settings and adjustments that may have an effect on the performance of the algorithm or are needed for replicating the study. In Fig. 2, as an example, we show how a typical genetic algorithm can be used for test case generation. We also present a description of the possible different customizations and parameter settings of genetic algorithms since this is the most commonly used algorithm for evolutionary testing (see SectionV.A.1.).

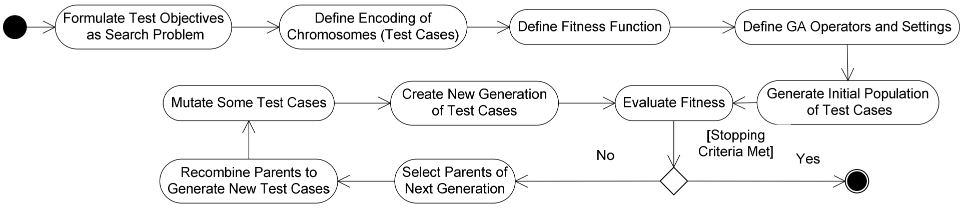
Formulate Test Objectives as Search Problem
Define Encoding of Chromosomes (Test Cases)
Define Fitness Function
Define GA Operators and Settings
Generate Initial Population of Test Cases
Evaluate Fitness
Create New Generation of Test Cases
Mutate Some Test Cases
Recombine Parents to Generate New Test Cases
Select Parents of Next Generation
No
[Stopping Criteria Met]
Yes

Fig. 2. : Test case generation using genetic algorithms

1. *Value encoding and chromosome definition*

Re–expressing the testing problem as a search problem includes the definition of a mapping from the solution space into the search space. This involves defining an encoding for the genes and the chromosomes. The genes are a constituent part of chromosomes. The chromosome encoding is dependent on the kind of problem being solved and for test case generation it can be, for example, the data for a specific test case. Other examples of encodings are binary, gray code, and decimal, as described in [37]. Attention must be paid to the kind of encoding being used because the subsequent operators (mutation and crossover) will also be based on the chosen encoding. This representation, which leads the ET technique to the best solution, is an important part of the paper and shows the ability of the ET technique to tackle the problem in an improved way. Therefore, it is suggested that authors precisely explain and justify the:

- Construction of genes and chromosomes,
- Mapping from solutions to chromosomes, and
- Constraints on the chromosomes, if any

2. *Fitness (cost) function and search problem formulation*

The fitness function is probably the main part of any genetic algorithm because it is responsible for guiding the genetic algorithm in its search for an optimal solution. An efficient fitness function, will select fit individuals for reproduction that will in turn result in achieving the search objective faster. How the fitness function is derived from the search problem, its assumption and limitations, and how it guides the search algorithm in reaching its objective should be clearly explained, in order to help assessing its suitability in other contexts.

3. *Initial population*

The authors should explicitly mention whether they use a random initial population or a specific strategy to select the initial population because the selection of the initial population has a direct impact on the performance of the algorithm. In any case, if a selection strategy other than random is used then it should be justified because it may increase the complexity of the technique and its cost, even though it is expected to yield benefits as well in terms of speeding up the search. One simple strategy for justifying a specific selection strategy for the initial population may be to just compare its results with those of an initial population selected at random.

4. *Selection strategy*

Since several strategies can be used for selecting parents for recombination, the authors should mention the employed selection strategy. Typical examples include: random pairing [38], roulette wheel [39], or tournament selection [39]. The justification of using a particular selection strategy should also be provided.

5. *Recombination*

The authors should describe the GA operators used for recombination along with the reasons for selecting particular operators. Especially, if a new operator is used, the reason for choosing it or a comparison between the results with simple operators and the new ones should be reported. Such an explanation is useful because the recombination operators are the means by which a search algorithm exhaustively explores the search space without getting stuck in local minima or maxima. The operators also have to be chosen carefully so that invalid chromosomes are impossible to generate or at least a rare occurrence [40]. The most commonly used recombination operators are crossover and mutation operators. However, there also exist some advanced recombination operators such as inversion and other reordering operators specific to certain situations [41]. Information about the following two commonly used operators should be mentioned in the paper:

a. Crossover

Different types of crossover operators are defined based on the number, the locus, and the probability (or rate) of choosing the chromosome for crossover. The authors should precisely define the crossover operator(s) used. The crossover operator along with the probability of crossover helps a genetic algorithm to move towards an optimal solution. Increasing the crossover probability too much does not help because this may result in losing some good individuals in a generation. It is therefore necessary to chose appropriate crossover operators with appropriate probabilities that help a genetic algorithm to efficiently find the solution. The commonly used crossover operators include single point and two-point crossover operators [29].

b. Mutation

The authors should mention mutation operators along with their probabilities (or rates). The mutation operator helps a genetic algorithm to avoid getting stuck in a local optimal solution by sampling new random points in the search space. If the mutation probability is increased too much, the genetic algorithm starts working like random search. It is therefore important to choose appropriate mutation operators with probabilities that aid a genetic algorithm in efficiently exploring the search space to find the global optimal solution. The most commonly used mutation operator is bit-flip operator [29].

c. Reinsertion strategy

Two points should be mentioned in this regard: How many individuals are retained from the previous population (i.e. not replaced by offspring) and what is the selection strategy for deciding which individuals should be retained. Sometimes this is referred to as a replacement strategy because then we talk about how many individuals are replaced (generation gap) and how do we select which individuals to replace with the new offspring. The reason for mentioning this is that the reinsertion strategy can lead to a fast and global search by balancing the exploration of the search space and the exploitation of discoveries made within the space. Some of the common reinsertion strategies are delete-all, steady-state and steady-state-no-duplicates [29].

6. Elitism

Elitism is a mechanism to ensure that the traits of the fittest individuals are transferred to the next generation. This is achieved by selecting some of the fittest individuals and keeping them as part of the next generation. These chromosomes are not affected by mutation and crossover. In many cases, the employment of elitism can produce better search results. The authors should mention whether elitism is used with genetic algorithm.

7. Stopping criteria

Selecting the stopping criteria for technique genetic algorithm is one of the challenging issues. Typical stopping criteria are: a maximum number of generations, a limit on the number of generations that do not show improvement in fitness values. On the one hand, one does not want to stop the evolutionary algorithm too early before it has found a solution. On the other hand, it is a waste of resources to continue if there is no possibility of finding a (better) solution. Hence the stopping criterion plays an important role in an ET technique that uses genetic algorithms and should be carefully selected.

C. Empirical Study Design

This section will define the most important items that should be reported about the study definition (through its objectives and hypotheses), design, and results.

1. Objectives and experimental hypotheses

One must define what is going to be empirically assessed and compared. The objective is usually to compare various ET techniques and alternatives in terms of code coverage, fault detection, test suite size, or test case generation time. The empirical study can be an assessment of a single ET technique, a comparison of two or more ET techniques, or a comparison of ET techniques versus non-ET technique. In any case, what is going to be compared should be precisely specified through formal hypotheses, thus leading to appropriate statistical significance testing. One notion important here is to state the kind of hypothesis that will be used: either a one-tailed hypothesis or a two-tailed hypothesis [42]. This has an impact on how we interpret the results in terms of p-values (probability of type I errors). In the context of ET, a one-tailed hypothesis would be used in the case when we are sure of the direction of the expected outcome. For example, when comparing a guided search algorithm such as genetic algorithm with random search, we expect the genetic algorithm to perform better or the same – but not worse – and as such we would use a one-tailed hypothesis. However, as an example, when comparing two genetic algorithms with different fitness functions, where we cannot state upfront which one would fare better in terms of cost or effectiveness, we would use a two-tailed hypothesis. In other words, when the theory regarding the search algorithms under study allows us to be a priori confident regarding the direction of differences in cost or effectiveness, then we should use a one-tailed test as this will increase our chances to uncover a statistically significant difference.

2. Target application domain

Empirical studies should specify a target application domain in which their results aim to be generalizable. Example application domains are: real-time, concurrent, distributed, embedded, and safety-critical systems. Testing techniques typically target specific faults that are more relevant in certain application domains, e.g., slow response time in real-time systems. Moreover, assumptions are typically made regarding the availability of information required to build the test model. Such assumptions tend to be more or less realistic depending on the application domain. For example, if one assumes the use of the MARTE UML profile [43] to design a system and then derive a test model, this is of course more realistic in the context of critical, real-time applications. Further, the selection of subject systems for empirical studies will then be determined by the target application domain.

3. Subject systems (Software Under Test or SUT) specification

After identifying the target application domain, SUTs are selected. It is important to justify why the selected SUTs are adequate matches for the target application domain as this will help the reader determine the extent to which the experimental results will generalize to this domain. For each SUT in the empirical study, the function of the SUT together with the domain that it is targeted should be described. In order to ensure repeatability of the experiment and help future comparisons of cost-effectiveness results, the hardware platform that the SUT executes on, the SUT size and the resulting size of the search space should also be defined.

4. Measures of cost and effectiveness for ET techniques

Measuring effectiveness and more particularly cost in our context is inherently difficult and the validity of measures is very often context-dependent. It is nevertheless crucial, in order to draw useful conclusions from studies involving ET techniques. The measures should be relevant for the particular study and comparable across the different techniques being investigated. Studies may use slight variations of an existing measure or introduce new ones, hence, it is important to explain the reasoning behind the measures and justify why they are applicable in the context they are being used. Along with the measure, the method used to collect the data related to the measures should be thoroughly explained.

In the context of ET, the effectiveness of a test case generation technique is closely related to the "quality" of the test suite generated by the technique. A good test suite can be characterized by its ability to uncover faults or to give confidence in the SUT by fulfilling a certain coverage criterion. Thus we can say that, in practice, there are two main categories of measures of effectiveness and they can be classified as coverage-based measures and fault-based measures. In the category of coverage-based measures, there may be many different types of measures depending on the adequacy criteria being used, for example, control-flow coverage criteria like branch or path coverage may be used. The fault-based measures are typically fault detection scores. They can be computed based on real, known faults or be estimated through mutation analysis. A program is seeded with faults based on mutation operators and techniques are assessed upon whether they are successful in detecting these faults. Depending on the number of faults caught, a so-called mutation score is calculated.

Cost measures are generally related to the speed of the technique to converge towards the test objective (in some cases it is referred to as the search technique's "efficiency"). Some common cost measures used in the ET domain are: (a) the number of iterations, which shows how many times an ET technique needed to iterate in order to find its best solution, e.g., the number of generations in genetic algorithms, or cycles in ant colony optimization algorithms, (b) the cumulative number of individuals in all iterations (usually each individual represents a test case in ET), (c) the number of fitness evaluations an algorithm needs to find the final solution, which depends on the number of newly generated individuals (usually each new population is made up of some individuals from the previous iteration and some newly generated ones), (d) the time spent by a MHS algorithm to find test cases meeting the targeted test objective which is sometimes referred to as “test case generation time”. This time can be either measured using clock time or CPU cycles. Clock time is the time from the “wall” clock and not easily comparable across different hardware architectures. However it is a practical measure that can be used to assess whether a technique can be used in practice. CPU cycles on the other hand is a measure that can be used across techniques for comparison on other hardware architectures as well, and (e) the size of the resulting test suite, which is a surrogate measure for the cost of executing the resulting test suite since a larger test suite would require more resources (e.g., time, CPU) to execute. Fig. 3 shows a simple example with four generations, 10 individuals in each population and in each new generation, four best individuals are selected from the previous generation and six new individuals are produced. The white circles inside each generation represent newly created individuals, whereas black circles represent the individuals that are taken from the previous generation. Based on this simple specification and definitions of these cost measures, we can calculate the values for the first three cost measures as follows. The first measure, the number of iterations, is easy to calculate because it is simply equal to the number of generations, which is four in this example. The number of individuals is the total number of individuals across all generations, which is equal to 40. The total number of fitness evaluations is calculated based on newly created individuals in each generation. In the first generation, the fitness of all 10 individuals is evaluated, in the second generation the fitness of only six individuals. Across all four generations, the number of fitness evaluations will be 28 in this case.

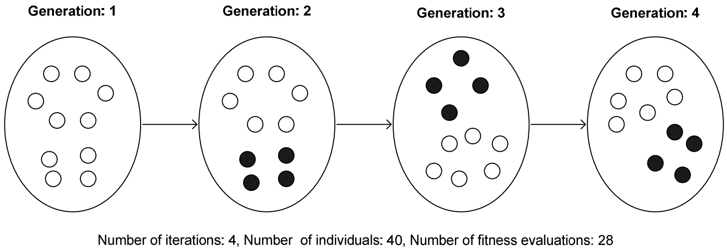
Generation: 1
Generation: 2
Generation: 3
Generation: 4
Number of iterations: 4, Number of individuals: 40, Number of fitness evaluations: 28

Fig. 3. : Example of different cost measures

Among the first three cost measures, the number of iterations is a very coarse grained measure and is not as precise as the number of individuals, which in turn is not as precise as the number of fitness evaluations. The number of fitness evaluations is more precise than the number of individuals because, in each iteration there are some individuals which are kept from the previous population and there is no cost for generating them. Therefore, the number of fitness evaluations can more precisely estimate the real cost of an ET technique. All these three measures are surrogate measures for the time used to generate the final test suite but none is perfect, because different search techniques may require a different amount of time per iteration, per creation of an individual (test case), or per fitness evaluation. For instance, it would not be a good idea to compare SA and GA based on the number of iterations because the amount of time required for each iteration in GA and SA is likely to differ significantly.

The cost of a technique is generally measured for one of two purposes: either to compare two techniques to assess which one will cost less for the same effectiveness or to assess whether a technique can be used in practice. From the measures discussed above, “test case generation time”, if it has been measured under similar conditions, is the only measure that can give users an intuitive idea of whether they can apply a particular technique to their situation within the time constraints that they face in practice. When comparing the cost of different techniques, it is also necessary to make sure that any other required resources are kept equal amongst the techniques. The fact that two techniques require the same amount of time does not mean that they have the same cost if one technique consumes much more memory than the other, for example.

5. Measures for scalability assessment

Scalability assessment is the process of assessing how the cost-effectiveness of an ET technique evolves as a function of the size of the test case generation problem to be addressed. This involves one or more measures of SUT size and the analysis of their relationships with the cost or effectiveness of the ET techniques under investigation. Some examples of measures that can be scaled up include the size of the SUT in terms of lines of code, size of search space in terms of number and range of input data parameters. The effect of this scaling is then observed on different cost and effectiveness measures to see if the ET technique is still cost-effective as the SUT gets larger and more complex.

6. Baselines for comparison

An ET technique can only be assessed if it is compared with a meaningful baseline. It is difficult to assess ET techniques in absolute terms and hence it is important to show, as a minimum, that the problem at hand could not be addressed by some simpler means. In other words, every study should have one or more baselines of comparison when assessing ET techniques and the minimum to be expected is a comparison with random search. The SUT investigated, may for example, be small and simple, and the fact that an ET technique performs well may not mean much. Random search can then serve as a basic verification that the search problem cannot be addressed by a simple random search and warrants the use of an ET technique. It is also preferable to use other simple techniques, such as hill climbing, as a comparison baseline for other more expensive ET techniques. This further demonstrates that the use of an ET technique is justified given the test case generation problem at hand. In addition—but this is context dependent—other ET techniques, previously published or considered plausible alternatives, can also be used as baselines of comparisons for the proposed ET techniques.

7. *Parameter settings*

Most ET techniques require parameter settings which tend to have a significant impact on their performance. In many studies, alternative parameter settings are investigated and compared. It is in any case highly important, to make any study replicable, to specify these parameters in a precise manner and justify them based on existing studies, when possible.

8. *Accounting for random variation in ET results*

Since ET techniques use MHS algorithms; their results can vary from one execution to another. So, it is important to ensure that we run the algorithms a sufficient number of times to capture the random variation of results and be able to perform statistical comparisons with other search techniques. It is difficult to precisely specify the number of runs required in general but, as a ballpark number, it should probably be above ten, so as to allow the use of basic statistical hypothesis testing and obtain a reasonable statistical power to detect differences [32]. Based on the expected (minimum) difference between techniques (if this can be estimated) and the statistical tests used to compare cost and effectiveness across techniques, the minimum required number of runs can be estimated using power analysis [44].

9. *Data Analysis*

During the design of an empirical study, it is important to decide about the data analysis methods that will be applied to cost-effectiveness and scalability results.

1. Data analysis methods for comparing cost-effectiveness

Performance in the case of ET usually relates to measuring the cost-effectiveness of the various search techniques. The cost and effectiveness of an ET technique are used together for assessing its performance. For example, a technique that has higher coverage than another technique may not be considered to have better performance, because it uses significantly more fitness evaluations (higher cost) to achieve that effectiveness, thus making it impractical for larger SUTs. Any claims of better performance should be backed by empirical evidence demonstrating lower cost or higher effectiveness when compared to the baseline and alternative techniques. In the ideal case, a study that is concentrating on measuring cost, should keep the effectiveness measures constant. For example, the study may measure the number of fitness evaluations needed to achieve 100% branch coverage. If, however, the aim is to measure effectiveness, then this can be done by keeping the cost constant, for example, by measuring how much branch coverage is achieved in some constant amount of time or the number of fitness evaluations. The reported performance results should include the results of the comparison baselines. At a high level, reported results should follow the structure below:

a. Reporting descriptive statistics

Both cost and effectiveness distributions should be reported (e.g., as a table with descriptive statistics) and analyzed. Looking at their standard deviation may indicate the level of uncertainty in terms of cost and effectiveness associated with an ET technique. This in turn may help determine how many runs would in practice be necessary to guarantee we obtain a satisfactory result.

b. Results of hypothesis testing

The purpose of statistical testing is to determine whether differences across ET techniques in terms of central tendencies for cost and effectiveness can be attributed to chance or whether they really capture a trend. Statistical hypothesis testing is necessary as ET techniques are always associated with a certain level of random variation in terms of cost or effectiveness. Because statistical testing is a standard practice, we will not detail it further here and interested readers may consult reference [45] for more details.

Statistical hypothesis testing should be used to accept/reject research hypotheses related to the cost-effectiveness analysis of ET techniques and comparison baselines. The choice of a specific statistical test depends on the specific objective of ET. In our context, hypothesis testing falls into three broad categories: (1) Comparing samples of runs in terms of effectiveness and cost. For example, comparing average or maximum branch coverage achieved across runs of alternative ET techniques and baselines of comparison. (2) Comparing samples of runs in terms of "successful" runs. For example, comparing the proportion of runs that find a deadlock across alternative ET techniques and baselines of comparison. (3) Comparing samples of targets (e.g., control flow branches) in terms of cost (e.g., iterations) or effectiveness (e.g., percentage of runs reaching that branch). In this last case, the samples are not independent, as observations in each sample are paired (identical targets). This leads to the application of specific statistical tests for paired samples. Moreover, though this is a standard issue, there can be two or more samples, and this will also affect the specific statistical test to be used. Moreover, as usual in other contexts, specific statistical tests have to be selected and justified based on the data distributions of the samples being compared to avoid incorrect conclusions to be drawn from the analysis. Statistical tests are usually classified as parametric and non-parametric [32]. When the sample follows a specific distribution (e.g., normal), certain parametric tests are applicable (e.g., *t*-test). Alternatively, non-parametric statistical tests are used when no appropriate assumptions can be made about the sample distributions. The issues related to selecting appropriate tests are however discussed in standard textbooks and will not be further addressed here. In Table I, as a guideline, we provide a mapping between the analysis situations we have encountered in ET studies and the type of statistical tests that are suitable (for the sake of simplicity, we are assuming two samples, that is, the comparison of two techniques,). This mapping is illustrated with examples.

TABLE I
MAPPING OF ET PROBLEMS TO STATISTICAL TESTS

ET Analysis Type	Type of Statistical Comparison	Example in the Context of ET	Type of Statistical Test (assuming two samples)
Comparing samples of runs in terms of effectiveness and cost	Comparing central tendencies of two or more independent samples, each corresponding to an ET technique	Comparing maximum branch coverage achieved across all runs between two ET techniques	Parametric *t*-tests or Non-Parametric Mann-Whitney U test
Comparing samples of runs in terms of "successful" runs	Comparing proportions in independent samples, each corresponding to an ET technique	Comparing the proportion of runs finding deadlocks across different ET techniques	z-score test for proportions
Comparing samples of target in terms cost to reach them or frequency at which runs reach them	Comparing central tendencies of matched pairs across samples	Comparing the frequency, across samples of runs matching each ET technique, according to which a branch (target) is covered. Note that the observations across samples are paired as they correspond to identical branches.	Parametric Paired *t*-tests or Non-Parametric Wilcoxon or Sign test

Data analysis should both address statistical and practical significance of differences among alternative search techniques. The former assesses whether differences among search techniques can be due to chance. The latter assesses whether the difference can be considered of practical significance, that is, whether they would make any difference in the day-to-day practice of test case generation given the specific test objectives being considered. For example, if statistical testing based on a large number of runs show that there is a significant difference between the cost of two search techniques in terms of time required for finding the best test suite, the actual difference may not be of practical importance if it is in the range of a few minutes. On the other hand, a lack of statistical significance despite a visible difference may be due to small samples, and therefore a lack of statistical power, which in our context means the number of runs for each compared search technique may be too small. The larger the number of runs, the more likely one is to obtain statistical significance when observing differences.

2. Data analysis methods for scalability

Scalability is used to assess whether an ET technique can be applied to either larger or more complex SUTs and still have satisfactory effectiveness and cost. If the aim of the empirical study is to show the scalability of an ET technique then appropriate measures of size and complexity should be clearly defined. There will be at least two measures involved – one size measure that will be scaled up through successive SUTs and the other that will measure the corresponding performance (cost and effectiveness). Then the effect of scaling up a particular measure can be reported in terms of a statistical relationship (recall the unavoidable random variation). For example, we may investigate several SUTs of variable sizes in terms of lines of code and then assess whether an ET technique can still reach a certain level of coverage at acceptable cost (e.g., measured as the number of generations) for larger SUTs and analyze how this cost evolves with the size of the SUT. A positive, exponential relationship between size and cost might then be problematic, for example, as it would undermine the applicability of the technique for large scale test models and systems. Similarly, if effectiveness (e.g., in terms of achieved coverage) is strongly decreasing as a function of SUT size, we also have a scalability problem.

As for scalability analysis, we need to characterize relationships between SUT size variables and measures of the ET technique’s cost and effectiveness. Such techniques are typically analyzed through regression analysis, though in practice, because the number of SUTs under study is likely to be small, such analysis is more likely to be qualitative, that is simply based on observing scatter plots in the cost-effectiveness and size space.

10. Discussion on validity threats

Validity threats should be considered throughout any empirical study, right from the study definition and design up to the analysis and interpretation of results [32]. The following types of threats can be discussed:

a. Construct validity threats

Measures of cost, effectiveness, and SUT size should be appropriate and justified given the context and objectives of investigation. No measure is expected to be perfect as the above concepts are usually not readily measurable. But in practice, by using several, complementary measures of cost, effectiveness, and SUT size, one is in a position to compare the cost-effectiveness and scalability of alternative search techniques.

b. Internal validity threats

If an ET technique performs better than another one, whether regarding effectiveness or cost, can it be due to something other than the ET technique? This could possibly be due to the following: 1) poor parameter settings of one or more of the ET techniques, 2) the biased selection of SUTs that have certain characteristics that can favor a certain ET technique.

c. Conclusion validity threats

- Has random variation been properly accounted for? Since ET techniques use MHS algorithms, randomness in results (inherent to metaheuristic approaches) should be accounted for, as discussed above. Has it been done in such a way as to enable statistical comparisons? It implies that a sufficient number of independent runs be performed to obtain a sufficient number of observations.
- Was the right statistical test employed? Statistical test procedures should be carefully selected given the hypothesis method (e.g. one-tailed vs. two tailed hypothesis) and the data collected (distributions of cost and effectiveness). Otherwise, certain required properties of a particular statistical test could be inadvertently violated leading to

incorrect conclusions. For example, many statistical tests assume that data distributions be normal [32].

- Is there any practically significant difference? To answer this question, the magnitude of the differences must be reported– this is known as the effect size and determines the practical significance of the results.

d. External validity threats

This is a difficult issue, as whether results can be generalized depends on whether the SUTs under investigation are representative of the targeted application domain and whether the faults considered (if used to assess test effectiveness) are representative of real faults. Ideally, ET empirical studies should also be run on many different SUTs of the target type, but every research endeavor faces limitations in terms of time and resources. At the very least, the issue should be carefully discussed and a good case should be made as to why one should be able to trust that the observed results can be generalized.

IV. RESEARCH METHOD

In this section, we will explain our review protocol. We define the research questions that this review attempts to answer, along with how we selected papers for inclusion and the data that we extracted.

A. Research Questions

The most important stage of any systematic review is to precisely define the research questions. Once the research questions have been specified, the systematic review can then proceed with the search strategy to identify relevant studies and extract the data required to answer the questions [46]. In this paper, we are interested in investigating empirical studies in the domain of ET. To proceed with our investigation, we defined the following three research questions:

RQ1: What is the research space of evolutionary testing?

The objective of this question is to characterize the research that has been undertaken so far. Though the research space can be identified from different angles, because our systematic review is about evolutionary testing, basic features of software testing (such as test level, targeted faults, test model, type of test cases, and application domain) and the type of MHS algorithms seem relevant characteristics to define the research space. Therefore, the first research question can be divided into the following sub-questions:

RQ1.1: Which metaheuristic search algorithms have been used for test case generation?

RQ1.2: Which types of test cases have been generated?

RQ1.3: For which test levels and targeted faults has evolutionary testing been used more frequently?

RQ1.4: For which test strategies has evolutionary testing been used more frequently?

RQ1.5: For which application domains have empirical studies in evolutionary testing been conducted?

All the above questions describe various aspects of the research space and in order to fully answer these questions we will use specific classifications, for example the types of MHS algorithms, the types of test cases and so forth. In the interest of not overloading the reader with too many details in this section, these classifications have been presented together with the results in Section V.

RQ2: How were the empirical studies in evolutionary testing designed and reported?

A study that has been properly designed and reported (as discussed in Section III) is easy to assess and replicate. The following sub-questions aim at characterizing some of the most important aspects of the study design and how well studies are designed and reported:

RQ2.1: How well is the random variation inherent in evolutionary testing, accounted for in the design of empirical studies?

RQ2.2: What are the most common alternatives to which ET techniques are compared?

RQ2.3: What are the measures used for assessing cost and effectiveness of evolutionary testing?

RQ2.4: What are the main threats to the validity of empirical studies in the domain of evolutionary testing?

RQ2.5: What are the most frequently omitted aspects in the reporting of empirical studies in evolutionary testing?

RQ3: How convincing are the reported results regarding the cost, effectiveness, and scalability of evolutionary testing techniques?

This research question attempts to synthesize the actual results reported in the studies in order to assess how much empirical evidence we currently have. To answer this question, we address the following sub-questions:

RQ3.1: For which metaheuristic search algorithms, test levels, and fault types is there credible evidence for the study of cost-effectiveness?

RQ3.2: How convincing is the evidence of cost and effectiveness of evolutionary testing techniques, based on empirical studies that report credible results?

RQ3.3: Is there any evidence regarding the scalability of the metaheuristic search algorithms for test case generation?

B. Study selection strategy

This is the step of a systematic review that aims at ensuring the completeness of the selection of papers on which the review is based. Study selection involves two main steps: (1) selection of the source repositories and identification of the search keywords (2) inclusion or exclusion of studies based on certain inclusion and exclusion criteria.

1. Source selection and search keywords

The process of selecting papers is started by executing a search query on the source repositories, which provides a set of papers. Since this set of papers is then subsequently used for all manual inclusions and exclusions, the selection of appropriate repositories and search strings is of utmost importance as it directly affects the completeness of the systematic review. The repositories that we used are: *IEEE Xplore, The ACM Digital Library , Science Direct (including Elsevier Science), Wiley Interscience* , and *Springer*. The first two repositories covered almost all important conferences, workshops, and journal papers, which are published either by IEEE or ACM. The next three repositories were mostly used for finding papers that are published in leading software engineering journals. We selected the following journals based on [46]: *IEEE Transactions on Software Engineering (TSE), ACM Transactions on Software Engineering and Methodologies (TOSEM), IEEE Software (SW), Springer: Software Testing Verification and Reliability (STVR), Springer: Empirical Software Engineering, Elsevier Science: Information and Software Technology (IST),* and *Elsevier Science: Journal of Systems and Software (JSS).* Since our review is about ET, we also included journals relating to software quality assurance and evolutionary computing. The journals that we included for this aim were: *Springer: Software Quality Journal* and *Springer: Genetic Programming and Evolvable Machines.* Another important source of publications that we included was the *Genetic and Evolutionary*

Computation Conference (GECCO) conference. GECCO is the largest conference in the field of genetic and evolutionary computation. GECCO's proceedings were published by Springer in 2003 and 2004 and afterwards by ACM.

A systematic way of formulating the search string includes (1) identifying the major search keywords based on the research questions (2) finding alternative words and synonyms for the major keywords and (3) creating a search string by joining major keywords with Boolean AND operators, and the alternative words and synonyms with Boolean OR operators.

Based on our main research focus, which is investigating empirical studies in the domain of ET, the following major search keywords are used in this paper: *software testing* and *metaheuristic search algorithm*.

We did not use *empirical study* as a keyword because we realized that not all papers that perform an empirical study, in the broad sense that we have defined it, use this keyword.

To formulate our search query we tried a number of search strings and came to the conclusion that *'software testing'* as an expression is not a good keyword because there are many papers which don't use these two words together but are nevertheless related to software testing. These papers may use terms such as testing, test case, test data and so on. On the other hand if we used the term testing alone, we would find too many unrelated papers. So we decided to use the terms *software* and *test* linked together with a Boolean AND instead of using *'software testing'* as an expression. Using 'software' and 'test' will find almost all related papers to software testing, but to make sure that we do not miss any interesting papers in test case generation we used the expression of 'test case generation' as an alternative for software testing.

Metaheuristic search algorithm is the second major term and also has many alternatives. We used general terms like *'evolutionary algorithm'*, *'meta-heuristic'*, and *'search based'* to explore the domain. Also, names of different MHS algorithms were used to make sure that no related papers were missed.

We also wanted to make sure not to miss any papers that have explicitly used the widely used term ‘*evolutionary testing*’, and thus included the expression of ‘*evolutionary testing*’ as a separate search string joined with the main string by an OR Boolean operator. The above decisions lead to the following search string shown in Fig. 4:

The whole string is searched in each repository in all titles, keywords, and abstracts. The expression ‘*evolutionary testing*’ is searched in the entire contents of all papers in the repositories as well.

```
{(((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>
```

Fig. 4. : The search string used for selecting the papers from repositories

One problem that we realized after some manual checking of the results of the search query was the fact that some search engines, such as IEEE Xplore, differentiate between the singular and plural form of words. To deal with this, we had to add some more alternative words and expressions to the search string by adding an ‘*s*’ to the end of all the words we already had. For example, we added ‘evolutionary algorithms’, ‘meta-heuristics’, ‘genetic algorithms’ and so on.

After finalizing the search string, the search query was run on the search engines of different repositories.

2. *Study selection based on inclusion and exclusion criteria*

Metaheuristic search algorithms have been used to automate a variety of software testing activities such as test case generation, test case selection, test case prioritization, and optimum allocation of testing resources. Since the focus of this systematic review is on test case generation, it is therefore necessary to define suitable inclusion and exclusion criteria for selecting relevant papers. In this section, we will discuss and justify the inclusion and exclusion criteria that were used.

We executed our search query on all selected databases and found 449 (after removing duplicates from different repositories) research papers in total. We only included papers up to the year 2007. In order to select the relevant papers to answer our research questions, we applied a two-stage selection process. At the first stage, we excluded papers based on abstracts and titles. All the papers were divided into three sets and each set was read by a researcher. We applied the following exclusion criteria:

- Abstracts or titles that do not discuss test case generation or any of the alternate terms that we used were excluded.
- Abstracts or titles that do not discuss the application of any MHS algorithm to automate test case generation were excluded.

If a researcher was unsure about a paper after reading its title and abstract, then the paper was included for the second phase of selection. After applying the inclusion criteria for the first phase, we were left with 121 papers.

At the second stage, we again divided the papers into three equal sets and divided among three researchers to check contents of the papers. We excluded papers based on the following exclusion criteria:

- Posters, extended abstracts, technical reports, PhD dissertations, and papers with less than three pages were excluded. Our goal was to account only for peer-reviewed, published papers that presented sufficient technical details.
- The papers that do not automate test case generation were excluded because this is the scope of our review.
- The papers that do not report any empirical study (see Section II.C for details on what we mean by empirical studies) were excluded.

In the cases where a researcher could not decide whether to keep or exclude a paper, then the paper was discussed with other researchers and a decision was made, by consensus. After applying the second phase of selection we remained with 67 papers that contained empirical studies about test case generation using MHS algorithms. However, three of these 67 papers, presented empirical studies that had already been reported in some other paper. This occurred, for example, when the journal version of a conference paper was found. In these cases we extracted data about the study from both the conference and journal versions of the paper and reported them as one study. Thus in the rest of the review we mention only 64 papers in total, even though we did analyze 67 papers. Details on the number of papers found in each database and number of papers included after applying inclusion and exclusion criteria are listed in Table II.

TABLE II
DISTRIBUTION OF PAPERS AFTER APPLYING INCLUSION AND EXCLUSION CRITERIA

Repository	Number of Included Papers After Applying Search Query	Number of Papers After Stage 1 Exclusion Criteria	Number of Papers After Stage 2 Exclusion Criteria
IEEE Xplore	297	77	33
ACM Digital Library	117	27	22
Wiley Interscience	8	2	2
Science direct	8	3	2
Springer	19	12	8
Total	449	121	67

3. *Data extraction*

We designed a data extraction form in Microsoft Excel to gather data from the research papers. We collected two sets of information from each paper. The first set included standard information [47] such as name of the paper, authors' names, a brief summary, researcher's name, and additional comments by the researcher. The second set included the information directly related to answering the research questions (see Table III for a summary list and Section III for details on each data item). To assess and improve consistency of data extraction among the researchers, a sample of papers were selected and read by all researchers and the relevant data extracted. The extracted data was then discussed by the researchers to ensure a common understanding of all data items being extracted and where necessary, the data collection procedure was refined. The final set of selected papers from each repository was then divided amongst three researchers. Each researcher read the allocated papers and extracted the data from the papers. In order to mitigate data collection errors, the data extraction forms of each researcher were read and discussed by two others. All ambiguities were clarified by discussion among the researchers.

TABLE III
RESEARCH QUESTIONS AND TYPE OF DATA COLLECTED

Research Questions		Type of Data Collected
RQ 1	RQ 1.1	Type of evolutionary algorithm
	RQ 1.2	Type of test cases
	RQ 1.3	Test level, targeted faults, test model source, and test model representation
	RQ 1.4	Application domain
RQ 2	RQ 2.1	Number of runs, analysis method
	RQ 2.2	Comparison baseline
	RQ 2.3	Measures of cost, measures of effectiveness
	RQ 2.4	Conclusion, external, internal, and construct validity threats
	RQ 2.5	All of the information from RQ2.1 to RQ2.4 is used, formal hypothesis, object selection strategy, data collection method
RQ 3	RQ 3.1	Test level, fault type, MHS algorithm
	RQ 3.2	Test purpose, Comparison baseline, cost and effectiveness results
	RQ 3.3	Scalability results

V. RESULTS

The following section outlines the results related to the research questions. Though no formal meta-analysis of the results of the empirical studies could be performed because of the variations in the way empirical studies are conducted and reported, results are compiled in structured, tabular forms to provide a structured overview.

A. RQ1: What is the research space of evolutionary testing?

The purpose of this research question is to identify and classify the overall research space of ET. To answer this, we further sub-divided the research question into five sub-questions. The main research question is answered based on the findings of each sub-question.

1. RQ1.1: Which metaheuristic search algorithms have been used for test case generation?

Evolutionary testing techniques employ metaheuristic search algorithms for test case generation. The more popular MHS algorithms found in the literature are: genetic algorithms, simulated annealing, particle swarm optimization, ant colony optimization, and genetic programming. These were the algorithms used to create an initial classification for all the MHS algorithms that have been used for test case generation. Whilst reading the papers, we came across many instances, whereby the standard MHS algorithms were tailored to achieve better performance, e.g., bacteriologic algorithm [48] [49] which is based on genetic algorithm. We put these papers into extensions of the standard MHS algorithms. The category ‘other’ was created for any algorithm that may have been introduced that was not an extension of the commonly used MHS algorithms. It is worth recalling from Section III.C that, for practical purposes, we do not label hill climbing as a MHS algorithm in this review.

Table IV shows the distribution of the various algorithms that we encountered for test case generation. Some papers had more than one empirical study in them and hence there are more uses of the MHS algorithms than the total number of papers. Within the 64 papers that we read, the genetic algorithm was used 46 times. This refers to the standard genetic algorithm, where well-known crossover and mutation operators are used. However, there were 16 additional empirical studies that attempted to extend the genetic algorithm to specific situations. Simulated annealing was used in seven cases and there were five cases where it was extended. The ant colony algorithm was used three times while there were three empirical studies that used genetic programming or an extension of it and two that used particle swarm optimization or its extensions. In some cases, there were attempts to use less known approaches for test case generation such as using tabu search, evolutionary strategies [50], or an algorithm called the great flood algorithm [51]. Such algorithms have been listed under 'Other'.

TABLE IV
DISTRIBUTION OF PAPERS ACCORDING TO METAHEURISTIC SEARCH ALGORITHM USED

Metaheuristic Search Algorithm	Number of Uses
Genetic Algorithms	46
Genetic Algorithm Extensions	16
Simulated Annealing	7
Simulated Annealing Extensions	5
Ant Colony Optimization	3
Genetic Programming and its extensions	3
Particle Swarm Optimization and its extensions	2
Other	3

The results show that the most commonly used algorithm is the genetic algorithm, followed by a more limited use of simulated annealing. The other algorithms are under-represented with three or less papers. There could be several reasons for this frequent use of genetic algorithms. First, there are numerous publications on the application of GA to various problems [52]. Furthermore, substantial empirical data is available for the different parameter settings required by GAs and this greatly helps the choice of appropriate parameters for a specific problem to be solved [53]. This together with the many books [38, 39] that exist on genetic algorithms makes it easier for researchers to learn how to adapt genetic algorithms to their context. Second, being a global search algorithm, GAs are proven to be better than local search algorithms [30], though there is no evidence showing that GA is better than other global search algorithm [52]. Last, GAs have many well known implementations in the form of commercial tools [54] and frameworks [55], which greatly facilitate their practical application.

2. *RQ1.2: Which types of test cases have been generated?*

As mentioned before, for this review, we did not use a very strict definition of the term ‘test case’; we included papers that generated a variety of different types of test cases. Hence a test case could be in the form of test data for some function of a system, it could be a sequence of method calls and their parameters, or even complete test drivers in some instances [48, 49, 56]. We have used the term test driver when a complete program that can be executed to test the SUT is generated. The term interaction sets is used to describe the situation where test cases are generated to cover all t-way interactions of different configuration parameters. We classified all the papers using combinatorial designs [13] for testing into the category of ‘interaction tests’. In some papers, a test case is a very specific combination of data and procedure, that is nevertheless used to exercise the functionality of the system. For example, in [16] a test case is a sequence of arrival times for aperiodic tasks – these types of test cases have been collected under the heading ‘other’ [16, 17, 40, 57].

From the 64 papers that we examined, we found that most deal with the generation of test inputs or data – 46 papers in total. This is perhaps due to the fact that test data generation is a common problem and one that is more easily amenable to automation. Also, initial research has focused on test data generation and it is natural for ET techniques to follow this trend. For the object-oriented paradigm, it is not enough to just generate test data for a method, and in this case

MHS algorithms have been used to generate sequences of method calls and their parameters or tackle other interesting aspects such as runtime exceptions [58] – in total we found seven such papers. Table V shows the different types of test cases generated using MHS algorithms.

From the results, we can conclude that test inputs or test data have been the most common type of test cases generated by MHS algorithms.

TABLE V
DISTRIBUTION OF PAPERS ACCORDING TO THE TYPE OF TEST CASE

Type of Test Case	Number of Uses
Test input/Test Data	46
Sequence of method calls and parameters	7
Interaction sets	5
Test drivers	3
Other	3
Total	64

3. *RQ1.3: For which test levels and targeted faults has evolutionary testing been used more frequently?*

The Encyclopedia of Software Engineering [35] describes four major levels at which testing is conducted. These are unit testing, integration testing, system testing, and acceptance testing. For this review, we are mainly concerned with the first three. Acceptance testing is used to prove that the system is ready for operational use (validation) rather than finding faults in it (verification). It is normally performed on the behalf of users to see if the system meets their requirements.

During our systematic review, we found that the majority of the papers have focused on unit testing - 48 papers. The term unit testing can refer to different types of units such as functions, components, or small programs consisting of a few functions. Any paper using one of these terms was classified under unit testing. There seems to be less attention on system and integration testing: system testing was tackled only by 11 papers and there were only five papers for integration testing. Table VI summarizes these results.

TABLE VI
DISTRIBUTION OF PAPERS USING DIFFERENT TEST LEVELS

Test Level	Number of Papers
Unit	48
Integration	5
System	11
Total	64

It is also useful to know which MHS algorithms have been used at which testing level. We found that for unit testing, genetic algorithms and its different extensions have been used the most. There were 38 papers that used genetic algorithms for test case generation at the unit testing level and 14 other papers that used extensions of genetic algorithms. The use of all other MHS algorithms for test case generation at the unit testing level is very low as shown in Table VII. An important point to note here is that since in each paper different MHS algorithms may have been used, the total numbers for all MHS algorithms at each test level differs from the total number of papers at each level as reported in Table VII. For integration testing, we didn’t see much variety in the use of different MHS algorithms. Simulated annealing and its extensions were used in three papers, whereas genetic algorithm was used in two papers. For system testing, it was once again genetic algorithm and its extensions that were used the most. Genetic algorithms and its extensions were used in eight papers, whereas simulated annealing and its extensions were used in three papers.

TABLE VII
DISTRIBUTION OF PAPERS FOR EACH EVOLUTIONARY ALGORITHM FOR EVERY TEST LEVEL

ET/Test Level	GA	Extended GA	SA	Extended SA	GP and Extensions	PSO and extensions	ACO and extensions	Other
Unit	38	14	3	3	3	1	1	1
Integration	2	0	2	1	0	0	0	2
System	6	2	2	1	0	1	2	0
Total	46	16	7	5	3	3	3	3

GA: Genetic Algorithm, SA: Simulated Annealing, GP: Genetic Programming, PSO: Particle Swarm Intelligence, ACO: Ant Colony Optimization.

At a high level, software requirements can be classified into two types: functional and non-functional requirements. Based on the type of requirements, we could also classify faults into either functional faults or non-functional faults as discussed in the framework. During the systematic review, we found nine papers specifically targeting functional faults, whereas five papers targeted non-functional faults. Different papers deal with different types of non-functional faults. Three papers focused on performance related faults [40, 59, 60], two papers on stress related faults [16, 59], and one paper specifically targeted faults related to robustness [61]. Apart from these papers, the remaining 50 papers didn’t specifically target any type of faults; rather they focused on achieving coverage of a test model. Using a test model is, however, often an indirect way of finding certain types of faults but is not explicit in doing so. Hence these papers were listed under the category ‘Not Discussed’. For example, if an ET technique uses a state machine as a test model, then the technique may be implicitly targeting state transition faults. Another example of such papers is an ET technique aiming to achieve branch coverage in the source code, it could be said that it is implicitly targeting control flow faults at the code level. Table VIII shows the summarized results.

TABLE VIII
DISTRIBUTION OF PAPERS ACCORDING TO FAULT TYPES

Fault Type	Number of Papers
Functional	9
Non-Functional	5
Not Discussed	50
Total	64

We can draw the following conclusions based on the above evidence:

- Most efforts for test case generation using MHS algorithms have focused on unit testing.
- Amongst all the testing levels, genetic algorithms and simulated annealing together with their extensions are used the most for ET.
- The majority of the papers do not target any specific faults or do not make this decision explicit.

4. RQ1.4: For which test strategies has evolutionary testing been used more frequently?

Section III.A reported that a test strategy is defined by the source used to create the test model, the representation of the test model, and the coverage criteria used to derive test cases from the test model. This is the information that we collected for each paper. A test model can be based on the artifacts that are created during the different phases of the software development life cycle. We chose to use a simple classification for model sources that turned out to be good enough when reading the papers: specification, design artifacts, and source code.

The classification used for the model representation is derived from the common test models representations used in the literature. For specification-based test models, we found Simulink models [22], input domain representation (see Section III.A.2. b.), and some problem specific representations, which are listed as ‘other’ in Fig. 5 [61, 62]; for test models based on design artifacts, we included state machines [63], markov usage models [57], hypergraphs [64], and other proprietary model representations [16, 17] and for the source-code based representations we found control flow graphs, data flow graphs, abstract syntax trees, lists of mutants (see Section III.A), and program dependence graphs. We define a test model as any software representation from which test cases are derived. Based on this definition, we also have model representations such as input domain representation and lists of mutants because the test cases are derived from these representations.

The final important aspect of the test strategy is the coverage criterion that is used to create test cases from the model. We used the classification defined in [65] for the coverage criterion. This includes control flow oriented, data flow oriented, fault-based, requirements-based, and N-wise coverage. Requirements-based coverage is based on covering a set of system requirements and is usually applied to test models whose source is specifications. In this case, traceability between requirements and the specification must be established. For example, a state machine specification can be annotated with requirements identifiers. N-wise coverage requires test cases to cover all possible combinations of N configurations, e.g., N=2 corresponds to pair-wise coverage [65]. In addition to this, we found papers that aim to find test inputs having the best case and worst case execution time. These papers have proposed test strategies that use test inputs as the test model representation. These papers cannot be classified into any of the above categories. In order to classify such papers, we created a new category of coverage criteria called execution time-based coverage.

During our systematic review, we found 44 papers that used source code based test models, 17 papers that used specification-based models and only four papers that used design artifacts. In the following sections, we report on the different test model representations that we encountered based on their source. We also report on the coverage criteria that were applied to different test models.

a. Source code

From our data we observed that control flow graphs are very commonly used as the model representation in the domain of ET, it was used in 36 papers. Naturally, control flow oriented criteria are used on control flow graphs. The most common coverage criteria that we observed were branch coverage (24 papers) and path coverage (seven papers). Branch coverage is one of the most widely used coverage criterion and is recommended by many standards as the minimum testing requirement such as by ANSI/IEEE 1008-1987 [66]. Other control flow-based criteria that we found were: loop coverage, condition coverage, and statement coverage. Other less observed test model representations are shown in Fig. 5 along with the coverage criteria applied to them.

b. Specifications

For specification-based models, we found that the input domain representation has been used the most as the model representation (14 papers). Other model representations that are used infrequently are shown in Fig. 5 along with their frequency of occurrence. In the 14 papers that used input domain representation as the test model representation, six papers applied execution-time based coverage, five papers applied n-wise coverage, two papers applied requirements-based coverage, and one paper applied data flow coverage.

c. Design artifacts

We observed only four papers that used design artifacts as the model source (see Fig. 5). Different control flow-oriented coverage criteria were applied on each of these test models such as arc coverage on markov usage models and all-states and all-transitions coverage on state machines.

Based on the above evidence, we can draw the following conclusions regarding the test strategies used in ET:

- The most commonly used source of test models is source code and the least frequently used is design artifacts.
- Control flow graphs are the most frequently used test model representation (36 papers) and input domain representation is the second most frequently used test model representation (14 papers).
- The most widely used coverage criterion across all test model sources is control flow oriented criterion. In total, 40 papers used control flow oriented coverage criteria on

different test model representations.

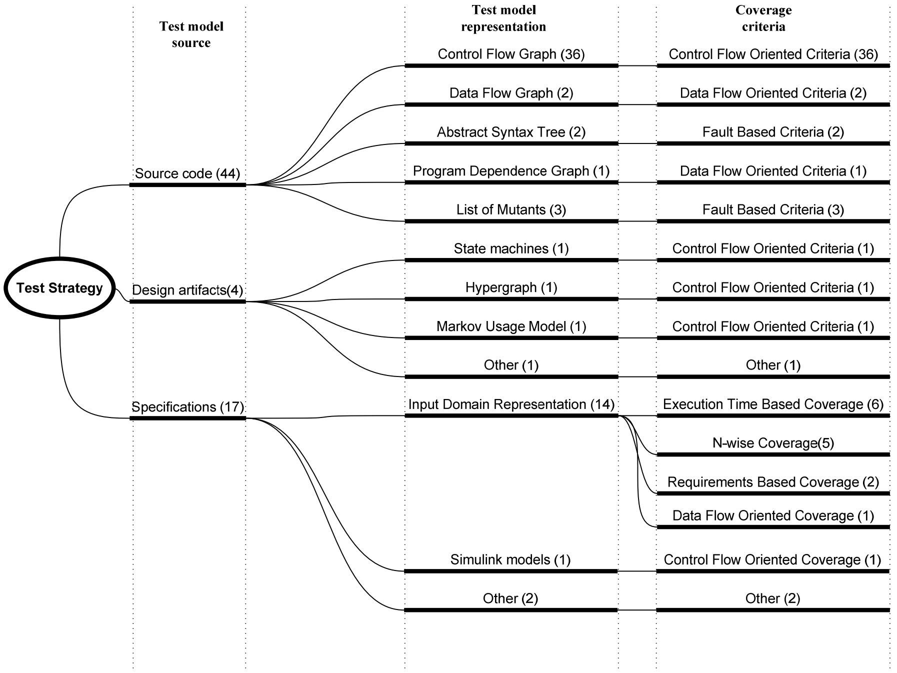
Test model source
Test model representation
Coverage criteria
Test Strategy
Source code (44)
Design artifacts(4)
Specifications (17)
Control Flow Graph (36)
Data Flow Graph (2)
Abstract Syntax Tree (2)
Program Dependence Graph (1)
List of Mutants (3)
State machines (1)
Hypergraph (1)
Markov Usage Model (1)
Other (1)
Input Domain Representation (14)
Simulink models (1)
Other (2)
Control Flow Oriented Criteria (36)
Data Flow Oriented Criteria (2)
Fault Based Criteria (2)
Data Flow Oriented Criteria (1)
Fault Based Criteria (3)
Control Flow Oriented Criteria (1)
Control Flow Oriented Criteria (1)
Control Flow Oriented Criteria (1)
Other (1)
Execution Time Based Coverage (6)
N-wise Coverage(5)
Requirements Based Coverage (2)
Data Flow Oriented Coverage (1)
Control Flow Oriented Coverage (1)
Other (2)

Fig. 5. : Classification of test strategies

5. *RQ1.5: For which application domains have empirical studies in evolutionary testing been conducted?*

Software is prevalent in almost all industries, and to completely address RQ1, it is also necessary to address the different application domains in which the ET research has concentrated on. We picked some initial domain characteristics to describe some of the domains with the intention that we would add on more characteristics as we encountered them in the various papers that we read. However as can be seen from Table IX, we found that most papers do not generally describe the domain that they are targeting. From the 64 papers that we read, for 45 of them we could not derive any data that would indicate which domain the test case generation was intended for. When we could derive this information, it seems that most of the papers dealt with embedded or real-time systems, with the occasional paper on safety-critical systems.

However, some of the 45 paper for which we could not distinguish the domain, dealt with general software development areas – they may have dealt with test case generation for object-oriented programs or test cases were generated for some form of control flow in procedural programs. In these cases it was just not possible to derive a specific targeted application domain since the information would be applicable to a large variety of domains. The domain characteristics are not mutually exclusive, a paper can target multiple domains, for example, a paper may generate test cases for an embedded, real-time system. Hence it is important to mention that three papers fell into both the embedded systems and safety critical categories and therefore the total number of uses (67) in Table IX is greater than the total number of papers that we included for this systematic review (64).

TABLE IX
DISTRIBUTION OF PAPERS ACCORDING TO APPLICATION DOMAIN

Application Domain	Number of Uses
Real-Time	8
Embedded Systems	10
Safety-Critical	3
Distributed Systems	1
Not Discussed	45

6. *Conclusion*

Based on the findings in each sub-research question, we can answer the main research question as follows:

In the context of evolutionary testing, most of empirical studies have used genetic algorithms, simulating annealing, and their various extensions to automate test case generation at the unit testing level. A few of the papers target specific application domains and most the frequently observed domains were real-time and embedded systems. Most of the papers defined test cases as test data (or test input) [67] and didn't target any specific faults. The papers that didn't target any specific faults, aimed to achieve a specific coverage of a test model, thus indirectly targeting certain types of faults. The most frequently observed test model was the control flow graph where a variety of control flow-based criteria were applied to generate test cases.

B. RQ2: How were the empirical studies in evolutionary testing designed and reported?

The purpose of this research question is to investigate and assess the design and reporting of empirical studies in the domain of evolutionary testing. To answer this question, we further divided this question into five sub-questions. By answering each sub-question individually, we will answer the main research question.

1. RQ2.1: How well is the random variation inherent in evolutionary testing, accounted for in the design of empirical studies?

We discussed the necessity and importance of accounting for random variation and using appropriate data analysis methods in Section III.C. To assess whether random variation has been accounted for, we classified the papers into two main categories: (1) papers which accounted for random variation in their design and reported this information and (2) papers which either did not account for random variation or did not report it well. To be classified in the first category, the study in the paper had to report the number of times the MHS algorithm was executed, sufficient information to determine whether the runs were independent, and report the data analysis methods used to compare alternative algorithms and baseline solutions. The independence of different runs can be determined in different ways in different MHS algorithms. For instance, in the case of hill climbing algorithm, if the algorithm is started from the same starting point in each run, then all the runs will not be independent and hence every time the algorithm will find the same solution. Different runs in hill climbing are normally made independent by choosing different starting points in each run. Additionally, the number of runs for each MHS algorithm had to be at least ten, a ballpark figure to enable the application of statistical hypothesis testing with minimal statistical power. Papers that did not report the number of runs or were executed less than ten times were placed in the second category (Random Variation Not Accounted).

Within the first category, we further divided the papers according to the type of data analysis that had been performed. If only the average of the results or the percentage of successful runs over all runs was reported, then these papers were classified as having "poor" descriptive statistics (the definition of successful run varies across papers, but generally speaking, if the test target to be covered is found, then the run is considered successful. A test target, for example, could be a branch to cover). This is because the average does not convey any information about the dispersion of the results being examined. Papers which report the level of variation as well as the measures of central tendency are counted in the sub-category "good" descriptive statistics. The final category is the set of papers that in addition to reporting "good" descriptive statistics also reported the results of statistical hypothesis tests comparing MHS algorithms and baselines and establishing the statistical significance of differences.

The results in Table X show that 25 papers did not account for random variation. Most of these, 20 papers, either did not provide any information about the number of runs or just reported the result of one unknown run (the best or the only run). In five papers, the study was repeated less than ten times.

TABLE X
RESULTS OF HOW RANDOM VARIATION IS ACCOUNTED FOR IN EMPIRICAL STUDIES

Random Variation Accounted			Random Variation Not Accounted	
Poor Descriptive Statistics	Good Descriptive Statistics	Statistical Data Analysis	Random variation not discussed or accounted for	Insufficient number of runs
24	8	7	20	5
38%	12%	11%	31%	8%

Amongst 39 papers which accounted for random variation, 24 papers reported only the average of the cost or effectiveness results across all runs, for example, the average number of killed mutants as an effectiveness result or the average number of iterations as a cost result. In some cases, the percentage of successful runs amongst all runs is reported instead of, or along with the average of the effectiveness results (e.g., average coverage or average mutation score). At least one measure of dispersion like standard deviation, variance, or the variation interval ([Min, Max]) was reported for eight papers. These papers are categorized as having “good” descriptive statistics. There were seven papers that reported statistical tests as well as good descriptive statistics. One or more of the following statistical tests were used: *t*-test, paired *t*-test, Mann-Whitney test, F-test, ANOVA, and Tukey test [45, 68]. There was one paper in this sub-category, which reported the use of statistical tests, but did not specify the specific test being used and did not provide any descriptive statistics.

From the results, we can see that 39% of the papers did not account for random variation at all, and 38% of the papers only had “poor” descriptive statistics, so in total 77% of papers either did not account for random variation or reported it poorly. The remaining 23% of papers are divided between 12% providing only good descriptive statistics and just 11% performing some kind of statistical hypothesis testing to assess the statistical significance of differences that is whether they can be due to chance. To answer RQ2.1, this review suggests that ET would greatly benefit from paying more attention to accounting for random variation in search heuristics and applying more rigor in analyzing and reporting cost and effectiveness results.

2. *RQ2.2: What are the most common alternatives to which ET techniques are compared?*

In assessing the cost-effectiveness of any technique, the comparison baseline is an important factor. To classify the papers based on comparison baselines, we defined two categories. The first category ‘single’ includes the papers, which compared one ET technique with exactly one alternative ET or non-ET technique (such as random search). The comparison to non-ET techniques serves a dual purpose: it helps determine if the problem at hand is simple enough to be satisfactorily solved by a simple search algorithm; otherwise it provides justification for why an ET technique is necessary. In addition, a baseline of comparison is necessary to assess the benefits of using ET techniques. The second category ‘multiple’ includes the papers that feature several comparison baselines, which can be either only non-ET techniques or both ET and non-ET techniques. It is worth remembering from Section II that, for the sake of simplifying discussions, test case generation techniques which are based on hill climbing are categorized as non-ET baselines.

From the 64 papers that we read, 35 papers had a single comparison baseline and 13 papers had multiple comparison baselines. Amongst the papers with a single baseline, 19 papers compared an ET technique to another, whereas 16 papers compared an ET technique to a non-ET technique. The non-ET technique was most often random search – we found 16 such papers. Some other examples of the non-ET baselines are testing techniques based on hill climbing, static analysis of the code, or some other techniques specific to the test problem such as schedulability analysis [16, 17].

For the papers that had multiple comparison baselines, there were five papers that compared an ET technique to more than one non-ET baseline, whereas eight papers had a mixture of both ET and non-ET baselines. As shown in Table XI, 16 studies did not discuss the comparison baseline at all. These studies did not include any kind of comparison; they usually introduced the use of a MHS algorithm for test case generation and performed an empirical study to show that the technique does indeed generate satisfactory test cases. These papers are missing the justification for why the ET technique was necessary to address the test case generation problem at hand and how much better it actually is compared to other existing, simpler techniques that are available to solve the problem at hand.

TABLE XI
COMPARISON BASELINES USED IN EVOLUTIONARY TESTING IN TERMS OF NUMBER OF PAPERS

Comparison Baselines				
Single		Multiple		Not Discussed
Evolutionary Technique	Non-Evolutionary Technique	Both Evolutionary and Non-Evolutionary Techniques	Non-Evolutionary Technique Only	
19	16	8	5	16

3. RQ2.3: What are the measures used for assessing cost and effectiveness of evolutionary testing?

Assessing the cost-effectiveness of ET techniques for test case generation is the main objective of empirical studies in our context. Therefore, measuring cost and effectiveness in a valid manner is a basic requirement for all empirical studies.

a. Effectiveness Measures

As it is discussed in Section III, effectiveness measures are categorized into two main classes: coverage-based and fault-based measures. Under the coverage-based category, we found three main sub-categories: (1) control flow based coverage criteria such as branch, statement, path, condition, and condition-decision coverage (2) data flow based coverage criteria such as all-DU coverage, and (3) N-wise coverage criteria, when ET techniques are used for testing combinatorial designs [13]. In the category of fault-based measures, mutation analysis is the core strategy and mutation score and the number of mutants killed are measures that were found in this review.

We found some other measures for effectiveness, which are still related to the quality of the generated test cases, but do not fit into any of the above categories. In this review, these measures are labeled “Others”. Based on the papers included in this review, we identified two sub-classes among them and labeled the rest as miscellaneous. Papers in the first sub-category use different kinds of measures related to the execution time of test cases and we called these time-based measures. The second sub-category addresses the distribution of fitness values of individuals (solutions) as the measure of effectiveness (e.g., average, maximum fitness). Such a measure is usually used when the goal of a search algorithm is not finding a targeted solution, but the goal is to be as close as possible to the targeted solution. An example of such papers is in [16, 17], where the goal was stressing the real-time systems by scheduling input sequences to maximize delays in the execution of targeted aperiodic tasks. In this study, the cost is measured by fitness values, which shows how close the completion time of a specific task is to its deadline. Table XII presents the number of papers in our review per the category of effectiveness measures.

TABLE XII
DISTRIBUTION OF EFFECTIVENESS MEASURES ACROSS EMPIRICAL STUDIES

Coverage-based measures			Fault -based measures	Others			No effectiveness measure
Control flow	Data flow	N-wise		Time-based measures	Fitness value of individuals	Miscellaneous	
43	2	2	11	6	5	3	3

The data we collected revealed 61 papers using one or more effectiveness measures in a total 72 different effectiveness measurements across reported studies. There were three papers that did not discuss the effectiveness of the ET technique at all. There were 47 instances (65%) that used some type of coverage criterion as the measure of effectiveness. The most often used criteria were control flow based criteria with 43 instances (60%). Among them, 23 instances (32%) used branch coverage, which is the most frequently used effectiveness measure. All-DU coverage, which is based on data flow analysis, was used in two instances and two instances used N-wise coverage as the coverage criterion.

There were 11 instances (15%) that used fault detection rate as the measure of effectiveness, where mutation analysis is used so as to report the mutation score or the number of killed mutants. In some cases, the fault-based measures are reported along with other effectiveness measures. Among the 14 instances (19%), which used the other measures for the quality of test cases, five papers used the fitness value of individuals and six papers used different kinds of execution time-based measures. Most of the time-based measures were related to CPU cycles spent for test case execution. They are used in studies which try to use ET techniques to generate test cases that will find the best/worst case execution time of a program.

Looking at the results in Table XIII, we can see that control flow based coverage criteria targeted at white-box testing are the most often used effectiveness measures and as we mentioned in the above discussion, branch coverage is the criterion that has received the most attention. As a result, this problem is now pretty well understood and there is a widely accepted, standard way of calculating fitness values based on approximation level and branch distance [27] on control flow graphs. Fault-based effectiveness measures got relatively little attention in the literature reporting ET studies as compared to coverage-based measures. Similarly, the applications of ET techniques to artifacts other than code are rare as white-box testing seems to have been by far the main focus.

b. Cost Measures

Based on the definition of cost measures in Section III and what we found in this review, we categorized cost measures into two main classes (1) ‘cost of finding the target’, which is related to the cost of automating test case generation and (2) ‘cost of executing the generated test suite’, which is related to the cost of test case execution. These are both relevant and complementary. Based on the measures found in the studies, the first category is classified into four sub-categories: (a) the number of iterations, (b) the cumulative number of individuals in all iterations, (c) the number of fitness evaluations an algorithm needs to find the final solution, and (d) test case generation time. The only measure for the category of ‘the cost of executing generated test suite’ that we found in the papers was the size of the test suite, which is a surrogate measure for test execution time.

TABLE XIII
DISTRIBUTION OF COST MEASURES ACROSS EMPIRICAL STUDIES

Cost of finding the target				Cost of executing the final test suite	No cost Measure
Number of iterations	Number of individuals	Number of fitness evaluations	Test case generation time	Size of test suite	
27	6	14	15	8	7

Table XIII shows that among 64 papers, seven papers did not perform any cost analysis and in the remaining 57 papers most empirical studies reported at least one cost measure in 70 different cost measurements reported across studies.

Based on the abovementioned classification, 62 instances (86%) used measures in the category “Cost of finding the target”. The most often used measure among them was the number of iterations, which is used in 27 instances (39%). A total of six instances (4%) used the number of individuals (test cases) and the number of fitness evaluations is used by 14 instances (20%) as the measure of cost. Finally, there were 15 instances (21%) that used the ‘test case generation time’ measure.

In the second main category, ‘cost of executing the final test suite’, the size of test suite was the only measure that we found and it was used in eight instances. Some of these instances, which report the number of test cases in the final solution, reported the cost of finding the target as well. In some of these instances, the target of the ET technique was actually creating test suites with minimum size for covering a specific criterion such as a minimal test suite that exhibits pair-wise coverage [69].

Summarizing the results of cost measures, we can see that the most commonly used measure is the number of iterations. This measure is, however, the least precise measure based on the discussion in the framework in Section III. Another conclusion is that most studies use cost measures only for comparison purposes with other alternative techniques. There are just 15 instances (21%) that used measures such as test case generation time, which conveys whether a particular technique is likely to be practical and scale up.

4. RQ2.4: What are the main threats to the validity of empirical studies in the domain of evolutionary testing?

In order to answer this question, we carefully assessed the studies using the proposed framework in Section III. For the construct validity threats, we looked at the validity of the cost and effectiveness measures. The most frequently observed threat was using some measures of cost that have severe limitations as they are not precise. As discussed in the framework, the imprecision of cost measures such as ‘the number of iterations’ makes the comparison between different ET techniques very coarse grained. In addition, measures such as the number of iterations, the number of individuals, and the number of fitness evaluations can only be used for comparison across ET techniques and cannot demonstrate the practicality of ET techniques. On the other hand, cost measures such as ‘test case generation time’, if measured as clock time, are suitable for showing the practicality of a technique under time constraints. Such measures are, however, platform dependent and therefore not easy to use for comparisons across techniques and studies.

The most frequently encountered conclusion validity threat is related to accounting for the random variation that exists in the results obtained from ET techniques. As discussed in RQ2.1, 39% of the papers did not take the random variation of results into account and 38% did not analyze or report it properly. This leads to a frequent threat regarding the statistical significance of the results. Therefore, not accounting for randomness and not applying proper data analysis (Section III.C and RQ 2.1) makes it very difficult to confidently draw practical conclusions from the results reported in most studies. Moreover, among the 11% of papers that discussed statistical hypothesis tests, just one paper has discussed the practical significance of differences that is whether differences among techniques justify the use of more complex techniques.

Regarding internal validity threats, the most important concern is the instrumentation of code and the use of different tools for data collection without reporting sufficient information about them. If the data collection and code instrumentation is not through a commercial, well-identified tool, then detailed information about the process of data collection should be reported. An example of this would be the use of a tool that instruments the code to collect, for instance, branch coverage information. If the tool is developed for experimentation purposes only and has not been thoroughly tested, then the coverage information generated by tool might not be reliable and hence leading to an internal validity threat. A possible way to deal with this validity threat is to use some commercial tool for this purpose.

The lack of clearly defining the target SUTs and having a clear object selection strategy are the most common threats to external validity. Usually the algorithms are executed on very small programs and no clear justification is provided for their choice and why they may be representative of the target domain, if specified. This can result in invalid generalization of the results.

5. RQ2.5: What are the most frequently omitted aspects in the reporting of empirical studies in evolutionary testing?

In the previous sections, we have discussed the lack of properly reported descriptive statistics and statistical hypothesis testing (statistical significance) as the most commonly missing aspects in many empirical studies. Only 23% of the reviewed papers reported proper descriptive statistics or statistical significance results. In addition to this aspect, as discussed in the framework, there are other aspects that are also important and should be reported. These aspects are: discussion of validity threats, specification of formal test hypotheses, object selection strategy, parameter settings, and data collection method. For validity threats, 10% discussed conclusion validity, 6% discussed external validity, 3% discussed construct validity, and only 3% of the papers discussed internal validity threats. We found that only two papers out of 64 specified formal hypotheses, 44% of the papers discussed about object selection strategies, and 39% of the papers described their data collection methods. Parameter settings (see Section III.C) were discussed by most, but not all of the papers (88%). However, all papers did not discuss all parameters required for their study; usually there is only a partial discussion. In some cases the authors provide justification of why they chose particular values for the parameters but this was rare.

TABLE XIV
THE MOST OMITTED ASPECTS OF EMPIRICAL STUDIES

The most omitted aspects in the reporting of empirical studies		Number of papers	Percentage
Good Descriptive statistics and statistical test		15	23%
Validity threats	Construct	2	3%
	Internal	2	3%
	Conclusion	7	10%
	External	4	6%
Formal Hypothesis		2	3%
Object selection strategy		28	44%
Data collection method		25	39%

Summarizing the above information, Table XIV depicts the most frequently omitted aspects in the reporting of empirical studies. Not reporting this information makes the full interpretation of the results very difficult. For example, poor reporting may make it difficult to determine whether differences are statistically significant, and whether differences are expected to matter in practice. It is also usually difficult to determine if results can be generalized and to what domain.

6. Conclusion

In our context, defining good and relevant cost and effectiveness measures is a prerequisite for a useful empirical study. Almost all of the papers use appropriate (though not perfect) cost and effectiveness measures to perform empirical studies. However, there were two major problems in the majority of the papers. First, most of the papers do not account for the random variation in cost and effectiveness of ET techniques. Even the majority of the papers that did account for the random variation didn't use proper data analysis and reporting methods (descriptive statistics and statistical hypothesis testing). Thus, there is a general lack of rigor in the statistical analysis and reporting of results in most empirical studies assessing the use of MHS algorithms for test case generation. Second, most of the papers didn't demonstrate the benefits of ET by comparing it with simpler, non-ET techniques such as random search or hill climbing. These two factors are highly important for yielding interpretable empirical studies in the context of test case generation using ET techniques. Furthermore, many other relevant aspects of empirical studies such as the reporting of validity threats, the definition of formal hypotheses, the object selection strategy, and data collection methods are not reported by most of the papers. We can therefore conclude that most empirical studies in the context of test case generation using ET techniques are still not properly conducted and reported and that improving this situation should be an important objective of the research community in future studies.

C. RQ3: How convincing are the reported results regarding the cost, effectiveness, and scalability of evolutionary testing techniques?

There is a lot of research being conducted on test case generation based on MHS algorithms. In order to draw general conclusions from the current body of work, we need to assess how convincing is the evidence regarding the cost, effectiveness, and scalability of ET techniques. The first step is to clearly identify studies that provide complete and credible evidence from an empirical standpoint. Credible results are the consequence of a well designed and conducted empirical study. Based on the discussions in section III, a well designed study in the context of MHS algorithms should account for the random variation present in the results and have a meaningful comparison baseline to show that the targeted test problem benefits from a MHS approach. Therefore, in order to answer this research question, we first selected papers that at a minimum account for the random variation of results and compare their technique with the results of a simpler, non-ET technique (such as random search, hill climbing, static source code analysis, or some other technique applicable to the test problem under consideration). The first sub question, RQ3.1, will provide an overview of these papers. The second step to answer RQ3 is to select those papers that performed and reported proper data analysis. To satisfy this criterion, we expect papers to report descriptive statistics on the variation in the results (cost, effectiveness), where relevant or results of statistical hypothesis testing comparing alternative test case generation algorithms, and in particular MHS algorithms with simpler baseline alternatives. We deemed this set of papers as having credible evidence regarding the cost, effectiveness, and scalability of ET. In sub question RQ3.2, we provide detailed information about the cost and effectiveness results presented in these papers along with a short description of the test problem that they tackled.

1. *RQ3.1: For which metaheuristic search algorithms, test levels, and fault types is there credible evidence for the study of cost-effectiveness?*

This sub-question provides a summary of the research papers that met the minimum criteria of accounting for random variation in results and performing comparisons with a simpler, non-ET baseline. Out of the 64 papers that we analyzed, we found 39 that accounted for random variation of results. This number was reduced to 18, after selection of only those papers that also had a non-ET comparison baseline. Thus, based on the criteria that we used, we had to exclude 46 papers as not being applicable for answering our research question. It is worth mentioning that there were 14 papers among those 46 discounted papers that had the minimum requirement of accounting for random variation, but they had another MHS algorithm as the comparison baseline. For example, they may have proposed an extension to a genetic algorithm that would possibly enhance its capacity for test case generation and compared their results to a genetic algorithm not having this extension. In this review, those studies are not considered as credible evidence, since they do not show, in any way, that a simple non-ET technique such as random search or hill climbing could not, in this particular context, equal or outperform their technique. This is an important consideration, since there is no a priori reason to believe that a MHS algorithm is more cost-effective and efficient than simpler algorithms in all test case generation contexts. The size of the search space is only a weak indicator of the extent of the search challenge as the search difficulty also depends on the search space landscape and distribution of satisfactory solutions across this space. Table XV summarizes this set of 18 papers in terms of the MHS algorithms used, the testing levels, and the fault types targeted in the empirical studies.

As can be seen in Table XV, amongst the 18 papers that report credible evidence, most papers (13 out of 18) applied an ET technique at the unit testing level. The most commonly investigated MHS algorithm is the genetic algorithm with 12 papers out of 18, followed by simulated annealing with just four papers. This trend is the same as that observed in the full set of 64 papers in Section V.A. There are also only two papers that target specific faults, one targeting functional faults and the other non-functional faults.

TABLE XV
TEST LEVELS, FAULT TYPES, AND THE TYPE OF METAHEURISTIC ALGORITHMS USED BY PAPERS HAVING CREDIBLE EMPIRICAL STUDIES

	Test Level			Fault Type		Type of Metaheuristic Search Algorithm						
Paper ID	Unit	Integration	System	Non-Functional	Functional	GA	EGA	SA	ESA	ACO	GP	PSO
Jones et. al. [1]	√	–	–	–	–	√	–	–	–	–	–	–
Puschnerand Nossal [4]	√	–	–	–	–	√	–	–	–	–	–	–
Tracey et. al. [6]	√	–	–	–	–	-	–	√	–	–	–	–
Bueno and Jino [8]	√	–	–	–	–	√	–	–	–	–	–	–
Michael et. al. [11]	√	–	–	–	–		√	–	–	–	–	–
Wegener et. al. [12]	√	–	–	–	–	√	–	–	–	–	–	–
Shiba et. al. [14]	–	–	√	–	–	√	–	–	–	√	–	–
Briand et. al. [16, 17]		–	√	√	–	√	–	–	–	–	–	–
Miller et. al. [19]	√	–	–	–	–	√	–	–	–	–	–	–
Watkins and. Hufnagel [20]	√	–	–	–	–	√	–		–	–	–	–
Zhan and Clark [22]	–	–	√	–	√	–	–	√	–	–	–	–
Zhan and Clark [23]	–	–	√	–	–	–	–	√	√	–	–	–
Bueno et. al. [24]	–	–	√	–	–	–	–	–	–	–	–	√
Harman et. al. [25]	√	–	–	–	–	–	√	–	–	–	–	–
Harman and McMinn [3]	√	–	–	–	–	√	–	–	–	–	–	–
Harman et. al. [26]	√	–	–	–	–	√	–	–	–	–	–	–
Wappler and Schieferdecker [28]	√	–	–	–	–	√	–	–	–	–	–	–
Xiao et. al. [30]	√	–	–	–	–	√	–	√	√	–	–	–

GA: Genetic Algorithm, EGA: Extended GA, SA: Simulated Annealing, ESA: Extended SA, GP: Genetic Programming, PSO: Particle Swarm Intelligence, ACO: Ant Colony Optimization.

2. *RQ3.2: How convincing is the evidence of cost and effectiveness of evolutionary testing techniques, based on empirical studies that report credible results?*

Along with accounting for random variation in the results and having a non-ET comparison baseline, studies must also report proper descriptive statistics or statistical hypothesis testing results in order to present credible and interpretable evidence. After the application of these criteria, there were just eight papers left and the results of these papers are summarized in Table XVI followed by more detailed information on each of these papers.

TABLE XVI
TEST PURPOSES, COMPARISON BASELINES, AND RESULT HIGHLIGHTS FOR THE PAPERS WITH CREDIBLE RESULTS

Article	Test purpose	Comparison baseline	Result highlights
Puschner and Nossal, 1998	Creating an input data set with the worst-case program execution time	RS BEDG StA	In most cases, GA performed equal to or better than RS in terms of effectiveness measured as execution time of the SUT. For smaller size SUTs, GA had results as good as BEDG and StA
Briand et. al., 2005 and 2006	Stressing a real-time system by creating input sequences that maximize delays in the execution of target tasks and increase chances of missing deadlines.	ScA	The technique can schedule tasks to miss the deadline(s) even though schedulability analysis identified them as schedulable. The GA is successful in bringing task completion times closer to their deadlines, thus leading to stressing the system with that respect.
Miller et. al., 2006	Test case generation using genetic algorithms and program dependence graphs.	RS, GA	1) The results showed that, for simple programs there is a little difference in the results (branch coverage) between RS and their proposed GA approach (TDGen). 2) The difference is seen in larger programs, where a much smaller number of generations are required to achieve 100% branch coverage. 3) It is also observed that for some SUTs, TDGen can achieve 100% branch coverage, where RS and GADGET cannot.
Watkins et. al., 2005	Comparison of different fitness functions for path coverage	RS	Based on the study, it was concluded that there is no single fitness function that works well in all cases. A two-step method using two best fitness functions is therefore suggested in the paper.
Harman and McMinn, 2007	Test data generation to answer three research questions formulated based on royal road theory (see [3]) for GA	RS, HC	1) GA was able to find inputs to exercise the branches that have royal road features and HC and RT were not successful at all. 2) GA was unable to find the inputs to exercise the branches that have royal road features if crossover operators were removed. 3) HC performed better or no worse than GA for the branches that do not have royal road features.
Harman et. al., 2007	Investigation of the relationship between the size of the search space (consisting of test inputs) and the performance of search algorithms measured as the number of fitness evaluations to cover a branch	RS, HC	1) There is no relationship between search space reduction and reduction in cost for random search. 2) There is significant improvement in cost reduction for both hill climbing and the genetic algorithm. 3) The reduction in cost is more for the genetic algorithm than for hill climbing. 4) There is no relationship between search space reduction and search effectiveness in terms of coverage for any of the search algorithms.
Wappler and Schieferdecker, 2007	An approach for testing non-public methods without breaking the encapsulation of the class using an objective function specifically designed to cover non-public methods via public methods.	RS, GP	The new GP technique achieved higher overall branch coverage than RS and higher coverage of non-public methods than their existing GP based approach.
Xiao et. Al. , 2007	Empirical evaluation of different MHS algorithms and RS for test data generation.	GA, SA, two extensions of SA (SA/AAN, GSA), RS	GA performed better than all other algorithms including random search. After GA, SA/AAN performed better in terms of both cost (condition decision coverage) and effectiveness (number of SUT executions).

HC: Hill Climbing, RS: Random Search, GA: Genetic Algorithm, SA: Simulated Annealing, GP: Genetic Programming, SA/AAN: SA with Advanced Adaptive Neighbors, GSA: Genetic SA, ScA: Schedulability Analysis, BEDG: Best Effort Data Generation, StA: Static Analysis

a. Testing the Results of Static Worst-Case Execution-Time Analysis (Puschner and Nossal, 1998)

Puschner and Nossal [4] used a genetic algorithm for creating an input data set with worst-case program execution time, which is important in demonstrating the temporal correctness of real-time systems. They performed their proposed technique in a simulation environment (an in-house developed environment) with seven SUTs. Out of these seven, four of them were small functions such as sorting or general mathematical functions with one or two input variables (n by n integer arrays with different ranges) or encoding functions with two short string inputs. The last three SUTs were very small artificial programs implemented to check the applicability of their GA-based approach for some specific execution time characteristics. The GA-based technique is compared with three other techniques: random search, best effort data generation (which is the best results among different examined methods, such as complete or partial systematic test data generation, data generation based on code inspection, and trial and error for execution time assessment), and static analysis of the code (that gives the actual WCET).

While evaluating different techniques, the effectiveness of each technique is measured by the execution time of the SUT using the generated input data set, and its average, max, and standard deviation across ten runs are reported. For all SUTs except one artificial program, GA performs equal to or better than random search and except in two cases, the same as best effort. The GA results are the same as real WCET (static analysis results), where the execution time is less than 1000 CPU cycles. Although GA always performs better than or equal to random search, the differences are not large and since there is no statistical hypothesis testing, it is difficult to find out if there is any significant difference between the results or not. Also, only the cost of the three artificial programs is reported and since the cost of random search is not reported we cannot perform any comparison. Since it is not clear whether the difference in effectiveness between the GA and random search are statistically significant, it would have been beneficial to have the cost of random search as well, in order to assess if the effort involved in using the GA is worth it. Another shortcoming is that the selected SUTs were not from their targeted application domain (real-time systems), which then introduces a threat to external validity.

b. Using Genetic Algorithms for Early Schedulability Analysis and Stress Testing in Real-Time Systems (Briand et. al., 2005 and 2006)

Briand et. al [16, 17] proposed a GA-based approach for stressing real-time systems by scheduling input sequences to maximize delays in the execution of target aperiodic tasks. The technique was applied on three SUTs: the first two were artificial tasks with given execution time priority (tasks from the second SUT were interdependent). The last SUT was a real-time avionics application, where eight high priority schedulable tasks were examined by the technique. The results showed that there are cases where their technique can schedule tasks to miss the deadline(s) even though schedulability analysis (Generalized Completion Time Theorem-GCTT) identified them as schedulable. The cost of the technique is evaluated by the average execution time which was between one to 46.5 minutes. A weak point that is seen in the study is the lack of comparison of their technique with random test data generation to see if a GA is warranted to solve the targeted test case generation problem.

c. Automatic Test Data Generation Using Genetic Algorithm and Program Dependence Graphs (Miller et. al., 2006)

This paper [19] presents an automatic test data generation approach that uses genetic algorithms along with program dependence graphs (PDGs). The premise is that branch coverage is influenced by the paths that are executed in the SUT. Hence, it is important to select those paths that will lead to the covering of the target branch. Static analysis, using the PDGs together with two new metrics, is used to select the appropriate paths to cover a particular branch. Once the appropriate path has been selected, constraint information linked to that path is used in the fitness function of a genetic algorithm in order to guide the search to cover the targeted branch (the whole approach is called TDGen). An empirical study to test the premise is presented that compares the results of using the static analysis with genetic algorithm to random search and to the previous work done by Michael et al. [70] using the GADGET framework for genetic algorithms. The technique was applied on six small programs such as bubble sort, quadratic formula and triangle classification.

The results showed that, for the simpler programs there is little difference in the results between random and TDGen. The difference is seen in larger programs where even though both random and TDGen obtain close to 100% statement and branch coverage, TDGen requires a much smaller number of generations, which is reported by mean, minimum, and maximum values across 10 runs. It is also observed that for some SUTs, TDGen can achieve 100% coverage, where random generation and GADGET cannot. The empirical study compares the three approaches based on the time taken by the search algorithms to reach the target, however the amount of time involved in performing the static analysis is not presented and this could be a factor in the cost of using the TDGen system. Even if the analysis is very fast and would not affect the results, lacking this information makes it difficult to interpret the cost results.

d. Evolutionary Test Data Generation: A Comparison of Fitness Functions (Watkins et. al., 2005)

Watkins et. al. in [20] evaluated the cost and effectiveness of ET for path coverage using seven different fitness functions proposed in previous studies and compared the results with random search. The comparison is performed on: (a) one small (30 LOC and 3 integer input variables) SUT with three different paths (one very easy or two very difficult to find in the search space) as test objectives, and (b) one program from industry (a tax benefit optimization) with around 300 LOC, a series of Boolean conditions, 31 input variables and thousands of paths. The mean, median, and standard deviation of the number of unique test cases generated across all generations is reported in addition to the completed runs (the number of times the GA succeeded in producing a test case that traversed the targeted path) and the average number of paths found for the cases, where the target is covering all paths. The statistical significance of differences between results is also examined using ANOVA on a path-by-path basis and wherever the F-test was significant, pair-wise comparisons using Tukey test were performed. Results show that there is no best fitness function for all SUTs and paths. Even though there is always at least one fitness function that works better than random search. Therefore, the conclusion was identifying the best two fitness functions in most cases and suggesting a two-step method using the two best fitness functions.

e. A Theoretical and Empirical Analysis of Evolutionary Testing and Hill Climbing for Structural Test Data Generation (Harman and McMinn, 2007)

An empirical evaluation of ET was presented in [3] to answer the questions such as when and why ET works and how it compares with other search techniques. The royal road theory for GA [3] was used as the basis to derive research questions. This theory states that GA with royal road fitness function will outperform local search algorithms such as hill climbing because of the way GA uses crossover operators. Based on this theory, the following research questions were formulated: 1) Does GA perform better than HC and random search for the branches having royal road functions? According to the theory, GA should perform better than HC and random search. 2) How does GA performs on the branches with royal road functions if crossover operators are removed? According to the theory, GA shouldn’t perform well if crossover operators are removed. 3) How does GA performs in comparison with HC for the branches that do not have royal road functions? According to the theory, GA should perform worst or no better than HC in this case. In order to confirm the stated theory, an empirical study was performed on six test objects chosen from real world applications. The size of the test objects ranged from 183 LOC to 2210 LOC. The total number of branches in all test objects was 640 and the search space of all test objects ranged from 10^7 to $10^{544.}$ In order to answer the first research question, branches having royal road features were selected as the targets for the search algorithms. GA was always able to find inputs to exercise these branches, whereas HC and random were never successful. For the second research question, the study was repeated with crossover operators removed for GA. In that case, GA wasn't successful in finding any inputs to exercise any of the branches. For the third research question, the branches that did not have royal road features were selected as the targets. In this instance, in almost all of the cases, HC was either better or equally as successful as GA. The study was repeated 30 times and *t*-tests were applied to determine the statistical significance of results.

f. The Impact of Input Domain Reduction on Search-Based Test Data Generation (Harman et. al., 2007)

In search-based test input data generation, the input domain is the search space of the system under test. The work in [26] investigates the relationship between the size of the search space and the effectiveness of search algorithms for test data generation. A theoretical analysis is presented regarding the effect of domain reduction on random, local, and global search, and four hypotheses are presented that predict the effects of input domain reduction on random search, hill climbing, and genetic algorithms. The study is conducted on six real world objects taken from industry and open source projects. The test objects consist of two embedded controllers, several functions selected from three open-source graphics manipulation packages, and one test object specifically designed for this empirical study that has a large input domain and branches that incrementally include all the variables of the input domain. The SUTs' size vary from 138 to 867 LOCs and the input domain size varies from 10^{10} to $10^{126.}$ The search space is reduced by removing the irrelevant variables for every target branch in the programs. The impact is measured by measuring the percentage of branch coverage achieved before and after input domain reductions.

The results indicate that for random search there is no relationship between search space reduction and reduction in cost. There is significant reduction in cost in terms of the number of fitness evaluations for 82 branches for hill climbing and for 86 branches for the genetic algorithm out of 360 branches in total. The improvement in the genetic algorithm is more than that seen in hill climbing and the percentage of improvement increases as the number of irrelevant variables removed increases. There is no relationship between search space reduction and search effectiveness in terms of coverage for any of the search algorithms. The success rate fluctuation is also reported for branches where there was a +/-5% change in the average number of fitness evaluations as well as the results of branches which differed significantly using paired t-tests. Cost information is also reported as the average number of fitness evaluations and the standard deviation per branch.

g. Improving Evolutionary Class Testing in the Presence of Non-Public Methods (Wappler and Schieferdecker, 2007)

A challenge in the unit testing of object-oriented systems is the testing of the non-public methods and attributes of classes without breaking the encapsulation rules of the class. Wappler and Schieferdecker [28] presented an approach for testing non-public methods without breaking encapsulation of the class by using only the public interfaces of the class under test. A new objective function for genetic programming is proposed for this purpose. A search space representation that includes both the method call sequences as well as the method parameters was also proposed. The accompanying empirical study compared their technique with the genetic programming system called ECJ [71] and random search on 34 real-world classes taken from open source Java projects. The details of only seven test objects are presented due to space restrictions. The size of the classes varies from 30 to 317 executable lines of code; the number of total branches to cover is from seven to 110 and the number of branches in non-public methods to cover varies from 1 to 70.

The results for the seven test objects show that their proposed technique achieved higher branch coverage than random search and the difference is statistically significant. The average of 50 runs is shown for the branch coverage of non-public methods with the proposed technique and standard Genetic Programming. Their technique had higher coverage for five of the test objects while for the others the results were the same. However, it would have been useful to know the cost of the random search and how this cost compares to that of the genetic programming algorithm. It would also be useful to know specifically which statistical tests were applied for hypothesis testing.

h. Empirical Evaluation of Optimization Algorithms When Used in Goal-oriented Automated Test Data Generation Techniques (Xiao et. al., 2007)

Different MHS algorithms were empirically evaluated for automated test case generation in [30]. The study was conducted on five small test objects written in C/C++. There were 36 to 87 test requirements to achieve full condition-decision coverage for all test objects and the size of the search space ranged from 2^6 to $2^{32.}$ The study was performed using different algorithms including GA, SA, Genetic Simulating Annealing (GSA), SA with Advanced Adaptive Neighbors (SA/AAN), and random search. In all 10 runs of the study, GA outperformed all other search techniques in terms of achieving condition-decision coverage in lesser number of fitness evaluations. SA/ANN was the second best after GA. For two of the test objects, the study was further extended to observe performance of ET for two different search spaces. Based on the study, it was concluded that SA and GSA performed well only for small search spaces. Generally, GSA performed slightly better than random search. The practical significance of differences between results is analyzed using Cohen's d for measuring effect size and the possible threats to validity were discussed and addressed. This empirical study was however performed on test objects that were small and simple, thus presenting an external validity threat.

Based on the information presented in Table XVI and summaries of the papers, it is apparent that there is a scarcity of convincing evidence for cost-effectiveness of ET techniques. Within this scarce evidence, we can see that most of the studies are very recent, either from 2006 or 2007. This is representative of an increased focus in software engineering on performing well-designed empirical studies as well as the fact that the field of ET is becoming more mature. The papers are also a representative sample from the different types of investigations that are performed with MHS algorithms for test case generation. MHS algorithms have been recently applied to increasingly diverse areas and this is seen in this sample of papers by comparing the content of the "test purpose" column across papers. This ranges from specialized purposes such as testing the performance of real time systems to more general purposes such as testing non-public methods in object-oriented programs. Despite the diversity of objectives, we can see that in most of these papers, MHS algorithms, mostly GA, were compared with random search and the results show that GA outperformed random search for the test case generation problems at hand. This suggests that this type of problems indeed requires guided search algorithms.

3. RQ3.3: Is there any evidence regarding the scalability of metaheuristic search algorithms for test case generation?

During our systematic review, we did not find any paper specifically targeting the scalability of the MHS algorithm in the context of evolutionary testing. However, there was one paper where the authors performed a small scale scalability analysis [30]. The study was conducted on five small test objects written in C/C++. There were 36 to 87 test requirements to achieve full condition-decision coverage for all test objects and the size of the search space ranged from 26 to 232. The study was performed using different algorithms including GA, SA, Genetic Simulating Annealing (GSA), SA with Advanced Adaptive Neighbors (SA/AAN), and random search. In two of the SUTs used for the study, two different search spaces (one small and one large) were used to measure the performance (condition-decision coverage vs. the number of SUT iterations) of different MHS algorithms and random search. Based on the empirical evaluation, it was concluded that GA performed well for both the small and the large search space. SA/ANN was the second best. SA and GSA performed well only for the small search space. All MHS algorithms performed better than random search. As a result, we can say that scalability analyses of ET techniques in the domain of test case generation are very rare and there is a need to focus more on scalability analysis in future studies.

4. Conclusion

Based on the discussions in the three sub-questions above, the number of papers which contain well-designed and reported empirical studies in the domain of test case generation using ET is very small. As a result, there is a limited body of credible evidence that demonstrate the usefulness of ET technique for test case generation. This evidence is, in addition, very partial as it mostly focuses on the use of genetic algorithms at the unit testing level. This evidence, however, consistently shows that the genetic algorithms outperform random search in terms of structural coverage. However, this evidence is just based on eight papers and cannot be generalized to state that genetic algorithms at the unit testing level will always outperform random search regardless of the test objectives. More empirical studies must be conducted to provide strong and generalizable evidence about the suitability and applicability of different MHS algorithms for test case generation at different testing levels and for other test objectives than structural coverage.

VI. THREATS TO THE VALIDITY OF THIS REVIEW

The main validity threats to our review are related to the possible incomplete selection of publications, inaccuracy of data extraction, and bias in quality assessment of studies.

A. Incomplete selection of publications

In Section IV.B, we have discussed and justified the systematic and unbiased selection strategy of publications. However, it is still possible to miss some relevant literature. One such instance is the existence of grey literature such as technical reports and PhD theses. In our case, this literature can be important if the authors report the complete study which is briefly reported in the corresponding published paper. In this review, we did not include such information.

Another instance that may lead to an incomplete selection of publications is the difficulty of finding an appropriate search string. In Section IV.B we provide justification for the repositories that we selected and the search string that we used. However, there may still be some papers, which have used some other related terms other than our keywords. We refined our search string several times because we found a paper missing from our selected papers, which was in the reference list of another paper. In order to deal with this problem, we refined our search string until it included all such papers and we were sure that our set of selected papers did not miss any paper that is referred to and relevant for this review.

B. Inaccuracy in data extraction

Inaccurate data can be the result of subjective and unsystematic data extraction or invalid classification of data items. In our review, we tried to deal with this problem by two means. First, we defined a framework, which clearly identified the data items that should be extracted. Second, all the data extracted was reviewed by three researchers and all discrepancies were settled by discussion to make sure that the extraction was as objective as possible. Therefore, the remaining problem is the validity of the framework itself. We have defined the framework based on the current guidelines for empirical studies in software engineering and adapted them to our domain of interest based on experience. Hence, we believe that it is a good starting point, but it can be further improved by feedback and discussion from other researchers in the domain.

C. Unbiased quality assessment

Assessing the quality of the papers for answering RQ3 was a challenging issue. Even though the data extracted from the papers to judge their quality was detailed and based on a well thought framework, the criteria used to select the papers themselves could be thought of as subjective. Our justification for the validity of this criterion is discussed in the Section V.C and we reemphasize the fact that this is the minimum requirement for having a valid empirical study in the domain of ET.

VII.CONCLUSION

The automation of test case generation has been a long-standing problem in software engineering. Evolutionary testing, or in other words the application of metaheuristic search (MHS) algorithms for test case generation, has shown to be a very promising approach for solving this problem by re-expressing test case generation problems as search problems. As a result, a great deal of research has been conducted and published. The time was therefore ripe to perform a systematic review of the state of the art and appraise the evidence regarding the cost-effectiveness of such an approach. A systematic review is very different from more informal, traditional surveys, in the sense that it aims at being comprehensive in its coverage and repeatability by relying on well-defined paper selection and analysis procedures. This systematic review focuses, due to space constraints, on one specific but crucial aspect: the way ET techniques have been empirically assessed. This aspect is highly important as all MHS algorithms are heuristics and therefore cannot guarantee their success in solving a test case generation problem or any other problem for that matter. Only an empirical investigation can provide the necessary confidence that a specific MHS algorithm is appropriate for a given test case generation problem.

In addition to a large-scale, systematic review, our contribution also includes guidelines, in the form of a framework, on how to conduct empirical studies in evolutionary testing. Results of our review have shown that the research reported so far has mostly focused on structural coverage and unit testing. However, the research is increasingly more diversified in the types of topics being tackled. Results also show that empirical studies in this field would benefit from more standardized and rigorous ways to perform and report studies. More specifically, three important empirical issues stand out from our analysis. Studies need to, more systematically and rigorously, account for the random variation in the results generated by any MHS algorithm. Such random variation implies that alternative techniques can only be compared by statistical means, that is, statistical hypothesis testing. This, unfortunately, is not performed well in most published papers and our framework provides guidelines about which statistical test to perform in which circumstance. Last, another important issue is that it is impossible to assess how a MHS technique performs in absolute terms: to be able to conclude on its usefulness to tackle a specific test case generation problem, a proposed technique needs to be compared with simpler and existing alternatives to determine whether it brings any advantage. This is again missing in an important number of papers and need to be carefully addressed by all studies in the future.

Despite the above limitations, credible results are available and existing results confirm that MHS algorithms are indeed promising for solving a wide variety of test case generation problems. Future research work will have to better establish their limitations and the types of problems for which they are applicable and required.

REFERENCES

[1] B. F. Jones, H. H. Sthamer, and D. E. Eyres, "Automatic structural testing using genetic algorithms," *IEEE Software Engineering Journal*, vol. 11, pp. 299-306, 1996.

[2] B. Beizer, *Software testing techniques*: Van Nostrand Reinhold Co., 1990.

[3] M. Harman and P. McMinn, "A theoretical empirical analysis of evolutionary testing and hill climbing for structural test data generation," in *Proceedings of the 2007 International Symposium on Software Testing and Analysis* London, United Kingdom: ACM, 2007.

[4] P. Puschner and R. Nossal, "Testing the results of static worst-case execution-time analysis," in *Proceedings of the 19th Real-Time Systems Symposium*, 1998, pp. 134-143.

[5] I. H. Osman and J. P. Kelly, *Metaheuristics: Theory and Applications*: Kluwer Academic Publishers, 1996.

[6] N. Tracey, J. Clark, and K. Mander, "Automated program flaw finding using simulated annealing," in *Proceedings of the International Symposium on Software Testing and Analysis* Clearwater Beach, Florida, United States: ACM SIGSOFT 1998.

[7] R. Drechsler and N. Drechsler, *Evolutionary Algorithms for Embedded System Design*: Kluwer Academic Publishers, 2002.

[8] P. M. S. Bueno and M. Jino, "Identification of potentially infeasible program paths by monitoring the search for test data," in *Proceedings of the 5th International Conference on Automated Software Engineering*, 2000, pp. 209-218.

[9] C. Hart, *Doing a Literature Review: Releasing the Social Science Research Imagination*: Sage Publications Ltd, 1999.

[10] K. S. Khan, R. Kunz, J. Kleijnen, and G. Antes, *Systematic Reviews to Support Evidence-Based Medicine: How to Review and Apply Findings of Healthcare Research*: Royal Society of Medicine Press Ltd 2003.

[11] C. C. Michael, G. McGraw, and M. A. Schatz, "Generating software test data by evolution," *IEEE Transactions on Software Engineering*, vol. 27, pp. 1085-1110, 2001.

[12] J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary test environment for automatic structural testing," *Information and Software Technology*, vol. 43, pp. 841-854, 2001.

[13] A. P. Mathur, *Foundations of Software Testing*: Pearson Education, 2008.

[14] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using artificial life techniques to generate test cases for combinatorial testing," in *Proceedings of the 28th Annual International Computer Software and Applications Conference*, 2004, pp. 72-77

[15] A. Bertolino, "Software Testing Research: Achievements, Challenges, Dreams," in *Future of Software Engineering*: IEEE Computer Society, 2007.

[16] L. C. Briand, Y. Labiche, and M. Shousha, "Using genetic algorithms for early schedulability analysis and stress testing in real-time systems," *Genetic Programming and Evolvable Machines*, vol. 7, pp. 145-170, 2006.

[17] L. C. Briand, Y. Labiche, and M. Shousha, "Stress testing real-time systems with genetic algorithms," in *Proceedings of the Genetic and Evolutionary Computation Conference* Washington DC, USA: ACM, 2005.

[18] M. Harman and B. F. Jones, "Search-based software engineering," *Information and Software Technology*, vol. 43, pp. 833-839, 2001.

[19] J. Miller, M. Reformat, and H. Zhang, "Automatic test data generation using genetic algorithm and program dependence graphs," *Information and Software Technology,* vol. 48, pp. 586-605, 2006.

[20] E. M. H. Alison Watkins, "Evolutionary test data generation: a comparison of fitness functions," *Software: Practice and Experience,* vol. 36, pp. 95-116, 2006.

[21] J. Clarke, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd, "Reformulating software engineering as a search problem," *IEE Proceedings on Software,* vol. 150, pp. 161-175, 2003.

[22] Y. Zhan and J. A. Clark, "Search-based mutation testing for Simulink models," in *Proceedings of the Genetic and Evolutionary Computation Conference* Washington DC, USA: ACM, 2005.

[23] Y. Zhan and J. A. Clark, "The state problem for test generation in Simulink," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation* Seattle, Washington, USA: ACM, 2006.

[24] M. S. B. Paulo, W. E. Wong, and J. Mario, "Improving random test sets using the diversity oriented test data generation," in *Proceedings of the 2nd International Workshop on Random Testing: co-located with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)* Atlanta, Georgia: ACM, 2007.

[25] K. Lakhotia, M. Harman, and P. McMinn, "A multi-objective approach to search-based test data generation," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* London, England: ACM, 2007.

[26] M. Harman, Y. Hassoun, K. Lakhotia, P. McMinn, and J. Wegener, "The impact of input domain reduction on search-based test data generation," in *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering* Dubrovnik, Croatia: ACM, 2007.

[27] P. McMinn, "Search-based software test data generation: A survey," *Software Testing, Verification and Reliability,* vol. 14, p. 52, 2004.

[28] S. Wappler and I. Schieferdecker, "Improving evolutionary class testing in the presence of non-public methods," in *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering* Atlanta, Georgia, USA: ACM, 2007.

[29] E. K. Burke and G. Kendall, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*: Springer 2006.

[30] M. Xiao, M. El-Attar, M. Reformat, and J. Miller, "Empirical evaluation of optimization algorithms when used in goal-oriented automated test data generation techniques," *Empirical Software Engineering,* vol. 12, pp. 183-239, 2007.

[31] B. A. Kitchenham, T. Dybå, and M. Jørgensen, "Evidence-Based Software Engineering," in *Proceedings of the 26th International Conference on Software Engineering*: IEEE Computer Society, 2004.

[32] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering: An Introduction*: Kluwer Academic Publishers, 2000.

[33] T. Dybå, B. A. Kitchenham, and M. Jørgensen, "Evidence-Based Software Engineering for Practitioners " *IEEE Software,* vol. 22, p. 8, January/February 2005 2005.

[34] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Transactions on Software Engineering,* vol. 28, p. 14, August 2002.

[35] J. J. Marciniak, *Encyclopedia of Software Engineering,* 2nd ed.: Wiley-Interscience, 2002.

[36] P. Ammann and J. Offutt, *Introduction to Software Testing*: Cambridge University Press, 2008.

[37] P. Tonella, "Evolutionary testing of classes," in *Proceedings of the International Symposium on Software Testing and Analysis* Boston, Massachusetts, USA: ACM SIGSOFT, 2004.

[38] R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms*: Wiley-Interscience, 1997.

[39] D. A. Coley, *An Introduction to Genetic Algorithms for Scientists and Engineers*: World Scientific Publishing Company, 1997.

[40] R. Nilsson and D. Henriksson, "Test case generation for flexible real-time control systems," in *the Proceedings of 10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 2005, p. 8 pp.

[41] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*: Addison-Wesley Professional, 2001.

[42] J. A. Capon, *Elementary Statistics for the Social Sciences*: Wadsworth Publishing Co Inc, 1988.

[43] "UML Profile for Modeling and Analysis of Real-time and Embedded Systems (MARTE)," 2008.

[44] T. Dybå, V. B. Kampenes, and D. I. K. Sjøberg, "A systematic review of statistical power in software engineering experiments," *Information and Software Technology,* vol. 48, pp. 745-755, 2006.

[45] D. S. Moore and G. P. McCabe, *Introduction to the Practice of Statistics*, Fourth ed.: W. H. Freeman, 2002.

[46] K. Y. Cai and D. Card, "An analysis of research topics in software engineering - 2006," *Journal of Systems and Software,* vol. 81, p. 8, June 2008 2008.

[47] B. A. Kitchenham, "Guidelines for performing Systematic Literature Reviews in Software Engineering," 2007.

[48] B. Baudry, F. Fleurey, J. M. Jezequel, and Y. Le Traon, "Automatic test case optimization: a bacteriologic algorithm," *Software, IEEE,* vol. 22, pp. 76-82, 2005.

[49] B. Baudry, F. Fleurey, J. M. Jezequel, and Y. Le Traon, "Genes and bacteria for automatic test cases optimization in the .NET environment," in *Proceedings of the 13th International Symposium on Software Reliability Engineering,* 2002, pp. 195-206.

[50] A. Bouchachia, "An Immune Genetic Algorithm for Software Test Data Generation," in *7th International Conference on Hybrid Intelligent Systems,* 2007, pp. 84-89.

[51] R. C. Bryce and C. J. Colbourn, "One-test-at-a-time heuristic search for interaction test suites," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* London, England: ACM, 2007.

[52] M. Harman, "The Current State and Future of Search Based Software Engineering," in *2007 Future of Software Engineering*: IEEE Computer Society, 2007.

[53] M. Srinivas and L. M. Patnaik, "Genetic algorithms: a survey," *Computer,* vol. 27, pp. 17-26, 1994.

[54] "GEATbx - The Genetic and Evolutionary Algorithm Toolbox for Matlab," 2007.

[55] "Genetic Algorithms Framework," 2004.

[56] S. Wappler and F. Lammermann, "Using evolutionary algorithms for the unit testing of object-oriented software," in *Proceedings of the Conference on Genetic and Evolutionary Computation* Washington DC, USA: ACM, 2005.

[57] K. Doerner and W. Gutjahr, "Extracting Test Sequences from a Markov Software Usage Model by ACO," in *Genetic and Evolutionary Computation — GECCO 2003*, 2003, pp. 214-214.

[58] S. Wappler and J. Wegener, "Evolutionary Unit Testing Of Object-Oriented Software Using A Hybrid Evolutionary Algorithm," in *IEEE Congress on Evolutionary Computation (CEC).* 2006, pp. 851-858.

[59] A. Watkins, D. Berndt, K. Aebischer, J. Fisher, and L. Johnson, "Breeding software test cases for complex systems," in *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, 2004, p. 10 pp.

[60] J. Wegener, H. Sthamer, B. F. Jones, and D. E. Eyres, "Testing real-time systems using genetic algorithms," *Software Quality Journal,* vol. 6, pp. 127-135, 1997.

[61] C. McCubbin, D. Scheidt, O. Bandte, S. Marshall, and I. Trifonov, "Using genetic algorithms for naval subsystem damage assessment and design improvements," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* London, England: ACM, 2007.

[62] F. E. Vieira, F. Martins, R. Silva, R. Menezes, and M. Braga, "Using genetic algorithms to generate test plans for functionality testing," in *Proceedings of the 44th Annual Southeast Regional Conference* Melbourne, Florida: ACM, 2006.

[63] R. Lefticaru and F. Ipate, "Automatic State-Based Test Generation Using Genetic Algorithms," in *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC).* , 2007, pp. 188-195.

[64] A. Habibi and S. Tahar, "Towards an efficient assertion based verification of SystemC designs," in *Ninth IEEE International High-Level Design Validation and Test Workshop.*, 2004, pp. 19-22.

[65] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*: Morgan Kaufmann, 2006.

[66] "IEEE Standard for Software Unit Testing -Description (ANSI/IEEE 1008-1987)," IEEE Computer Society 1987.

[67] M. Fewster and D. Graham, *Software Test Automation*: Addison-Wesley Professional 1999.

[68] D. J. Sheskin, *Handbook of parametric and nonparametric statistical procedures*, Third ed.: Chapman & Hall/CRC, 2003.

[69] S. A. Ghazi and M. A. Ahmed, "Pair-wise test coverage using genetic algorithms," in *The 2003 Congress on Evolutionary Computation (CEC).* 2003, pp. 1420-1424 Vol.2.

[70] C. C. Michael, G. McGraw, and M. A. Schatz, "Generating Software Test Data by Evolution," *IEEE Trans. Softw. Eng.,* vol. 27, pp. 1085-1110, 2001.

[71] G. C. Wilson, A. M. Intyre, and M. I. Heywood, " Resource Review: Three Open Source Systems for Evolving Programs–Lilgp, ECJ and Grammatical Evolution," *Genetic Programming and Evolvable Machines,* vol. 5, pp. 103-105, 2004.