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

Shaukat Ali, *Student Member*, *IEEE*, Lionel C. Briand, *Fellow*, *IEEE*,
Hadi Hemmati, *Student Member*, *IEEE*, and
Rajwinder K. Panesar-Walawege, *Student Member*, *IEEE*

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 "Search-based Software Testing" (SBST), has been used for a wide variety of test case generation purposes. Since SBST 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 SBST 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 SBST cost-effectiveness and what empirical evidence is available in the literature regarding SBST 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 SBST techniques can be empirically assessed. The intent is to aid future researchers doing empirical studies in SBST by providing an unbiased view of the body of empirical evidence and by guiding them in performing well-designed and executed empirical studies.

Index Terms—Evolutionary computing and genetic algorithms, frameworks, heuristics design, review and evaluation, test generation, testing strategies, validation.

1 Introduction

Software is being incorporated into an ever-increasing number 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 50 percent of the entire development cost [6], 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 [41]. The main reason for such an interest is that test case generation problems can often be reexpressed as optimization or search problems.

There has been a tremendous amount of research in applying MHS algorithms to test case generation and a large body of research exists: A search of the most relevant databases (as detailed in Section 4.2.1) found 450 articles which, after reading abstracts, resulted in 122 relevant articles published over the years 1996-2007 on this specific topic, often referred to as search-based software testing (SBST) [4].

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 SBST 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 SBST, we decided to conduct a comprehensive systematic review of the current literature, as this is commonly done in other scientific fields of research such as medicine [25] and social science [29]. The purpose of this systematic review is to collect, classify, and assess the empirical studies on SBST in order to assess the current body of evidence regarding the cost and effectiveness of SBST. 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 2, we provide the background relevant to the material presented in this paper. Section 3 suggests a framework used to assess the empirical studies in SBST, and Section 4 presents the method used to conduct this systematic review. In Section 5, we present the results of our review, while Section 6 outlines its validity threats. The final conclusions that we can draw from this systematic review are presented in Section 7.

- *The authors are with the Simula Research Laboratory, PO Box 134, 1325 Lysaker, Norway, and the University of Oslo, Norway.*
  E-mail: {shaukat, briand, hemmati, rpanesar}@simula.no.

Manuscript received 6 Sept. 2008; revised 27 Apr. 2009; accepted 7 Aug. 2010; published online 14 Aug. 2009.
Recommended for acceptance by M. Harman and A. Mansouri.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSESI-2008-09-0283.
Digital Object Identifier no. 10.1109/TSE.2009.52.

2 Background

In this systematic review, we are analyzing which MHS algorithms have been used to address test case generation 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: search-based software testing, systematic reviews, and empirical studies.

2.1 Search-Based Software Testing

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 [36]. 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 SBST. In order to perform test case generation systematically and efficiently, automated test case generation strategies are employed. Bertolino [7] addresses the need for 100 percent automatic testing as a means to improve the quality of the 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 [23].

Search-based software engineering attempts to solve a variety of software engineering problems by reformulating them as search problems [15]. 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 [15]. 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 with 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 nonlinear nature of software resulting from control structures such as if-statements and loops [17]. In such cases, simple search strategies may not be sufficient and global MHS algorithms[1] may, as a result, become a necessity, as they implement global search and are less likely to be trapped into local optima [16]. The use of MHS algorithms for test case generation is referred to as search-based software testing [4]. Mantere and Alander [35] discuss the use of MHS algorithms for software testing in general and McMinn [37] 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 search-based software testing are evolutionary algorithms, simulated annealing, hill climbing, ant colony optimization, and particle swarm optimization [12]. Among these algorithms, hill climbing (HC) [12] is a simpler, local search algorithm. The SBST techniques using more complex global MHS algorithms are often compared with test case generation based on HC and random search to determine whether their complexity is warranted to address a specific test case generation problem. The use of the more complex MHS algorithm may only be justified if it performs significantly better than HC.

2.2 Systematic Reviews

Systematic reviews are a means of synthesizing existing research regarding a specific research question [29]. 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 are also presented in the protocol. All of 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.

2.3 Empirical Studies for Search-Based Software Testing

Kitchenham et al. [19], [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 SBST techniques can be applied in practice, we need to conduct empirical studies to assess their cost-effectiveness and scalability. The cost-effectiveness of an SBST technique is normally measured in terms of the ability of the technique to

[1] Global MHS algorithms are often contrasted with local MHS algorithms. The former are based on strategies for the search to avoid being stuck in local minima, thus being more effective in situations with complex search landscapes [12].

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 SBST 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 [52]. 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 SBST in order to be comprehensive in our investigation.

In order to determine what constitutes a proper empirical study in SBST, we looked at existing guidelines [27], [32], [52] for conducting empirical studies in software engineering and those for evaluating SBST techniques in other fields. Wohlin et al. [52] and Kitchenham et al. [32] present guidelines on how to conduct experimentation and empirical research in the specific context of software engineering, whereas Johnson [27] presents a general guide for experimental analysis of algorithms. We have tailored and augmented some of these guidelines to create a specific framework for conducting and reporting empirical studies in the domain of SBST. This was necessary as SBST studies involve the analysis of automation techniques in which no human subjects are involved and present many specific challenges. In addition, the fact that SBST techniques are based on MHS algorithms makes it important to account for the inherent random variation that exists in their results. Furthermore, there should also be some means to show that an SBST 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 SBST 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 SBST. 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.

3 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 SBST 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 and reproducible. Whenever relevant, we will refer to Johnson's general guidelines on the experimental analysis of algorithms [27], either to point the reader to further, more general considerations, or to show that our more specific guidelines are a specialization of these more general ones.

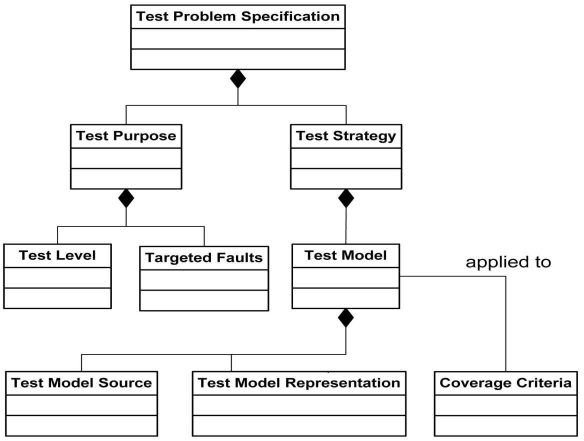

Fig. 1. Concept diagram of test problem specification.

3.1 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 search-based software testing strategy will take. Fig. 1 outlines the constituent parts of a test problem specification. The general purpose of software testing is to gain sufficient confidence in the dependability of a software artifact. Explicitly, this is usually done by targeting specific types of faults at different levels (such as unit, integration, and system testing). The targeted faults can be categorized in many ways, depending on the view one takes of a system. At the highest level, one differentiates functional from nonfunctional faults, e.g., faults related to performance, security, robustness, and safety requirements.

A testing strategy is defined by a model of the 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. 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. As discussed in [5], possible sources for a test model can be the SUT specification, design artifacts, or the source code itself. Based on the model source (specification, design, or source code), different types of test models can be constructed. Typical examples of models derived from source code include control and data flow graphs, whereas test models based on SUT design include state machines or Markov usage models. 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

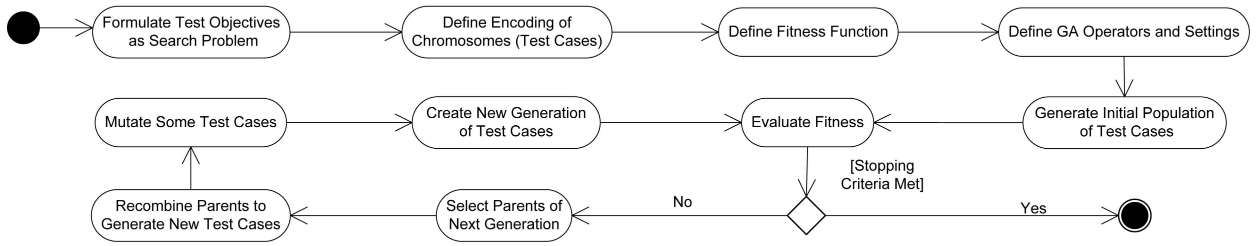
Formulate Test Objectives as Search Problem
Define Encoding of Chromosomes (Test Cases)
Define Fitness Function
Define GA Operators and Settings
Generate Initial Population of Test Cases
Evaluate Fitness
[Stopping Criteria Met]
Yes
No
Select Parents of Next Generation
Recombine Parents to Generate New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

Fig. 2. Test case generation using genetic algorithms.

stronger requirement, while in the case of control flow graphs, a typical coverage criterion is branch coverage. It is important to clearly specify the coverage criteria as it is often used to measure the effectiveness of SBST techniques regarding test case generation.

3.2 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 SBST 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, we show how a typical genetic algorithm can be used for test case generation. The important parameters to report for a genetic algorithm would be the encoding of the chromosomes, the fitness function created to guide the search, the strategy for creating the initial population, the selection strategy for selecting parents for the next generation, the various recombination operators, such as crossover and mutation operators and their values, the reinsertion strategy, and the stopping criteria. We discuss in [5] how these parameters affect the results of empirical studies involving the use of genetic algorithms for test case generation.

3.3 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.

3.3.1 Objectives and Experimental Hypotheses

One must define what is going to be empirically assessed and compared. The objective is usually to compare various SBST 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 SBST technique, a comparison of two or more SBST techniques, or a comparison of SBST techniques versus non-SBST techniques (i.e., not relying on metaheuristic search algorithms). The latter includes, for example, random search, static analysis, greedy algorithms, or some other specific technique for the test problem under consideration, e.g., schedulability analysis in the case of real-time systems. In any case, what is going to be compared should be precisely specified through formal test 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 [14]. This has an impact on how we interpret the results in terms of p-values (probability of type I errors). In the context of SBST, a one-tailed hypothesis would be used in the case when, based on the properties of the fitness function, we have a theoretical basis to assert the direction of the expected outcome. For example, when comparing a guided search algorithm such as a genetic algorithm with random search, we may, based on an analysis of the fitness function, expect the genetic algorithm to be equally or more effective at hitting the search target—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 possible 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.

3.3.2 Target Application Domain

Empirical studies should specify a target application domain in which their results are intended to be generalized. Example application domains are: real-time, concurrent, distributed, embedded, and safety-critical. 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 [3] to design a system and then derive a test model, this is, of course, more realistic in the context of embedded, real-time applications. Further, the selection of subject systems for empirical studies will then be partly determined by the target application domain.

3.3.3 Subject Systems (Software under Test or SUT) Specification

After identifying the target application domain, specific SUTs fitting that domain are selected. It is important to carefully select SUTs and precisely 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. This discussion should be in terms of the inherent properties of the SUT such as its size, complexity, or structure. This is particularly important when one is creating artificial SUTs specifically for the experiment, a common situation when one is trying to account for SUTs of varying size and complexity. For each SUT in the empirical study, the function of the SUT together with relevant properties affecting its representativeness of the domain should be carefully reported in order to ensure the reproducibility of the experiment and help future comparisons of cost-effectiveness results. Johnson [27] discusses the general problem of instance selection (i.e., SUTs here) in experiments (Principle 3: Use instance testbeds that can support general conclusions) and defines reproducibility (Principle 6: Ensure Reproducibility) when experimenting with algorithms as the capacity to “perform similar experiments that would lead to the same basic conclusions.” The goal is to make it possible to confirm the results of an original experiment independently from the precise settings and details of the experiment. In addition to SUT properties, the hardware platform that the SUT executes on is also important to specify. Johnson [27] provides an in-depth discussion of the latter issue (Principle 7: Ensuring Comparability), which is not specific to SBST, and suggestions to address it. In his Principle 9, about well-justified conclusions, Johnson [27] also discusses the danger of drawing conclusions from small instances that are then generalized to much larger instances, as the former do not always predict the latter well, and recommends using instances that are as large as possible.

3.3.4 Measures of Cost and Effectiveness for SBST Techniques

Measuring effectiveness and, more particularly, cost in our context is inherently difficult and the validity of measures is very often context-dependent. As discussed by Johnson in [27] (Principle 6: Ensure Reproducibility), just reporting effectiveness and cost values is not very informative as it does not provide direct insights into what these values actually imply. It is nevertheless crucial in order to draw useful conclusions from studies involving SBST techniques to be able to use appropriate comparison baselines. In our context, one usually resorts to comparing the investigated technique to simpler, existing techniques (see Section 3.3.6 on baselines of comparisons) in order to assess the relative goodness of a search. 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 effectiveness and cost 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 SBST, 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, which can be referred to as coverage-based measures and fault-based measures. In the former category, there may be many different types of measures, depending on the adequacy criteria being used, for example, control-flow coverage criteria such as 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 are estimated through mutation analysis [48]. In the latter case, the program is seeded with faults based on mutation operators and depending on the number of faults caught, a so-called mutation score is calculated. The techniques are assessed upon how successful they are at detecting the seeded faults.

Cost measures are generally related to the speed of the technique to converge toward the test objective (in some cases, it is referred to as the search technique’s “efficiency”). Some common cost measures used in the SBST domain are:

1. The number of iterations, which shows how many times an SBST 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.
2. The cumulative number of individuals in all iterations (usually, each individual represents a test case in SBST).
3. 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).
4. The time spent by an 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 if 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.
5. The size of the resulting test suite, which is a surrogate measure for the cost of the time it would take to execute the resulting test suite since a larger test suite would require more resources to execute.

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 that are kept from the previous population and there is no cost for generating them. Therefore, the number of evaluations can more precisely estimate the real cost of an SBST technique. All three of these 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 simulated annealing (SA) and genetic algorithms (GAs) 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 given expected time constraints. 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 have. When comparing the cost of different techniques, it is also necessary to make sure that any other required resources are kept equal among 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. Therefore, all relevant types of resources must be accounted for when comparing the cost of SBST techniques.

3.3.5 Measures for Scalability Assessment

Scalability assessment is the process of assessing how the cost-effectiveness of an SBST 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 SBST techniques under investigation. Some examples of measures that can be scaled up include the size of the SUT in terms of lines of code or the 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 SBST technique is still cost-effective as the SUT gets larger and more complex.

3.3.6 Baselines for Comparison

An SBST technique can only be assessed if it is compared with a carefully selected, meaningful baseline since the optimal solution is normally not known. Since it is difficult to assess SBST techniques in absolute terms, it is therefore important to show, at 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 SBST 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 SBST 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 SBST technique. It is also preferable to use other simple SBST techniques, such as HC, as a comparison baseline for other more expensive SBST techniques. This further demonstrates that the use of an SBST technique is justified given the test case generation problem at hand. In addition—but this is context-dependent—other SBST techniques, previously published or considered plausible alternatives, can also be used as baselines of comparisons for the proposed SBST techniques.

As discussed in [27], once baseline techniques are selected, one must ensure that reasonably efficient implementations are used for all techniques in order for cost and effectiveness to be comparable. Documentation, source code, and URLs for downloadable tools or, at the very least, a careful description of the implementation, should be provided.

3.3.7 Parameter Settings

Most SBST 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 therefore highly important, to make any study reproducible, to specify these parameters in a precise manner. It is also interesting to justify their values based on existing studies when possible, as this provides insights into how cost and effectiveness could be affected if they were changed or if a different SUT with different properties was used. One particularly important parameter in our context is the stopping criterion of the search (Principle 6: Ensure Reproducibility). It can be based on whether the search objective has been reached (or one is sufficiently close), execution time or a surrogate measure (due to practical constraints), or any significant progress is observed over a period of time.

3.3.8 Accounting for Random Variation in SBST Results

Since SBST 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 10, so as to allow the use of basic statistical hypothesis testing and obtain a reasonable statistical power to detect large differences [52]. 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 [18].

When dealing with multiple runs, in our context we are often interested in the best run yielding the best test suite or test case according to some fitness function (e.g., bringing the execution time of a task as close as possible to its deadline). Another frequent case is when we are interested in the frequency with which a certain target was reached across runs (e.g., test input data satisfying certain constraints). In both cases, it is important to report the execution time and other cost measures of all runs and, when relevant, information about their fitness distribution. The basic principle is that it should be possible to estimate the total cost of achieving the best solution or, depending on what is relevant, the expected cost to achieve the search target. From a more general standpoint, Johnson (Principle 6: Ensure Reproducibility) [27] warns against reporting only effectiveness and cost data for the best run.

3.3.9 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.

Data analysis methods for comparing cost-effectiveness. Performance in the case of SBST usually relates to measuring the cost-effectiveness of the various search techniques. The cost and effectiveness of an SBST 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

TABLE 1
Mapping of SBST Problems to Statistical Tests

SBST Analysis Type	Type of Statistical Comparison	Example in the Context of SBST	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 a SBST technique	Comparing maximum branch coverage achieved across all runs between two SBST 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 a SBST technique	Comparing the proportion of runs finding deadlocks across different SBST techniques	*z*-score test for proportions
Comparing samples of target in terms of 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 SBST 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

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 percent 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 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:

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 SBST technique. This, in turn, may help determine how many runs would, in practice, be necessary to guarantee that we obtain a satisfactory result, i.e., achieve the objective.

Results of hypothesis testing. The purpose of statistical testing is to determine whether differences across SBST 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 SBST 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 [40] for more details.

Statistical hypothesis testing should be used to accept/reject research hypotheses related to the cost-effectiveness analysis of SBST techniques and comparison baselines. The choice of a specific statistical test depends on the specific objective of SBST. 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 SBST 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 SBST 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, because 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 drawing incorrect conclusions from the analysis. Statistical tests are usually classified as parametric and nonparametric [52]. When the sample follows a specific distribution (e.g., normal), certain parametric tests are applicable (e.g., t-test). Alternatively, nonparametric 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 1, as a guideline, we provide a mapping between the analysis situations we have encountered in SBST 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.

Data analysis should address both the statistical and the 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 shows 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 that 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.

Data analysis methods for scalability. Scalability is used to assess whether an SBST 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 SBST 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 SBST 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 SBST 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.

3.3.10 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 [52]. The following types of threats can be discussed:

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.

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

Conclusion validity threats

- Has random variation been properly accounted for? Since SBST 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 versus 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 [52].
- 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.

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, SBST 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.

4 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.

4.1 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 [13]. In this paper, we are interested in investigating empirical studies in the domain of SBST. To proceed with our investigation, we defined the following three research questions:

RQ1: What is the research space of search-based software 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 SBST, 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. Because of size constraints, RQ1 will not be addressed in detail in this paper and the results will simply be summarized to provide context information to the reader and facilitate the interpretation of subsequent research results. Interested readers may consult the technical report [5] corresponding to this paper for a detailed discussion of the results.

RQ2: How are the empirical studies in search-based software testing designed and reported?

A study that has been properly designed and reported (as discussed in Section 3) is easy to assess and replicate. The following subquestions 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 search-based software testing, accounted for in the design of empirical studies?
- RQ2.2: What are the most common alternatives to which SBST techniques are compared?
- RQ2.3: What are the measures used for assessing cost and effectiveness of search-based software testing?
- RQ2.4: What are the main threats to the validity of empirical studies in the domain of search-based software testing?
- RQ2.5: What are the most frequently omitted aspects in the reporting of empirical studies in search-based software testing?

RQ3: How convincing are the reported results regarding the cost, effectiveness, and scalability of search-based software 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 subquestions:

- 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 search-based software 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?

4.2 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 and 2) inclusion or exclusion of studies based on certain inclusion and exclusion criteria.

4.2.1 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*, *Springer*, and *MIT Press*. The first two repositories covered almost all important conferences, workshops, and journal papers which are published by either the IEEE or the ACM. The next four repositories were mostly used for finding papers that are published in leading software engineering journals.

We selected the following journals based on [13]: *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 SBST, we also included journals relating to software quality assurance and evolutionary computing: *Springer: Software Quality Journal*, *Springer: Genetic Programming and Evolvable Machines, IEEE: Transactions on Evolutionary Computation*, and *MIT Press: Evolutionary Computation*. Another important source of publications that we included was the *Genetic and Evolutionary Computation Conference (GECCO)*. Based on the impact factor, GECCO is one of the top conferences in the fields of artificial intelligence, machine learning, robotics, and human-computer interaction [1] and is directly related to the field of genetic and evolutionary computation. GECCO's proceedings were published by Springer in 2003 and 2004 and afterward 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 SBST, 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 such as "*evolutionary algorithm*," "*metaheuristic*," 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 that we do not miss any papers that have explicitly used the widely used term {(((*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 3. The search string used for selecting the papers from repositories.

"*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 led to the search string shown in Fig 3.

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.

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*," "*metaheuristics*," "*genetic algorithms*," and so on.

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

4.2.2 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 450 (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 of 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 122 papers.

At the second stage, we again divided the papers into three equal sets and divided them among three researchers to check the contents of each paper. 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 2.3 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. It is important to mention that we didn't exclude papers based on the realism of SUTs used in their case studies. The reason is that exclusion would then be subjective, as no precise criterion can be defined, and would probably lead to a very small number of selected papers. After applying the second phase of selection, we had 68 papers remaining that contained empirical studies about test case generation using MHS algorithms. However, four of these 68 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 68 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 2.

TABLE 2
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
MIT Press	1	1	1
Total	450	122	68

4.2.3 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 [30] such as name of the paper, authors' names, a brief summary, researcher's name, and

TABLE 3
Research Questions and Type of Data Collected

Research Questions		Type of Data Collected
RQ 1		Type of MHS algorithms, test levels, targeted faults, test model, type of test cases, and 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

additional comments by the researcher. The second set included the information directly related to answering the research questions (see Table 3 for a summary list and [5] and Section 3 for details on each data item). To assess and improve the consistency of data extraction among the researchers, a sample of papers was selected and read by all researchers and the relevant data extracted. The extracted data were 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 among 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.

5 Results

The following section outlines the results related to the research questions. 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, and as such, results are compiled in structured tabular form.

5.1 RQ1: What Is the Research Space of Search-Based Software Testing?

As previously mentioned, we provide here only the most salient results to the research question. The reader is invited to read the technical report [5] corresponding to this paper to obtain detailed results. The results show that in the majority of the papers, SBST techniques have been applied at the unit testing level (75 percent). Moreover, most papers (78 percent) do not target any specific faults, but rather focus on structural coverage of different test models. The most commonly used algorithm is the GA and its extensions (73 percent), followed by a more limited use of simulated annealing and its extensions (14 percent). 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 [21]. Furthermore, substantial empirical data are 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 [46]. This, together with the many books [16], [26] 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 have been shown to usually perform better than local search algorithms [53], though there is no evidence showing that GA is better than other global search algorithm [21]. Last, GAs have many well-known implementations in the form of commercial tools [42] and frameworks [2], [34], which greatly facilitate their practical application.

5.2 RQ2: How Are the Empirical Studies in Search-Based Software 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 search-based software testing. To answer this question, we further divided this question into five subquestions. By answering each subquestion individually, we will answer the main research question. Though the results are presented in tables that summarize the main findings, the reader can obtain a breakdown of which papers led to these findings in the technical report [5] corresponding to this paper.

5.2.1 RQ2.1: How Well Is the Random Variation Inherent in Search-Based Software 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 3.3. 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 the HC algorithm, if it is started from the same starting point in each run using the same strategy to select neighbors, then all of the runs will not be independent, and hence, the algorithm will find the same solution every time. Different runs in HC are normally made independent by choosing different starting points in each run or by using a random strategy to select neighbors. Additionally, the number of runs for each MHS algorithm had to be at least 10, a ballpark figure to enable the application of statistical hypothesis testing with minimal

TABLE 4
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%

statistical power. Papers that did not report the number of runs or were executed less than 10 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 subcategory "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. However, most of the papers did not have detailed information on sample distributions and the validity of statistical test assumptions. It was therefore usually not possible to determine if a paper used the correct statistical procedure for a particular problem and data set.

The results in Table 4 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 10 times.

Among 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 among 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 was used: *t*-test, paired *t*-test, Mann-Whitney test, F-test, ANOVA, and Tukey test [40], [44]. There was one paper in this subcategory 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 percent of the papers did not account for random variation at all, and 38 percent of the papers only had "poor" descriptive statistics, so, in total, 77 percent of papers either did not account for random variation or reported it poorly. The remaining 23 percent of papers are divided between 12 percent providing only good descriptive statistics and just 11 percent 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 SBST 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.

5.2.2 RQ2.2: What Are the Most Common Alternatives to Which SBST Techniques Are Compared?

In assessing the cost-effectiveness of any technique, the comparison baseline is an important factor. In order to classify the papers, we defined four categories of comparison baselines:

1. "Global SBST," where the baseline of comparison is an SBST technique using a global MHS algorithm.
2. "Local SBST" includes the techniques that use a local MHS algorithm such as HC.
3. "Non-SBST" baselines do not use an SBST technique and feature baselines such as random search.
4. "Not discussed" addresses papers that do not report any comparison baseline.

The comparison to non-SBST techniques or local SBST 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 a more complex SBST technique is necessary. In addition, a simple baseline of comparison is necessary to assess the benefits of using complex SBST techniques.

As shown in Table 5, 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 an 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 SBST 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.

There were 34 studies that reported "Non-SBST" baselines within which random search is used in 24 studies, static analysis in three, greedy algorithm in three, constraint solving in one study, and three studies used some other technique specific to their context. We see that random search is the most commonly used comparison baseline among Non-SBST techniques. There is limited use of "Local SBST" baselines with only three studies using HC. There are many studies (33) that used Global SBST techniques as comparison baselines. This is usually done when investigating the effects of different parameter settings of MHS

TABLE 5
Comparison Baselines Used in SBST in Terms of Number of Papers

Global SBST baselines			Local SBST baselines	Non-SBST baselines					Not Discussed
GA and Extensions	SA and Extensions	Others	Hill Climbing	Random Search	Static Analysis	Greedy Algorithm	Constraint Solving	Others	
22	6	5	3	24	3	3	1	3	16

TABLE 6
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

algorithms. This is most evident within GA and SA, where 22 studies used either GA or its extensions as baselines and six studies used SA and its extensions.

5.2.3 RQ2.3: What Are the Measures Used for Assessing Cost and Effectiveness of Search-Based Software Testing?

Assessing the cost-effectiveness of SBST 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.

Effectiveness measures. As is discussed in Section 3, effectiveness measures are categorized into two main classes: coverage-based and fault-based measures. Under the coverage-based category, we found three main subcategories: 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 SBST techniques are used for testing combinatorial designs [36]. 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 subclasses among them and labeled the rest as miscellaneous. Papers in the first subcategory use different kinds of measures related to the execution time of test cases and we called these time-based measures. The second subcategory addresses the distribution of fitness values of individuals (solutions) as the measure of effectiveness (e.g., average and 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 [8], [9], 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 6 presents the number of papers in our review per the category of effectiveness measures.

The data we collected revealed 61 papers using one or more effectiveness measures in a total of 72 different effectiveness measurements across reported studies. There were three papers that did not discuss the effectiveness of the SBST technique at all. There were 47 instances (65 percent) 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 percent). Among them, 23 instances (32 percent) 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 percent) 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 percent) 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 SBST techniques to generate test cases that will find the best/worst-case execution time of a program.

Looking at the results in Table 6, 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 [37] on control flow graphs. Fault-based effectiveness measures received relatively little attention in the literature reporting SBST studies as compared to coverage-based measures. Similarly, the applications of SBST techniques to artifacts other than code are rare as white-box testing seems to have been by far the main focus.

TABLE 7
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

Cost measures. Based on the definition of cost measures in Section 3 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 subcategories:

1. The number of iterations.
2. The cumulative number of individuals in all iterations.
3. The number of fitness evaluations an algorithm needs to find the final solution.
4. 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 7 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 aforementioned classification, 62 instances (86 percent) 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 percent). A total of six instances (4 percent) used the number of individuals (test cases) and the number of fitness evaluations is used by 14 instances (20 percent) as the measure of cost. Finally, there were 15 instances (21 percent) that used the "test case generation time" measure.

In the second main category, "cost of executing the final test suite," the size of the 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 SBST technique was actually creating test suites with minimum size for covering a specific criterion such as a minimal test suite that exhibits pairwise coverage [20].

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 3. Another conclusion is that most studies use cost measures only for comparison purposes with other alternative techniques. There are just 15 instances (21 percent) that used measures such as test case generation time, which conveys whether a particular technique is likely to be practical and scale up.

5.2.4 RQ2.4: What Are the Main Threats to the Validity of Empirical Studies in the Domain of Search-Based Software Testing?

In order to answer this question, we carefully assessed the studies using the proposed framework in Section 3. 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 SBST 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 SBST techniques and cannot demonstrate the practicality of SBST 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 SBST techniques. As discussed in RQ2.1, 39 percent of the papers did not take the random variation of results into account and 38 percent 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 3.3 and RQ 2.1) make it very difficult to confidently draw practical conclusions from the results reported in most studies. Moreover, among the 11 percent of papers that discussed statistical hypothesis tests, just one paper has discussed the practical significance of differences, which 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 are not done through a well-identified and available 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 the tool might not be reliable, and hence might lead to an internal validity threat. A possible way to deal with this

TABLE 8
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%

validity threat is to use readily available (open source, downloadable, or commercial) tools 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.

5.2.5 RQ2.5: What Are the Most Frequently Omitted Aspects in the Reporting of Empirical Studies in Search-Based Software 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 percent 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 percent discussed conclusion validity, 6 percent discussed external validity, 3 percent discussed construct validity, and only 3 percent of the papers discussed internal validity threats. We found that only two papers out of 64 specified formal hypotheses, 44 percent of the papers discussed object selection strategies, and 39 percent of the papers described their data collection methods. Parameter settings (see [5]) were discussed by most, but not all of the papers (88 percent). However, all papers did not discuss all parameters required for their study; usually, there was only a partial discussion. In some cases, the authors provided justification of why they chose particular values for the parameters, but this was rare.

Summarizing the above information, Table 8 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.

5.2.6 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 the cost and effectiveness of SBST 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 SBST by comparing it with simpler techniques such as random search or HC. These two factors are highly important for yielding interpretable empirical studies in the context of test case generation using SBST 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 SBST techniques are still not properly conducted and reported and that improving this situation should be an important objective of the research community for future studies.

5.3 RQ3: How Convincing Are the Reported Results Regarding the Cost, Effectiveness, and Scalability of Search-Based Software 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 the evidence regarding the cost, effectiveness, and scalability of SBST techniques is. 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 3, a well-designed study in the context of SBST 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 an 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-SBST technique (such as random search, static source code analysis, or some other technique applicable to the test problem under consideration) or with HC. The first subquestion, 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 SBST. In subquestion 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.

5.3.1 RQ3.1: For Which Metaheuristic Search Algorithms, Test Levels, and Fault Types Is There Credible Evidence for the Study of Cost-Effectiveness?

This subquestion 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-SBST or local SBST techniques. 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 either a non-SBST or a simple, local MHS 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 did not have a non-SBST or local MHS 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, these studies are not considered as credible evidence since they do not show, in any way, that a simple non-SBST technique such as random search or a local MHS such as HC 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 an 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 9 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. These papers are referred to as "minimum criteria papers" in Table 9.

As can be seen in Table 9, among the 18 papers that report credible evidence, most papers (13 out of 18) applied an SBST 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 5.1. There are also only two papers that target specific faults: one targeting functional faults and the other nonfunctional faults.

5.3.2 RQ3.2: How Convincing Is the Evidence of Cost and Effectiveness of Search-Based Software Testing Techniques, Based on Empirical Studies That Report Credible Results?

Along with accounting for random variation in the results and having a non-SBST or local MHS 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, referred to as "sufficient criteria papers," are summarized in Table 10.

Based on the information presented in Table 10, it is apparent that there is a scarcity of convincing evidence regarding the cost-effectiveness of SBST techniques. Nevertheless, these papers are 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 types of problems 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 nonpublic 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 problem indeed requires guided search algorithms. It would also be interesting to see how the quality of the empirical studies that have been performed in this field have improved over the years. In order to investigate this, we compare three series, as shown in Fig. 4.

The "All Papers" series shows the number of papers per year expressed as a percentage of the total number of papers (64 papers). The "Minimum Criteria Papers" series shows the percentage per year of the papers satisfying our minimum criterion of accounting for random variation (as reported in Table 9) and the "Sufficient Criteria Papers" series shows the percentage per year of papers satisfying our secondary criteria of having an appropriate baseline and proper descriptive statistics or results of statistical hypothesis testing (as reported in Table 10). From Fig. 4, we can see that 40 percent of all papers, 55 percent of all minimum criteria papers, and 88 percent of all sufficient criteria papers were published in recent years (2006 and 2007). The trends that become apparent are that, first, the number of SBST publications has been steadily growing over the years, and second, the quality of empirical studies has increased dramatically in recent years.

5.3.3 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 SBST. However, there was one paper where the authors performed a small-scale scalability analysis [53]. The study was conducted on five small test objects written in C/C++. There were 36-87 test requirements to achieve full condition-decision coverage for all test objects and the

TABLE 9
Test Levels, Fault Types, and the Type of Metaheuristic Algorithms Used by “Minimum Criteria Papers”

	Test Level			Fault Type		Type of Metaheuristic Search Algorithm						
Paper	Unit	Integration	System	Non-Functional	Functional	GA	EGA	SA	ESA	ACO	GP	PSO
Jones *et al.* [28]	√	–	–	–	–	√	–	–	–	–	–	–
Puschner and Nossal [43]	√	–	–	–	–	√	–	–	–	–	–	–
Tracey *et al.* [47]	√	–	–	–	–	-	–	√	–	–	–	–
Bueno and Jino [10]	√	–	–	–	–	√	–	–	–	–	–	–
Michael *et al.* [38]	√	–	–	–	–		√	–	–	–	–	–
Wegener *et al.* [51]	√	–	–	–	–	√	–	–	–	–	–	–
Shiba *et al.* [45]	–	–	√	–	–	√	–	–	–	√	–	–
Briand *et al.* [8, 9]	–	–	√	√	–	√	–	–	–	–	–	–
Miller *et al.* [39]	√	–	–	–	–	√	–	–	–	–	–	–
Watkins and. Hufnagel [50]	√	–	–	–	–	√	–		–	–	–	–
Zhan and Clark [54]	–	–	√	–	√	–	–	√	–	–	–	–
Zhan and Clark [55]	–	–	√	–	–	–	–	√	√	–	–	–
Bueno *et al.* [11]	–	–	√	–	–	–	–	–	–	–	–	√
Lakhotia *et al.* [33]	√	–	–	–	–	–	√	–	–	–	–	–
Harman and McMinn [24]	√	–	–	–	–	√	–	–	–	–	–	–
Harman *et al.* [22]	√	–	–	–	–	√	–	–	–	–	–	–
Wappler and Schieferdecker [49]	√	–	–	–	–	√	–	–	–	–	–	–
Xiao *et al.* [53]	√	–	–	–	–	√	–	√	√	–	–	–

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 versus 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 SBST 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.

5.3.4 Conclusion

Based on the discussions in the three subquestions above, the number of papers which contain well-designed and reported empirical studies in the domain of test case generation using SBST is very small. As a result, there is a limited body of credible evidence that demonstrates the usefulness of SBST techniques 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 test objectives other than structural coverage.

TABLE 10
Test Purposes, Comparison Baselines, and Result Highlights for the "Sufficient Criteria Papers"

Paper	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 in 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 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 [24]) 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 (number of SUT executions) and effectiveness (condition decision coverage).

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

6 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.

6.1 Incomplete Selection of Publications

In Section 4.2, 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 gray 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

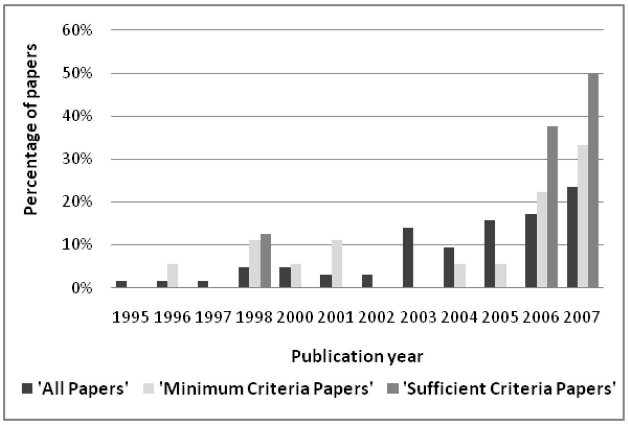

Fig. 4. Quality trends of SBST empirical studies based on the publication year.

appropriate search string. In Section 4.2, 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.

6.2 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 of the data extracted were 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.

6.3 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 were detailed and based on a well-thought-out 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 Section 5.3, and we reemphasize the fact that this is the minimum requirement for having a valid empirical study in the domain of SBST.

7 Conclusion

The automation of test case generation has been a long-standing problem in software engineering. Search-based software testing, or, in other words, the application of MHS algorithms for test case generation, has been shown to be a very promising approach for solving this problem by reexpressing 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 in which SBST 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 search-based software 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 an 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 needs 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.

Acknowledgments

The authors wish to thank the Simula School of Research and Innovation (SSRI) for funding this work.

References

[1] "Computer Science Conference Ranking," http://www.cs-conference-ranking.org/conferencerankings/topicsii.html., 2008.

[2] "Genetic Algorithms Framework," Rubicite Interactive, http://sourceforge.net/projects/ga-fwork, 2004.

[3] "UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE)," Object Management Group (OMG), http://www.omg.org/cgi-bin/doc?ptc/2008-06-08, 2008.

[4] W. Afzal, R. Torkar, and R. Feldt, "A Systematic Review of Search-Based Testing for Non-Functional System Properties," *Information and Software Technology,* vol. 51, pp. 957-976, 2009.

[5] S. Ali, L.C. Briand, H. Hemmati, and R.K. Panesar-Walawege, "A Systematic Review of the Application and Empirical Investigation of Evolutionary Testing," Technical Report Simula.SE.293, Simula Research Laboratory, 2008.

[6] B. Beizer, *Software Testing Techniques.* Van Nostrand Reinhold Co., 1990.

[7] A. Bertolino, "Software Testing Research: Achievements, Challenges, Dreams," *Proc. 2007 Int'l Conf. Future of Software Eng.,* 2007.

[8] L.C. Briand, Y. Labiche, and M. Shousha, "Stress Testing Real-Time Systems with Genetic Algorithms," *Proc. Genetic and Evolutionary Computation Conf.,* 2005.

[9] 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.

[10] P.M.S. Bueno and M. Jino, "Identification of Potentially Infeasible Program Paths by Monitoring the Search for Test Data," *Proc. 15th IEEE Int'l Conf. Automated Software Eng.,* pp. 209-218, 2000.

[11] P.M.S. Bueno, W.E. Wong, and M. Jino, "Improving Random Test Sets Using the Diversity Oriented Test Data Generation," *Proc. Second Int'l Workshop Random Testing: Co-Located with the 22nd IEEE/ACM Int'l Conf. Automated Software Eng.,* 2007.

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

[13] K.Y. Cai and D. Card, "An Analysis of Research Topics in Software Engineering—2006," *J. Systems and Software,* vol. 81, pp. 1051-1058, 2008.

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

[15] 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 Software,* vol. 150, pp. 161-175, 2003.

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

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

[18] T. Dyba, V.B. Kampenes, and D.I.K. Sjoberg, "A Systematic Review of Statistical Power in Software Engineering Experiments," *Information and Software Technology,* vol. 48, pp. 745-755, 2006.

[19] T. Dyba, B.A. Kitchenham, and M. Jorgensen, "Evidence-Based Software Engineering for Practitioners," *IEEE Software,* vol. 22, no. 1, pp. 58-65, Jan./Feb. 2005.

[20] S.A. Ghazi and M.A. Ahmed, "Pair-Wise Test Coverage Using Genetic Algorithms," *Proc. 2003 Congress on Evolutionary Computation,* pp. 1420-1424, 2003.

[21] M. Harman, "The Current State and Future of Search Based Software Engineering," *Proc. 2007 Int'l Conf. Future of Software Eng.,* 2007.

[22] M. Harman, Y. Hassoun, K. Lakhotia, P. McMinn, and J. Wegener, "The Impact of Input Domain Reduction on Search-Based Test Data Generation," *Proc. Sixth Joint Meeting of the European Software Eng. Conf. and the ACM SIGSOFT Symp. Foundations of Software Eng.,* 2007.

[23] M. Harman and B.F. Jones, "Search-Based Software Engineering," *Information and Software Technology,* vol. 43, pp. 833-839, 2001.

[24] M. Harman and P. McMinn, "A Theoretical Empirical Analysis of Evolutionary Testing and Hill Climbing for Structural Test Data Generation," *Proc. 2007 Int'l Symp. Software Testing and Analysis,* 2007.

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

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

[27] D. Johnson, "A Theoretician's Guide to the Experimental Analysis of Algorithms," *Proc. Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges,* pp. 215-250, 2002.

[28] B.F. Jones, H.H. Sthamer, and D.E. Eyres, "Automatic Structural Testing Using Genetic Algorithms," *Software Eng. J.,* vol. 11, pp. 299-306, 1996.

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

[30] B.A. Kitchenham, "Guidelines for Performing Systematic Literature Reviews in Software Engineering," Technical Report EBSE-2007-01, 2007.

[31] B.A. Kitchenham, T. Dyba, and M. Jorgensen, "Evidence-Based Software Engineering," *Proc. 26th Int'l Conf. Software Eng.,* 2004.

[32] 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 Trans. Software Eng.,* vol. 28, no. 8, pp. 721-734, Aug. 2002.

[33] K. Lakhotia, M. Harman, and P. McMinn, "A Multi-Objective Approach to Search-Based Test Data Generation," *Proc. Genetic and Evolutionary Computation Conf.,* 2007.

[34] S. Luke, L. Panait, G. Balan, S. Paus, Z. Skolicki, E. Popovici, K. Sullivan, J. Harrison, J. Bassett, R. Hubley, and A. Chircop, "A Java-Based Evolutionary Computation Research System," George Mason University's ECLab Evolutionary Computation Laboratory, http://www.cs.gmu.edu/~eclab/projects/ecj/, 2007.

[35] T. Mantere and J.T. Alander, "Evolutionary Software Engineering, a Review," *Applied Soft Computing,* vol. 5, pp. 315-331, 2005.

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

[37] P. McMinn, "Search-Based Software Test Data Generation: A Survey," *Software Testing, Verification and Reliability,* vol. 14, pp. 105-156, 2004.

[38] C.C. Michael, G. McGraw, and M.A. Schatz, "Generating Software Test Data by Evolution," *IEEE Trans. Software Eng.,* vol. 27, no. 12, pp. 1085-1110, Dec. 2001.

[39] 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.

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

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

[42] H. Pohlheim, "GEATbx—The Genetic and Evolutionary Algorithm Toolbox for Matlab," 2007.

[43] P. Puschner and R. Nossal, "Testing the Results of Static Worst-Case Execution-Time Analysis," *Proc. 19th IEEE Real-Time Systems Symp.,* pp. 134-143, 1998.

[44] D.J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures,* third ed. Chapman & Hall/CRC, 2003.

[45] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing," *Proc. 28th Ann. Int'l Computer Software and Applications Conf.,* pp. 72-77, 2004.

[46] M. Srinivas and L.M. Patnaik, "Genetic Algorithms: A Survey," *Computer,* vol. 27, no. 6, pp. 17-26, June 1994.

[47] N. Tracey, J. Clark, and K. Mander, "Automated Program Flaw Finding Using Simulated Annealing," *Proc. ACM SIGSOFT Int'l Symp. Software Testing and Analysis,* 1998.

[48] R.H. Untch, A.J. Offutt, and M.J. Harrold, "Mutation Analysis Using Mutant Schemata," *Proc. 1993 ACM SIGSOFT Int'l Symp. Software Testing and Analysis,* 1993.

[49] S. Wappler and I. Schieferdecker, "Improving Evolutionary Class Testing in the Presence of Non-Public Methods," *Proc. 22nd IEEE/ACM Int'l Conf. Automated Software Eng.,* 2007.

[50] A. Watkins and E.M. Hufnagel, "Evolutionary Test Data Generation: A Comparison of Fitness Functions," *Software: Practice and Experience,* vol. 36, pp. 95-116, 2006.

[51] J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary Test Environment for Automatic Structural Testing," *Information and Software Technology,* vol. 43, pp. 841-854, 2001.

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

[53] 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 Eng.*, vol. 12, pp. 183-239, 2007.

[54] Y. Zhan and J.A. Clark, "Search-Based Mutation Testing for Simulink Models," *Proc. Genetic and Evolutionary Computation Conf.*, 2005.

[55] Y. Zhan and J.A. Clark, "The State Problem for Test Generation in Simulink," *Proc. Genetic and Evolutionary Computation Conf.*, 2006.

Shaukat Ali received the master's degree in systems and software engineering from Mohammad Ali Jinnah University, Islamabad, Pakistan. He is currently working toward the PhD degree at the Simula Research Laboratory and the Department of Informatics, University of Oslo, Norway. He is a former member of the following research groups: Center for Software Dependability (CSD), Islamabad, Pakistan; Verification and Testing (VT) group, the University of Sheffield, United Kingdom; Software Quality Engineering Laboratory (SQUALL), Carleton University, Canada. His research interests include modeling software systems using UML and its various extensions and model-based testing of software systems. He is a student member of the IEEE.

Lionel C. Briand is a professor of software engineering at the Simula Research Laboratory and University of Oslo, leading the project on software verification and validation. Before that, he was on the faculty of the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, where he was a full professor and held the Canada Research Chair in Software Quality Engineering. He has also been the Software Quality Engineering Department head at the Fraunhofer Institute for Experimental Software Engineering, Germany, and worked as a research scientist for the Software Engineering Laboratory, a consortium of the NASA Goddard Space Flight Center, CSC, and the University of Maryland. He has been on the program, steering, or organization committees of many international IEEE and ACM conferences. He is the editor-in-chief of the *Empirical Software Engineering* (Springer) and is a member of the editorial boards of *Systems and Software Modeling* (Springer) and *Software Testing, Verification, and Reliability* (Wiley). He was on the board of the *IEEE Transactions on Software Engineering* from 2000 to 2004. His research interests include: model-driven development, testing and quality assurance, and empirical software engineering. He is a fellow of the IEEE.

Hadi Hemmati received the MEng degree in software engineering from Sharif University of Technology, Tehran, Iran. He is currently working toward the PhD degree at the Simula Research Laboratory and the Department of Informatics, University of Oslo, Norway. He has some years of industrial experience as a system analyst and software engineer in the telecommunication domain. His research interests include model-driven development, search-based software engineering, testing and quality assurance, ubiquities, and autonomic systems. He is a student member of the IEEE.

Rajwinder K. Panesar-Walawege received the MSc degree in computer science from the University of Victoria, British Columbia, Canada. She is currently working toward the PhD degree in software engineering at the Simula Research Laboratory and the Department of Informatics, University of Oslo, Norway. She has many years of industrial experience as a software engineer in the air traffic management domain, working for companies such as Raytheon Systems Limited and NAVCanada. Her research interests include model-driven development, testing and quality assurance of safety-critical systems, and empirical software engineering. She is a student member of the IEEE.