On the Impact of Tool Evolution and Case Study Size on SBSE Experiments: A Replicated Study with EvoMaster

Amid Golmohammadi$^{1[0000-0002-2324-5794]}$, Man Zhang$^{1[0000-0003-1204-9322]}$, and Andrea Arcuri$^{1,2[0000-0003-0799-2930]}$

[1] Kristiania University College, Norway
{amid.golmohammadi,man.zhang,andrea.arcuri}@kristiania.no
[2] Oslo Metropolitan University, Norway

Abstract. In the dynamic landscape of Search-Based Software Engineering (SBSE), tools and algorithms are continually improved, possibly making past experimental insights outdated. This could happen if a newly designed technique has side-effects compared to techniques and parameters settings studied in previous work. Re-tuning all possible parameters in a SBSE tool at each new scientific study would not be viable, as too expensive and too time consuming, considering there could be hundreds of them. In this paper, we carried out a series of experiments to study the impact that such re-tuning could have. For such a study, we chose the SBSE tool EvoMaster. It is an open-source tool for automated test generation for REST APIs. It has been actively developed for over six years, since November 2016, making it an appropriate choice for this kind of studies. In these experiments, we replicated four previous studies of EvoMaster with 15 REST APIs as case studies, using its latest version. Our findings reveal that updated parameter settings can offer improved performance, underscoring the possible benefits of re-tuning already existing parameters. Additionally, the inclusion of a broader range of case studies provides support for the replicated study's outcomes compared to the original studies, enhancing their external validity.

Keywords: White-Box Test Generation, SBST, RESTful APIs, Parameter Tuning, Replicating Studies

1 Introduction

In this paper, we examine the effects of tool evolution and case study size on Search-Based Software Engineering (SBSE) experiments.

The first objective of this study is to focus on the ongoing evolution of SBSE tools. We are interested in how a tool's maturity over an extended period of time might necessitate a re-examination of earlier experiments. As tools grow and integrate new techniques, there is a possibility that past studies, including how parameters were tuned, might no longer offer accurate insights. For example, if in a scientific study a new technique X is found to have best setting x', would a new study in which Y is introduced with best value y' still imply that x' is still best for X? Or maybe x'' would be a better choice when $Y = y'$? In essence, we aim to find out if the integration of new techniques into the tool, typically operated with default parameter values based on existing scientific studies, would necessitate any changes that would require re-adjusting those default settings.

The second objective aims to reduce the limitations imposed by the small size of case studies in the original, existing research work. In simpler terms, we want to test whether the conclusions of the original studies are still true when applied to a more extensive and diverse set of case studies. Threats to external validity are common to most empirical studies. In our work, we want to evaluate how serious such threats could be when replicating existing studies, but by using a larger body of artifacts for the experiments.

For the empirical experiments in this study, we use EvoMaster [9] as our subject tool. EvoMaster is a tool designed for automatically generating system-level test cases for RESTful APIs. It utilizes search-based software engineering techniques (e.g., Many Independent Objective (MIO) algorithm [5]) to optimize the generated test cases, aiming to maximize code coverage and number of found faults.

There could be many parameters and studies that could be replicated. To be able to run all the needed experiments within a viable amount of time, we selected four parameters to re-tune. The choice of these four parameters of the search algorithm was based on a review of three foundational studies on EvoMaster, spanning from 2018 to 2021 [6,19,18].

To enable such a comprehensive evaluation, we employed the latest version of EvoMaster, which was 1.6.2-SNAPSHOT at the time of conducting the experiments. In relation to the case studies used for this research, we opted for a comprehensive approach by selecting 14 different open-source REST APIs from the EMB repository [2]. This includes the case studies that were part of the original studies we replicate. Additionally, to augment the robustness of our study and make it more relevant to real-world applications, we also incorporated one industrial REST API into our set of case studies. This diverse selection aims to provide a well-rounded view of how the tool performs across a range of scenarios, thereby enhancing the external validity of our findings.

For the current study, every parameter, which is based on an earlier article, was assigned distinct values. This approach aimed to capture any emerging patterns or unique outcomes associated with the parameter variations.

The results of the varied configurations were gauged against three fundamental metrics: line coverage, which measures the extent of the codebase executed; branch coverage, measuring the percentage of decision points, such as `if` and `else` statements, that have been executed; and fault detection, indicating the capability to identify potential issues.

After the experimental phase, a comprehensive analysis ensued. This sought to identify the standout parameter configuration for each of the 15 REST APIs, assessing line and branch coverage, as well as fault detection. The findings enable us to see if previous results could be confirmed or invalidated. This could have an impact on how new experiments should be evaluated and analyzed, especially compared to all previous experiments on all the other configurations/parameters previously introduced. This type of research work deepens the knowledge on search-based software testing, possible pointing out some existing shortcomings when dealing with advanced, complex tools with many parameter settings.

2 Related Work

Parameter tuning has a critical role in the design and performance of Evolutionary Algorithms (EAs). The choice of parameter values can substantially influence the EA's performance, to the extent that an EA with well-chosen parameters can outperform one with poorly chosen parameters [11].

The study by Freisleben and Hartfelder [13] presents a method for finding the best Genetic Algorithm (GA) for a problem by treating it as an optimization problem and using another GA to solve it. It proposes a two-level optimization approach for finding the best Genetic Algorithm (GA) for a given problem. The bottom level consists of a secondary GA operating on gene strings that represent potential solutions. The top level involves a primary GA working on a population of secondary GAs represented as separate gene strings. Each secondary GA runs independently to produce a solution, and its fitness influences the primary GA's operation. The generations on the two levels are independent, and the primary GA aims to find the highest fitness string as the best GA for the original problem.

There have been multiple studies conducted to investigate the impact of parameter tuning on the performance of search-based test generation. Arcuri and Fraser [7] conducted over one million experiments in test data generation for object-oriented software using the EvoSuite tool. The results indicate that parameter tuning has an impact on search algorithm performance, but it is difficult to find settings that significantly outperform default values. This finding suggests that using default values is a reasonable choice for researchers and practitioners, as parameter tuning is a costly process with uncertain benefits.

Zamani and Hemmati [17] introduced a novel metric called *Tuning Gain* to determine the efficiency of tuning a specific class. The authors suggest predicting the Tuning Gain by analyzing the static characteristics of source code classes. Subsequently, they prioritize classes for tuning based on the estimated Tuning Gains and allocate the tuning budget specifically to the classes with higher rankings. For low-tuning budgets, it only distributes the tuning budget to a subset of classes with the greatest anticipated Tuning Gain, which increased the overall tuning outcomes by ten times.

Beyond the area of tuning, the importance of replication in search-based software engineering has also been examined. Tawosi *et al.* [16] focused on replicating and extending an existing study on CoGEE, a state-of-the-art tool for multi-objective software effort estimation. By using an independent implementation and a robust baseline, LP4EE, they enhanced the study's internal and external validity. CoGEE is also tested with four additional evolutionary algorithms and a Java framework, JMetal, to further validate its effectiveness. The results not only confirmed CoGEE's superior performance and accuracy, but also showed that its effectiveness is not dependent on a specific algorithm.

There is another study [10] which addressed the challenges of replicating and comparing computational experiments in the field of applied evolutionary computing. It highlighted the frequent lack of adequate documentation in existing studies, which made replication difficult. The paper also discussed the importance of adhering to scientific standards of experimentation and statistical precision, given the random nature of evolutionary algorithms. The primary goal was to provide guidelines to avoid common pitfalls and thereby improve the quality and reproducibility of future research in this domain.

3 Considered Parameters

Parameters in search algorithms are pivotal for optimizing performance and efficiency, affecting everything from the search space and convergence rate to runtime. Poorly chosen or untuned parameters can lead to inefficiencies, while well-selected ones can significantly improve the quality of results, making parameter tuning essential for effective algorithmic performance.

The *Many Independent Objective (MIO)* [5,6] is an evolutionary algorithm tailored based on the specific properties of test suite generation. The MIO algorithm has been employed not only by EvoMaster for test generation but also by other tools such as *EvoSuite* [12] and *Pynguin* [15]. The effectiveness and efficiency of test case generation with the MIO algorithm is controlled by particular parameters. When carefully adjusted, these variables could have a major impact on the results, ensuring a balance between the exploration and exploitation of the search landscape.

In this section, we discuss the four parameters used in EvoMaster that we chose for this study, namely: `feedbackDirectedSampling`, `probOfApply-SQLActionToCreateResources`, `probOfRandomSampling` and `focusedSearch-ActivationTime`. We could have examined more than 100 different parameters in EvoMaster, but we decided to focus on just four important ones. We picked these four because they were studied in three major research articles [6,19,18] in which EvoMaster has been extended with new techniques. This leaded us to consider these settings as they are potentially quite important for how the tool works.

Feedback-Directed Sampling (FDS) in the MIO algorithm is a technique that guides the generation of test inputs based on feedback from previous tests. Instead of randomly exploring the input space, it prioritizes areas more likely to produce valuable test cases quicker, making test generation possibly more efficient, especially when the search budget is limited. This feedback mechanism can help guide the evolution process, ensuring that the population of test cases evolves in a direction that is more likely to achieve the desired objectives, like finding faults or maximizing coverage. In a *many objective* problem (i.e., when there are several objectives to optimize for, and that are not necessarily conflicting, like in a multi-objective problem), it can make sense to first focus on the objectives that show a gradient in the fitness function. This approach also helps avoid spending excessive time on infeasible targets (as there would be no gradient there). In EvoMaster, this is controlled by the parameter `feedbackDirectedSampling`. In the original study in [6], three different values were studied: `NONE`, `LAST` and `FOCUSED`. The value `LAST` was selected as new default for EvoMaster. According to the Git history of the repository of EvoMaster, such default value has not been changed since 2017.

REST API testing is challenging, especially when creating system-level tests due to the potential complexity of their relationships with SQL databases. Endpoints are typically created in REST architectures around resources and the operations that can be performed on them. In order to improve code coverage, especially in white-box testing scenarios, it is beneficial to interact with these resources in all of their different states via the HTTP endpoints [21,22]. Moreover, resource manipulation is improved through the direct use of SQL commands such as `INSERT` [19], to create new data directly into the database. The parameter `probOfApplySQLActionToCreateResources` specifies the probability of applying such resource manipulation through the execution of SQL commands. The default value `0.5` currently in EvoMaster is based on the study in [19], which has not been changed since 2021.

The probability of applying random sampling is one of the most crucial parameters in the MIO algorithm [5]. It serves as one of the key parameters for managing the trade-off between *exploration*, which seeks out new, untested solutions, and *exploitation*, which fine-tunes known good solutions. Finding the right balance is key to the algorithm's effectiveness. Focusing too much on one can neglect the other, either missing new opportunities or failing to optimize what is already known. An excessive emphasis on randomness could make the search less effective, while a low value could restrict the exploration and put the algorithm at risk of getting stuck in local optima.

Within EvoMaster, the `probOfRandomSampling` parameter plays a pivotal role. The likelihood that EvoMaster will create a test case entirely from scratch as opposed to using pre-existing test cases or other search-based techniques is determined by this parameter. A `probOfRandomSampling` value of 0.2, for example, indicates a 20% likelihood that a new test case will be generated randomly, without relying on the knowledge gathered from previous test cases (i.e., no population in the MIO algorithm is used for mutation).

The parameter `probOfRandomSampling` was not studied on any concrete API in the articles introducing MIO [5,6]. However, due to its potential major impact on the search, it was studied in a followup work where MIO was extended with adaptive hyper-mutation [18]. Still, its default value `0.5` (which was confirmed still best in [18]) has not been changed since 2016.

The parameter `focusedSearchActivationTime` in EvoMaster denotes the percentage of the total search duration after which the tool switches from an exploratory approach to a targeted, focused search. For instance, a setting of 20% means that after completing 20% of the total search time in an exploratory manner, EvoMaster will transition to a more focused search for the remaining 80%. It is worth noting that quite a few parameters, such as `probOfRandomSampling`, decrease linearly over time until this focus search phase begins. For example, `probOfRandomSampling` goes to 0 when the focused search starts [5]. Proper tuning of this parameter is essential to ensure a balance between broad exploration and efficient, targeted searching.

Like in the case of `probOfRandomSampling`, the parameter `focusedSearch-ActivationTime` was not studied on any concrete API in the articles introducing MIO [5,6]. It was studied in the work that extended MIO with adaptive hypermutation [18]. Still, its default value `0.5` (which was confirmed still best in [18]) has not been changed since 2017.

4 Empirical Study

4.1 Research Questions

In this study, we aim at answering the following two research questions:

RQ1: How do the results of the replicated study compare to the original findings for each parameter in terms of line coverage, branch coverage, and found faults?

RQ2: What is the impact of increasing the number of case studies on the generalizability and reliability of the original study's findings?

4.2 Experiment Setup

We conducted our experiments using an existing corpus of Web/Enterprise Applications for REST API testing, namely EMB [2,4], which has been used in various recent studies (e.g., [8,14,20]). Note that all of the selected studies that we replicate [6,19,18] also employed the EMB corpus. However, over the years, the corpus has been extended with new Web APIs, i.e., from the five REST APIs[3] in the first release in 2017, to the 14 REST APIs[4] in the latest release 1.6.1 in 2023.

In our replicated study, we involved all of the 14 open-source REST APIs from EMB plus one industrial REST API. This enables a broader range of SUTs compared to the original studies we replicate [6,19,18]. To distinguish these new APIs from the ones used in the original studies, we reported our results in terms of **Original SUTs** and **Other SUTs**. Descriptive statistics of all case studies (i.e., the number of source files #SourceFiles, the line of code #LOCs, and the number of endpoints #Endpoints of each REST API) are shown in Table 1.

In the context of search-based algorithms, the term *time budget* represents a user-defined limitation on the duration for which the search process will be left running. EvoMaster utilizes evolutionary algorithms (e.g., the default is MIO [5]) to generate test cases, and these algorithms function by continuously

[3] https://github.com/EMResearch/EMB/tree/v0.1.1

[4] https://github.com/EMResearch/EMB/tree/v1.6.1

Table 1: Descriptive statistics of 15 REST APIs in our replicated study

SUT	#SourceFiles	#LOCs	#Enbdpoints
catwatch	106	9,636	14
cwa-verification	47	3,955	5
features-service	39	2,275	18
genome-nexus	405	30,004	23
gestaohospital	33	3,506	20
ind0	103	17,039	20
languagetool	1,385	174,781	2
market	124	9,861	13
ocvn	526	45,521	258
proxyprint	73	8,338	74
rest-ncs	9	605	6
rest-news	11	857	7
rest-scs	13	862	11
restcountries	24	1,977	22
scout-api	93	9,736	49
Total 15	2,991	318,953	542

refining and improving a set of solutions over numerous iterations. By setting a time budget, users provide a boundary to ensure that EvoMaster delivers results within the desired amount of time. However, there is a balancing act involved: if a user allocates a very short time budget, the tool may not have ample opportunity to explore diverse test cases, potentially resulting in tests that are not as comprehensive or effective. On the other hand, granting a longer time budget allows the evolutionary algorithm more iterations to evolve the test cases, leading to potentially higher-quality results. Still, this comes at the cost of waiting longer for the process to complete. This flexibility allows users to tailor the tool's operation based on their specific needs and constraints. For conducting this replication study, we allocated a one-hour time budget, as it is the stopping criterion used in the most recent study of EvoMaster [20].

In addition, search-based algorithms incorporate some level of randomness in their nature. Therefore, we followed the guidelines in [3] for conducting SBSE experiments, and repeated each setting of experiments 10 times. Multiple runs help to reduce the impact of randomness on the experimental results. To properly draw conclusions based on these results, we applied statistical analysis using the *Vargha-Delaney* effect size ($\hat{A}_{12}$) and the *Mann-Whitney-Wilcoxon U-test* (*p-value*) [3] for comparing the results yielded by each parameter setting.

All settings of re-tuning the four different parameters have three configurations for each parameter (recall Section 3), which resulted in $1 + (4 \times 2) = 9$ distinct configurations, i.e., the base settings plus 2 variants for each of the 4 parameters. Considering 15 REST APIs with one-hour time budget and 10 repetitions, all these experiments took over 50 days of computation time, i.e., $9 \times 15 \times 10 \times 1$h. The experiments were executed on a machine with the following specifications:

Table 2: Pair comparison of Line Coverage, Branch Coverage, and Faults achieved by different settings of Feedback-Directed Sampling (FDS). For instance, comparing x with y, $\hat{A}_{xy}$, values in **bold** mean x is statistically significant better than y, whereas values in red mean y is statistically significant better than x.

feedbackDirectedSampling A=LAST, B=FOCUSED, C=NONE									
SUT	Line Coverage			Branch Coverage			Faults		
	$\hat{A}_{AB}$	$\hat{A}_{AC}$	$\hat{A}_{BC}$	$\hat{A}_{AB}$	$\hat{A}_{AC}$	$\hat{A}_{BC}$	$\hat{A}_{AB}$	$\hat{A}_{AC}$	$\hat{A}_{BC}$
Original SUTs									
catwatch	0.49	0.58	0.59	0.51	0.62	0.60	0.55	0.56	0.51
features-service	0.48	0.48	0.49	0.56	0.53	0.47	**0.71**	**0.65**	0.48
proxyprint	0.48	0.56	0.58	0.56	0.57	0.53	0.56	0.58	0.54
rest-ncs	0.23	0.23	0.50	0.25	0.28	0.56	0.50	0.50	0.50
rest-news	0.40	0.40	0.49	0.45	0.45	0.49	0.56	0.54	0.48
rest-scs	0.23	0.16	0.34	0.30	0.23	0.38	0.23	0.23	0.50
scout-api	0.44	0.43	0.46	0.46	0.59	**0.68**	0.46	0.33	0.40
Average (original)	0.39	0.40	0.49	0.44	0.47	0.53	0.51	0.48	0.49
Median (original)	0.44	0.43	0.49	0.46	0.53	0.53	0.55	0.54	0.50
Other SUTs									
cwa-verification	0.45	0.45	0.50	0.41	0.44	0.53	0.43	0.44	0.52
genome-nexus	0.63	0.65	0.54	0.57	**0.68**	0.60	**0.94**	**0.93**	0.46
gestaohospital-rest	0.61	0.63	0.53	0.54	0.50	0.46	0.50	0.50	0.50
ind0	0.26	0.32	0.53	0.27	0.33	0.53	0.26	0.25	0.46
languagetool	0.46	0.36	0.32	0.45	0.34	0.35	0.49	0.37	0.37
market	0.48	0.59	0.60	0.47	0.60	0.61	0.52	0.46	0.44
ocvn-rest	0.57	0.51	0.44	0.60	0.54	0.44	0.19	0.40	**0.71**
restcountries	0.43	0.44	0.53	0.44	0.48	0.55	0.50	0.50	0.50
Average (other)	0.49	0.49	0.50	0.47	0.49	0.51	0.48	0.48	0.49
Median (other)	0.47	0.48	0.53	0.46	0.49	0.53	0.50	0.45	0.48
Average (all)	0.44	0.45	0.50	0.46	0.48	0.52	0.49	0.48	0.49
Median (all)	0.46	0.45	0.50	0.46	0.50	0.53	0.50	0.50	0.50

Processor Intel(R) Xeon(R) Gold 6240R CPU @2.40GHz, 2394 Mhz, 24 Core(s), 48 Logical Processor(s); RAM 192 GB; Operating System Windows 10 Pro for Workstations.

4.3 Experiment Results

Results of re-tuning the strategy of enabling Feedback-Directed Sampling. The original study [6] was published in 2018, and studied the performance of three configurations of `FDS`, i.e., `FDS` $\in$ {`NONE`, `FOCUSED`, `LAST`}. The original study showed that, in two cases (*proxyprint* and *rest-news*), `FOCUSED` provided statistically superior outcomes. Yet, in two different situations (*rest-scs* and *scout-api*), FDS yielded statistically lower coverage (i.e., `NONE` was better). The study was inconclusive regarding which value was the best, with `LAST` chosen as default based on the experiments on synthetic, artificial examples.

Table 2 represents results in our replicated study. Based on *Average* and *Median* results (see *Average (original)* and *Median (original)*) of the original case studies, `NONE` (i.e., not applying `FDS`) performed slightly better in terms of line coverage, `FOCUSED` had a slight advantage for branch coverage, and `LAST` showed a slight advantage in fault detection. A comparison of the results from the eight additional case studies (see *Average (other)* and *Median (other)*) revealed that, while there is no significant difference between the three configurations, `FOCUSED` had a slight overall advantage. It yielded slightly better branch coverage and almost the same line coverage and found faults compared to `LAST` and `NONE`. In total of 15 REST APIs, there is no clear winner among the three settings in terms of all of the three metrics, confirming the same conclusions as in the original study.

Results of re-tuning the probability of applying SQL handling to prepare resources. In the original study [19], the highest setting of `0.5` for `probOfApplySQLActionToCreateResources` yielded the best results for three projects: *rest-news*, *proxyprint*, and *scout-api*. This suggested that using just HTTP methods, like `POST` or `PUT`, to add new resources could be tricky in these cases. So, a higher chance of using SQL commands directly seemed to perform better. On the other hand, in projects like *features-service* and *catwatch*, a lower setting was better (i.e., `0.1` and `0.3` respectively). Based on these results, the authors of EvoMaster chose `0.5` as the default setting for this parameter.

In the current study, we carried out a replication of the original study to compare results and draw more robust conclusions. As it is shown in Table 3, in terms of achieving the highest code coverage, the findings are mixed. While a setting of `0.5` delivered the best coverage for *rest-news*, it was not the optimal choice for *proxy-print* and *scout-api*. These two SUTs performed best with settings of `0.3` and `0.1`, respectively. Interestingly, this diverges from the results of the original study where `0.5` was optimal across the board. For *feature-service* and *catwatch*, the setting of `0.5`, which was not the best choice in the original study, actually produced the highest code coverage in our replicated study. On average, when we looked at the five SUTs featured in the original study, a setting of `0.5` proved to deliver the best code coverage. Another intriguing observation was related to the detection of faults. A setting of `0.1` surprisingly led to the identification of the highest number of faults, which merits further investigation.

Furthermore, when we expanded our scope to include an additional 10 SUTs, we found `0.5` yielded slightly better line and branch coverage. Nonetheless, the count of detected faults remained unchanged.

Regarding results of all of the 15 REST APIs, the setting of `0.5` achieved the overall best performance, that is consistent with the original study.

Results of re-tuning the probability of applying random sampling. In the original study [18], two distinct parameters were examined: `probOfRandomSampli-ng` and `focusedSearchActivationTime`. We aimed to extend and replicate this study by incorporating eight additional case studies and adjusting these two parameters. For our research, we approached these parameters as independent entities and assessed each in isolation. The primary research indicated that a value of `0.5` was optimal for both parameters. However, our expanded investigation revealed divergent findings.

Table 3: Pair comparison of Line Coverage, Branch Coverage, and Faults achieved by different configurations of probability of applying SQL handling to prepare resources. For instance, comparing x with y, $\hat{A}_{xy}$, values in **bold** mean x is statistically significant better than y, whereas values in red mean y is statistically significant better than x

probOfApplySQLActionToCreateResources A=0.5, B=0.1, C=0.3									
SUT	Line Coverage			Branch Coverage			Faults		
	$\hat{A}_{AB}$	$\hat{A}_{AC}$	$\hat{A}_{BC}$	$\hat{A}_{AB}$	$\hat{A}_{AC}$	$\hat{A}_{BC}$	$\hat{A}_{AB}$	$\hat{A}_{AC}$	$\hat{A}_{BC}$
Original SUTs									
catwatch	0.53	0.65	0.60	0.53	**0.69**	0.64	0.40	0.64	**0.73**
features-service	0.54	0.53	0.47	0.54	0.59	0.54	0.39	0.52	0.63
proxyprint	0.46	0.38	0.45	0.47	0.45	0.49	0.47	0.44	0.46
rest-news	**0.70**	0.60	0.39	**0.77**	0.63	0.35	0.62	0.44	0.33
scout-api	0.48	0.54	0.54	0.43	0.48	0.54	0.43	0.49	0.57
Average (original)	0.54	0.54	0.49	0.55	0.57	0.51	0.46	0.50	0.54
Median (original)	0.53	0.54	0.47	0.53	0.59	0.54	0.43	0.49	0.57
Other SUTs									
cwa-verification	0.47	0.47	0.50	0.45	0.51	0.56	0.47	0.41	0.44
genome-nexus	0.51	0.54	0.54	0.46	0.50	0.57	0.53	0.50	0.48
gestaohospital-rest	0.62	0.62	0.51	0.51	0.50	0.50	0.50	0.50	0.50
ind0	0.45	0.40	0.42	0.50	0.43	0.42	0.55	0.53	0.48
languagetool	0.38	0.46	0.59	0.36	0.44	0.58	0.44	0.40	0.46
market	0.56	0.59	0.51	0.56	0.59	0.51	0.54	0.44	0.41
ocvn-rest	0.42	0.53	0.61	0.46	0.56	0.58	0.48	0.54	0.55
rest-ncs	0.45	0.43	0.46	0.46	0.46	0.48	0.50	0.50	0.50
rest-scs	0.57	0.57	0.50	0.58	0.57	0.51	0.58	0.58	0.51
restcountries	0.53	0.60	0.55	0.53	0.59	0.55	0.50	0.50	0.50
Average (other)	0.50	0.52	0.52	0.49	0.51	0.53	0.51	0.49	0.48
Median (other)	0.49	0.54	0.51	0.48	0.51	0.53	0.50	0.50	0.49
Average (all)	0.51	0.53	0.51	0.51	0.53	0.52	0.49	0.49	0.50
Median (all)	0.51	0.54	0.51	0.50	0.51	0.54	0.50	0.50	0.50

Regarding the `probOfRandomSampling` parameter, a glance at the original SUTs displayed in Table 4 shows that a value of `0.8` yielded the most favorable results (see *Average (original)* and *Median (original)*). Based on the results obtained by the additional eight SUTs, all of the three settings achieved similar results in terms of line coverage and branch coverage. Regarding fault detection, we observed one significant outperformance achieved by the setting of `0.8` on *ocvn-rest*, while the difference is not significant on the other seven case studies. However, based on results of all case studies in terms of all three metrics, the advantage of `0.8` over `0.5` is modest.

Results of re-tuning the time of activating focused search. As we assess the `focusedSearchActivationTime` parameter from Table 5, we note that while a value of `0.5` seemed significantly more effective for certain SUTs, such as *catwatch* and *feature-service*, a broader perspective considering other SUTs and the number of detected faults suggests a different story. Contrary to the findings of the original study [18], a value of `0.5` did not outperform others. Specifically,

Table 4: Pair comparison of Line Coverage, Branch Coverage, and Faults achieved by different settings of probability of applying random sampling. For instance, comparing x with y, $\hat{A}_{xy}$, values in **bold** mean x is statistically significant better than y, whereas values in red mean y is statistically significant better than x.

	probOfRandomSampling A=0.5, B=0.2, C=0.8								
SUT	Line Coverage			Branch Coverage			Faults		
	$\hat{A}_{AB}$	$\hat{A}_{AC}$	$\hat{A}_{BC}$	$\hat{A}_{AB}$	$\hat{A}_{AC}$	$\hat{A}_{BC}$	$\hat{A}_{AB}$	$\hat{A}_{AC}$	$\hat{A}_{BC}$
Original SUTs									
rest-ncs	0.52	0.48	0.48	0.52	0.51	0.50	0.50	0.55	0.55
rest-scs	0.60	0.50	0.43	0.63	0.47	0.37	0.60	0.47	0.38
rest-news	0.56	0.52	0.43	**0.71**	0.33	0.17	0.58	0.45	0.38
catwatch	**0.75**	0.40	0.16	**0.80**	0.41	0.11	**0.67**	0.44	0.27
features-service	**0.68**	0.57	0.35	0.57	0.64	0.58	0.51	0.45	0.45
proxyprint	0.64	0.38	0.23	0.60	0.45	0.36	0.49	0.37	0.35
scout-api	0.56	0.56	0.50	0.51	0.56	0.55	0.57	0.49	0.38
Average (original)	0.62	0.49	0.37	0.62	0.48	0.38	0.56	0.46	0.39
Median (original)	0.60	0.50	0.43	0.60	0.47	0.37	0.57	0.45	0.38
Other SUTs									
cwa-verification	0.43	0.44	0.52	0.47	0.45	0.50	0.47	0.46	0.49
genome-nexus	0.52	0.48	0.44	0.55	0.44	0.40	0.49	0.49	0.51
gestaohospital-rest	0.62	**0.66**	0.54	0.52	0.52	0.50	0.52	0.50	0.48
ind0	0.31	0.50	**0.70**	0.35	0.59	**0.72**	0.49	0.57	0.55
languagetool	0.47	0.41	0.45	0.44	0.41	0.47	0.52	0.50	0.48
market	0.46	0.52	0.56	0.47	0.52	0.56	0.49	0.48	0.49
ocvn-rest	0.54	0.47	0.45	0.53	0.50	0.48	**0.98**	0.00	0.00
restcountries	0.57	0.55	0.48	0.56	0.55	0.49	0.50	0.50	0.50
Average (other)	0.49	0.51	0.52	0.49	0.50	0.51	0.56	0.44	0.44
Median (other)	0.50	0.49	0.50	0.50	0.51	0.49	0.50	0.50	0.49
Average (all)	0.55	0.50	0.45	0.55	0.49	0.45	0.56	0.45	0.42
Median (all)	0.56	0.50	0.45	0.53	0.50	0.49	0.51	0.48	0.48

it resulted in lower code coverage and found fewer faults compared to `0.8`. This observation was consistent across the additional case studies.

4.4 Findings Summary

Results for RQ1. In the case of our first replicated study, focusing on the `FDS` parameter, the outcomes revealed a slight advantage of `FOCUSED` had a minor advantage in branch coverage but no configuration was a clear winner in all metrics, reaffirming the conclusions of the original study.

Our second replicated study looked into the `probOfApplySQLActionToCreateResources` parameter. Here, we found a consistent pattern with the original study: a setting of `0.5` did lead to better line and branch coverage. The number of detected faults were consistent with the original study, reaffirming its initial findings.

For the third and fourth replicated studies, which investigated the `probOfRandomSampling` and `focusedSearchActivationTime` parameters respectively, our results took a different turn. Unlike the original studies that found `0.5` as the optimal setting, our study showed that a `0.8` setting yielded better results across various APIs.

Table 5: Pair comparison of Line Coverage, Branch Coverage, and Faults achieved by different settings of time of activating focused search For instance, comparing x with y, $\hat{A}_{xy}$, values in **bold** mean x is statistically significant better than y, whereas values in red mean y is statistically significant better than x.

SUT	focusedSearchActivationTime A=0.5, B=0.2, C=0.8								
	Line Coverage			Branch Coverage			Faults		
	$\hat{A}_{AB}$	$\hat{A}_{AC}$	$\hat{A}_{BC}$	$\hat{A}_{AB}$	$\hat{A}_{AC}$	$\hat{A}_{BC}$	$\hat{A}_{AB}$	$\hat{A}_{AC}$	$\hat{A}_{BC}$
Original SUTs									
rest-ncs	0.49	0.48	0.50	0.51	0.47	0.47	0.52	0.50	0.48
rest-scs	0.54	0.60	0.56	0.57	0.59	0.53	0.53	0.55	0.52
rest-news	0.62	0.55	0.42	0.52	0.49	0.47	**0.67**	0.48	0.32
catwatch	**0.67**	0.42	0.29	**0.74**	0.45	0.20	**0.73**	0.38	0.16
features-service	0.60	0.47	0.35	0.57	0.44	0.36	**0.70**	0.49	0.30
proxyprint	**0.66**	0.29	0.17	**0.67**	0.35	0.20	0.44	0.40	0.44
scout-api	0.57	0.34	0.25	0.50	0.29	0.26	**0.65**	0.36	0.17
Average (original)	0.59	0.45	0.36	0.58	0.44	0.36	0.61	0.45	0.34
Median (original)	0.60	0.47	0.35	0.57	0.45	0.36	0.65	0.48	0.32
Other SUTs									
cwa-verification	0.48	0.41	0.43	0.49	0.43	0.45	0.55	0.44	0.39
genome-nexus	**0.71**	0.51	0.27	**0.67**	0.50	0.28	0.58	0.53	0.44
gestaohospital-rest	**0.72**	0.59	0.37	0.52	0.52	0.50	0.50	0.50	0.50
ind0	0.57	0.38	0.32	0.64	0.43	0.30	**0.65**	0.52	0.37
languagetool	0.51	0.47	0.44	0.48	0.46	0.48	0.49	0.39	0.40
market	0.55	0.55	0.51	0.54	0.55	0.51	**0.65**	0.39	0.26
ocvn-rest	0.60	0.45	0.36	0.60	0.56	0.46	**0.99**	0.03	0.00
restcountries	0.62	0.61	0.47	0.60	0.58	0.48	0.50	0.50	0.50
Average (other)	0.59	0.50	0.40	0.57	0.50	0.43	0.61	0.41	0.36
Median (other)	0.58	0.49	0.40	0.57	0.51	0.47	0.56	0.47	0.40
Average (all)	0.59	0.48	0.38	0.57	0.47	0.40	0.61	0.43	0.35
Median (all)	0.60	0.47	0.37	0.57	0.47	0.46	0.58	0.48	0.39

RQ1: While certain initial parameter suggestions remain valid, especially in the second study, other results indicate the necessity to reassess them as the EvoMaster *tool continues to be developed and evolved.*

Results for RQ2. By examining the results across the four replicated studies, there was a consistent pattern observed. The results derived from the original case studies were generally affirmed by the newly added case studies introduced in our research. For instance, while there was an anomaly in the second replicated study where the parameter setting of `0.1` detected slightly more faults for the original case studies, this distinction was neutralized when taking into account the results from the expanded set of SUTs.

RQ2: The inclusion of additional case studies in our replicated research generally confirmed the findings from the original studies' SUTs, enhancing the external validity of the results.

5 Threats to Validity

Conclusion Validity. To take into account the randomness of SBST techniques, our experiments were repeated 10 times. In order to draw reliable conclusions, we used statistical analysis techniques, including Mann-Whitney-Wilcoxon U-tests (*p-value*) and Vargha-Delaney effect sizes ($\hat{A}_{12}$), following common guidelines in software engineering research [3].

Internal Validity. The implementation of EvoMaster and the case studies in EMB, except one industrial API, are open-source and available online. As the authors of EvoMaster, we provide as well all the scripts used to setup experiments in most of our previous work. To reduce threats of running EvoMaster differently, we re-used those scripts as starting point to setup our experiments. We have published those modified scripts online on EvoMaster's GitHub repository [1]. This allows anyone to review and replicate this study.

External Validity. It is important to acknowledge that our study is specifically tailored to evolutionary algorithms in the domain of search-based software testing. Moreover, our case studies exclusively involved REST APIs and the tool EvoMaster. Our findings in this study may not be directly generalizable to other types of software, tools, algorithms, or problem domains. However, we incorporated all the JVM (i.e., Java/Kotlin) REST APIs in the EMB corpus for Web/Enterprise API testing research and an industrial REST API to ensure having a broad range of case studies (i.e., 15 REST APIs, 318k LOC and 542 endpoints). Furthermore, different results might have been reached if a selection of different parameters was done. Due to the high cost of running experiments on system test generation (50 days in our case, if experiments are run in sequence), our research concentrated on a restricted set of parameters, leaving potential for other parameter effects to be investigated in future research. Also, we only considered one value for the search budget, i.e., 1 hour. Different search budgets would have different impacts on tuning. For example, a "short" search budget might reward settings that put more emphasis on the "exploitation" of the search landscape, in contrast to a wider "exploration" of the search landscape which would require more time.

6 Conclusion

Our study had two primary goals: first, to re-evaluate the effectiveness and parameter tuning of EvoMaster with respect to its ongoing development; and second to determine whether the findings of the original studies still apply when a larger set of SUTs are used for the experiments.

Our results show that some of the original parameter suggestions still hold true, but other choices might need to be re-evaluated. This could also underline the importance of periodic retuning for SBSE tools such as EvoMaster. The inclusion of additional case studies strengthened the external validity and robustness of these results, by mostly confirming the conclusions from the original studies (i.e., case studies in the upper section of each table in Section 4).

In this study, we have identified new optimal values for certain parameters, such as `0.8` in contrast to current `0.5` for Probability of Random Sampling. However, one limitation is that we have not yet looked into how these values would perform when combined. It is difficult to predict how various parameter combinations can interact due to the complexity of SBST and mature tools such as EvoMaster. We need to conduct more studies to fill this important knowledge gap and see whether combining these parameters could have any unanticipated side-effects or emergent behaviors. Also, there is the need to study in more details how the search budget impacts parameter tuning. But, to be able to do it properly, studies will be needed to identify what are the common values for the search budgets selected by practitioners in industry that use SBSE tools.

This type of studies could also be beneficial in other mature SBSE tools such as EvoSuite and Pynguin.

Acknowledgment

This work is funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (EAST project, grant agreement No. 864972).

References

1. EvoMaster. https://github.com/EMResearch/EvoMaster
2. Evomaster benchmark (emb). `https://github.com/EMResearch/EMB`, online, Accessed November 21, 2023
3. Arcuri, A., Briand, L.: A Hitchhiker's Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering. Software Testing, Verification and Reliability (STVR) **24**(3), 219--250 (2014)
4. Arcuri, A., Zhang, M., Golmohammadi, A., Belhadi, A., Galeotti, J.P., Marculescu, B., Susruthan, S.: Emb: A curated corpus of web/enterprise applications and library support for software testing research. In: IEEE International Conference on Software Testing, Verification and Validation (ICST). IEEE (2023)
5. Arcuri, A.: Many Independent Objective (MIO) Algorithm for Test Suite Generation. In: International Symposium on Search Based Software Engineering (SSBSE). pp. 3--17 (2017)
6. Arcuri, A.: Test suite generation with the Many Independent Objective (MIO) algorithm. Information and Software Technology **104**, 195--206 (2018)
7. Arcuri, A., Fraser, G.: Parameter tuning or default values? an empirical investigation in search-based software engineering. Empirical Software Engineering (EMSE) **18**(3), 594--623 (2013)
8. Arcuri, A., Galeotti, J.P.: Enhancing Search-based Testing with Testability Transformations for Existing APIs. ACM Transactions on Software Engineering and Methodology (TOSEM) **31**(1), 1--34 (2021)
9. Arcuri, A., Galeotti, J.P., Marculescu, B., Zhang, M.: Evomaster: A search-based system test generation tool. Journal of Open Source Software **6**(57), 2153 (2021)
10. Črepinšek, M., Liu, S.H., Mernik, M.: Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guidelines to avoid them. Applied Soft Computing **19**, 161--170 (2014)

11. Eiben, A.E., Smit, S.K.: Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm and Evolutionary Computation **1**(1), 19--31 (2011)
12. Fraser, G., Arcuri, A.: EvoSuite: automatic generation for object-oriented software. In: ACM Symposium on the Foundations of Software Engineering (FSE). pp. 416--419 (2011)
13. Freisleben, B., Härtfelder, M.: Optimization of genetic algorithms by genetic algorithms. In: Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Innsbruck, Austria, 1993. pp. 392--399. Springer (1993)
14. Kim, M., Xin, Q., Sinha, S., Orso, A.: Automated test generation for rest apis: No time to rest yet. In: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. p. 289–301. ISSTA 2022, Association for Computing Machinery, New York, NY, USA (2022). `https://doi.org/10.1145/3533767.3534401`, `https://doi.org/10.1145/3533767.3534401`
15. Lukasczyk, S., Fraser, G.: Pynguin: Automated unit test generation for python. In: Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings. pp. 168--172 (2022)
16. Tawosi, V., Sarro, F., Petrozziello, A., Harman, M.: Multi-objective software effort estimation: A replication study. IEEE Transactions on Software Engineering **48**(8), 3185--3205 (2021)
17. Zamani, S., Hemmati, H.: A cost-effective approach for hyper-parameter tuning in search-based test case generation. In: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). pp. 418--429. IEEE (2020)
18. Zhang, M., Arcuri, A.: Adaptive hypermutation for search-based system test generation: A study on rest apis with evomaster. ACM Transactions on Software Engineering and Methodology (TOSEM) **31**(1) (2021)
19. Zhang, M., Arcuri, A.: Enhancing resource-based test case generation for restful apis with sql handling. In: International Symposium on Search Based Software Engineering. pp. 103--117. Springer (2021)
20. Zhang, M., Arcuri, A.: Open problems in fuzzing restful apis: A comparison of tools. ACM Transactions on Software Engineering and Methodology (TOSEM) (may 2023). `https://doi.org/10.1145/3597205`, `https://doi.org/10.1145/3597205`, just Accepted
21. Zhang, M., Marculescu, B., Arcuri, A.: Resource-based test case generation for restful web services. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 1426--1434 (2019)
22. Zhang, M., Marculescu, B., Arcuri, A.: Resource and dependency based test case generation for restful web services. Empirical Software Engineering **26**(4), 1--61 (2021)