Empirical Evaluation of Frequency Based Statistical Models for Estimating Killable Mutants

Anonymous Author(s)

ABSTRACT

Estimating residual risk, i.e., the upper-bound on the number of faults that are likely to remain in the software after a testing campaign, helps in addressing important questions about software quality, (e.g., *"How much can we trust a software after testing it?"*), and automatic test generation (e.g., *"What fraction of possible faults can it detect?"*). Traditionally, those questions are addressed using structural coverage metrics; for instance during fuzzing, frequency based statistical models for unknown software species have been used for estimating *unseen but reachable coverage*, a proxy for residual risk.

Mutation analysis with its ability to induce and evaluate faults in structural elements of the code can provide a more reliable answer than structural coverage and, provided one can accurately estimate the *number of live but killable mutants*, a more representative proxy for residual risk than unseen but reachable coverage.

In this paper, we report the results of a large-scale empirical study on the application of twelve widely known frequency based statistical models for estimating the number of killable mutants in ten mature software projects. Given the closeness of mutation analysis to coverage, we postulated that the frequency based statistical models used for estimating reachable coverage could also estimate the number of killable mutants. However, as the results of our investigation suggest, the considered statistical models lack sufficient predictive power and cannot produce reliable estimates of killable mutants that are program and test suite dependent.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

mutation analysis, equivalent mutants, residual risk

ACM Reference Format:

Anonymous Author(s). 2018. Empirical Evaluation of Frequency Based Statistical Models for Estimating Killable Mutants. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 16 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

```
def determinant(a,b,c,d):          # Tests
  ad = a * d
  bc = b * c                       ⊢ determinant(1,2,1,2) = 0
  return ad - bc                   ⊢ determinant(1,2,3,4) = -2
```

Figure 1: A simple program (left) and two of its tests (right)

```
def determinant(a,b,c,d):          def determinant(a,b,c,d):
- ad = a * d                         ad = a * d
+ ad = a / d                       - bc = b * c
  bc = b * c                       + bc = c * b
  return ad - bc                     return ad - bc
```

Figure 2: A killable mutant (left) and an equivalent one (right)

1 INTRODUCTION

Mutation analysis the premier means of assessing the quality of software test suites [47] in preventing defects. Evaluating a test suite with mutation analysis involves generating mutants *exhaustively* and evaluating them against the test cases comprising that test suite (Fig. 1-right). Specifically, mutants are copies of the original code (Fig. 1-left) in which artificial, but plausible faults that share strong similarities with real faults are injected (Fig. 2) [3, 4, 21, 34].

A mutant that exhibits a detectable change in behavior of the program when exercised by a test case is said to be detected, or *killed*, by that test case. Conversely, we call mutants undetected by the test suite *surviving* mutants. The ratio of the number of mutants killed by a test suite to the number of *killable* mutants is called *mutation score* and is considered a good indicator of that test suite effectiveness in preventing faults [33]. Surviving killable mutants can be also considered as a reasonable proxy for residual risk [31] because they represent potential bugs that would have not been discovered by the test suite.

Killable Mutants = **Total Mutants** - **Equivalent Mutants** (1)

Unfortunately, not all the surviving mutants could be killed. For instance, the so-called *equivalent mutants* (see Fig. 2-right for an example) implement behaviors equivalent to the original code and cannot be killed no matter the test input used [11]. The amount of equivalent mutants, which is generally program specific [30, 42], directly determines the number of killable mutants (see Equation (1)) but cannot be established *a-priori* because there is no means to automatically prove programs equivalence in general [11, 50]. Consequently, the mutation score and the estimate of residual risk might be inaccurate and affected by high variability [29].

> An accurate estimate of the killable mutants is important for evaluating the **effectiveness of test suites** as well as evaluating **residual risk** after a test run, both of which are important problems for a practitioner.

1.1 Motivation and Main Contributions

Recently, researchers proposed the Software Testing as Species Discovery (STADS) framework [8] as a general framework for estimating *unseen* software elements that fulfill a property when only a fraction of those software elements has been discovered. For instance, STADS is used for estimating unseen, but reachable, coverage target [60] during fuzzing.

The STADS framework is based on an extension of Good-Turing estimator called Chao's estimator [16] that was used to crack the Enigma code during second world war [60]. At its core lies the idea that the relative frequency of observation of software elements such as covered structural elements and observed defects contains information about the number of similar software elements yet to be observed. Frequency based estimators such as Chao's are claimed to be robust toward strong biases in sampling [17] and are commonly used in ecology to estimate the count of various species, which are often highly correlated and geography specific.

While STADS was initially proposed for estimating reachable coverage, we argue that the same model is applicable for estimating surviving killable mutants because covering a structural element is a prerequisite for detecting any mutation in that element. In the light of this observation, we investigated whether using frequency based statistical models, that is, using the frequency count of known elements to estimate unknown elements, can predict the number of killable mutants from killed ones. Specifically, we conducted a large empirical study on how well twelve statistical estimators from the literature predict killable mutants generated by the state-of-art mutation testing framework PIT [20] for ten mature, well-tested open-source projects. This paper summarizes the main findings of our study and is complemented by the replication package [1].

> Our large study empirically investigates the hypothesis that (at least) one of the surveyed frequency based estimators can be effectively used to estimate killable mutants. That is, it is accurate, reliable, and produces better estimates than the naïve upper bound of total number of generated mutants.

1.2 Empirical Study Results Overview

After selecting well maintained, open-source projects and analyzing them using PIT, we manually classified 1016 live mutants that survived existing test suites and used this result for assessing the prediction quality of a large selection of frequency based statistical estimators (see Section 2.2). However, we found that the estimates from the selected estimators were significantly different from the manual estimates. Hence, we questioned our experimental settings and identified two major threats to the validity of our study: (1) the manual classification of live mutants could be in error; hence, the estimators might be closer to the truth than the estimators derived from manual classification, or (2) the tests themselves could have been biased, with more effective tests checking only specific parts of the code (e.g., business logic). Such biases might affect the frequency of killed mutants and might lead to faulty estimates.

To mitigate the first threat, we extended our initial study with test suites created with EvoSuite [24], a state-of-the-art test generation tool, using two fundamentally different configurations (i.e., a random strategy and guided by coverage). We expected the estimations obtained from manual test suites to coincide with estimations from EvoSuite test suites as they predict the same quantity. To mitigate second threat, we compared the estimates from EvoSuite test suites and from the manual classification. We expected the estimates from EvoSuite to match estimates from manually classified live mutants as EvoSuite test suites are free of manual bias.

We, again, found that manual classification was significantly different from estimations from automatically generated test suites, and the estimates from the automatically generated test suites were significantly different from each other. Therefore, we critically inspected our experiment and identified another possible source of error–the possibility that different test objectives in generating different test suites may somehow lead to different detectable mutants.

To check this, we changed the sampling unit used. Since we obtained first estimates by considering test methods as test cases, we next considered test classes as test cases. Because both test methods and test classes sample the same quantity, i.e., the killed mutants, we expected that the statistical estimators produce consistent estimates, although possibly with different degrees of uncertainty.

Once again, our results showed that the estimates from test methods and test classes were significantly different, which left us with the only option that the considered frequency based statistical estimators cannot be reliably applied to mutation analysis. That is, our results show that none of the considered statistical estimators behaved consistently, and none of them is robust to changes in the sampling strategy. For instance, we found the estimators changed their predictions significantly when using different sampling methods and, almost all the times test class based sampling let to predict more killable mutants than the existing ones. Additionally, we found that in several cases the results are program dependent, and that the predictions from manual classifications and statistical estimators agreed ($< 1\%$ difference) only when there were exceptionally strong test suites (i.e., $> 99\%$ killed mutants).

1.3 Summary of Contributions

In summary, our main contributions are:

- The first empirical study on the application of twelve, widely used frequency based statistical estimators applied to the problem of estimating killable mutants, showing that they are not yet ready for the prime time.
- One of the largest mutation analysis datasets containing 1016 live mutants manually classified by three researchers on ten mature projects, multiple test suites generated manually and automatically, and their mutation analysis resulting in more than 2.5B test executions.

2 STATISTICAL FRAMEWORK

We wish to measure the number of killable mutants with help of statistical estimators from biometrics. These estimators, which have been used to estimate the size of unknown populations, have also been successful in counting systems in computer networks [**?**] and estimating residual defects in the STADS framework [9].

In the next sections, we first introduce the underlying statistical model and, then, we describe the various statistical estimators that we considered in our study.

2.1 The Urn Statistical Model

At first, let's consider the following well known statistical model: Suppose, we have an urn with colored balls, from which n balls are sampled with replacement and let $S(n)$ the colors we observed. Furthermore, let's assume that each ball has multiple colors. Now, we are interested in how many colors S this urn can contain. In application to our problem, this urn sampling, which is described with the Bernoulli product model, associates each test (a ball) with the killed mutants (the colors of that ball). We use the standard definitions from the Bernoulli product described in the STADS framework to formalize the intuition behind the urn model.

Let $\mathcal{P}$ be the program under test and $\mathcal{D}$ the set of all tests X that exercise $\mathcal{P}$. We can model a testing campaign $\mathcal{T}$ as a stochastic process

$$\mathcal{T} = \{X_n | X_n \in \mathcal{D}\}_{n=1}^{T}$$

where T tests are sampled with replacement from $\mathcal{D}$. Let $\{M_i\}_{i=1}^{S}$ be a set of mutants. Mutant M_i can be detected with a probability π_i that might be affected by factors specific to M_i's definition (e.g., the mutation operators that generated it or the location in the code where it is applied). In the Bernoulli product model, a test can kill one or more mutants. For a testing campaign of size T, we let the *kill incidence* matrix, or simply killmatrix, $W_{S\times T}$ be defined as

$$W_{S\times T} = \left\{W_{ij} | i = 1, 2, \ldots, S \wedge j = 1, 2, \ldots, T\right\},$$

where $W_{ij} = 1$ if test X_j kills mutant M_i and $W_{ij} = 0$ otherwise.

This way, the i_{th}-row sum of W ($Y_i = \sum_{j=1}^{T} W_{i,j}$) denotes the incidence-based frequency of mutant M_i (being killed). We further define the incidence frequency counts Q_k, where $0 \leq k \leq T$, as the number of mutants killed by exactly k tests. Consequently, the *unobservable* frequency count Q_0 denotes the number of mutants that remain undetected.

We assume that the probability that a mutant M_i is detected by a test X_j is defined as $P(W_{i,j} = 1) = \pi_i \cdot \nu_j$, where variables $\{\nu_1, \nu_2, \cdots, \nu_T\}$ are responsible for test effects. Indeed, the ability of a test to kill mutants might be affected by various factors, such as coverage, input data, environment or flakiness. We model those test effects as a random variable from an unknown probability density function $h(\nu)$, whereas we assume fixed mutant detection rates π_i. Hence, we model probability distribution of each element W_{ij} of the killmatrix as a Bernoulli random variable conditioned on ν_j:

$$P(\forall (i,j) W_{ij} = w_{ij} \,|\, \nu_j) = (\pi_i \nu_j)^{w_{ij}} (1 - \pi_i \nu_j)^{1 - w_{ij}}$$

The probability distribution for the incidence matrix can be expressed as the probability for all $i : 1 \leq i \leq S$ and $j : 1 \leq j \leq T$ that we have $W_{ij} = w_{ij}$.

$$P\left(\forall (i,j) W_{ij} = w_{ij} | \nu_i\right) = \prod_{j=1}^{T} \prod_{i=1}^{S} \pi_i \nu_j{}^{w_{ij}} (1 - p_i)^{1 - w_{ij}}$$

Integrating out all possible values of $\{\nu_1, \nu_2, \cdots, \nu_T\}$, we obtain the unconditional marginal distribution for the incidence-based frequency Y_i for the mutant M_i, which follows a Binomial distribution:

$$\begin{aligned} P(Y_i = y_i) &= \binom{T}{y_i} \left[\pi_i \int \nu h(\nu) d\nu\right]^{y_i} \left[1 - \pi_i \int \nu h(\nu) d\nu\right]^{T - y_i} \\ &= \binom{T}{y_i} \lambda_i{}^{y}_{i} (1 - \lambda_i{}^{y}_{i})^{T - y_i}, \end{aligned}$$

where $\lambda_i = \pi_i \int \nu h(\nu) d\nu$. That is, the frequency Y_i is a binomial random variable with detection probability λ_i, and the incidence frequency counts Q_k can be derived as:

$$Q_k = E\left[\sum_{i=1}^{S} I(Y_i = k)\right] = \sum_{i=1}^{S} \binom{T}{k} \lambda_i^k (1 - \lambda_i)^{T-k}.$$

2.2 Frequency Based Estimators

Estimators that can estimate the total number of colors under Bernoulli Product model belong to the class of frequency based estimators. They have been used extensively in biometrics (Ecology) to estimate *unseen* species [17] and address questions about population size (e.g.,*"How many individuals of a particular species are in a given geographical area?"*) and species richness (e.g., *"How many species can be found in a given geographical area?"*).

The problem of estimating species richness, firstly formulated in biometrics [17], is challenging because the geographical area to consider while counting the number of species is often large, and can't be exhaustively surveyed. Hence, ecologists resort to *sampling*: (1) they divide the area into *sampling units*, and (2) randomly select a number of sampling units to survey for number of species found.

Sampling data might show different distributions of species in sampling units and might be incomplete. For instance, a certain number of species may be found in multiple sampling units, while other (rarer) species may not be found in any. Hence, species richness estimation requires to estimate the total number of species that is present in the considered geographical area including species not found in any sampling unit. The essential idea exploited by species richness estimators is that the number of species that are never sampled can be estimated by considering the next rarest species, such as species detected only once, or *singleton*, twice, or *doubleton*, and the number of species found.

Chao [17] identified two kinds of species richness estimators based on whether they adopt *incidence data*, i.e., data about the presence of species across multiple sampling units, or *abundance data*, i.e., data about the number of individuals of different species found. As our model leverages kill incidence matrix, we primarly resort to incidence sampling estimators.

Incidence sampling considers T sampling units randomly selected among all the available ones and assumes these are independent [17]. In each sampling unit, the relevant (categorical) data is the presence of various species; hence, incidence sampling data do not consider the explored size of each species. After surveying all the T sampling units, incidence sampling reports the count of species that appear only once (Q_1), twice (Q_2), and so on. Using these values, the estimators predicts Q_0, i.e., the number of species that never appeared during sampling.

2.2.1 *Chao estimators [14].*

The basic Chao estimator is:

$$\hat{S}_{Chao} = \begin{cases} S_{obs} + \frac{T-1}{T} Q_1^2/(2Q_2) & \text{if } Q_2 > 0. \\ S_{obs} + \frac{T-1}{T} Q_1(Q_1 - 1)/2 & \text{otherwise.} \end{cases}$$

S_{obs} is the observed species count, Q_1 and Q_2 the frequency of singleton and doubleton species, and T is the number of sampled units. This estimator is represented as `Chao` in the rest of the paper.

Chiu et al. [19] derived an improved version (referred to as `iChao`), which makes use of the additional information of tripletons Q_3 and quadrupletons Q_4 to estimate undetected species richness.

$$\hat{S}_{iChao} = S_{Chao} + \frac{T-3}{T}\frac{Q_3}{4Q_4} \times \max\left[Q_1 - \frac{T-3}{T-1}\frac{Q_2 Q_3}{2Q_4}, 0\right]$$

Chao estimators are known to provide *lower bounds* estimates.

2.2.2 Jackknife estimator [12, 13]. The basic, first order, Jackknife estimator is computed as follows:

$$S_{jack1} = S_{obs} + Q_1\left(\frac{T-1}{T}\right)$$

The second order Jackknife [53], which is more robust to sampling bias [32] but has larger standard error, is given by:

$$S_{jack2} = S_{obs} + \frac{2T-3}{T}Q_1 - \frac{(T-2)^2}{T(T-1)}Q_2.$$

Higher-order Jackknife estimators (S_{jackJ}) are constructed similarly. We consider up to J = 5, and automatically select the best performing one (referred as `Jackknife`). These estimators underestimate on small sample size, and overestimate otherwise [17].

2.2.3 Incidence coverage estimator [18]. The Incidence Coverage Estimator (referred as `ICE`) is given by:

$$S_{ICE} = S_{freq} + \frac{S_{infreq}}{\hat{C}_{infreq}} + \frac{Q_1}{\hat{C}_{infreq}}\hat{\gamma}^2_{infreq}, \quad \hat{C}_{infreq} = 1 - Q_1 / \sum_{i=1}^{k} Q_i,$$

$$\hat{\gamma}^2_{infreq} = \max\left[\frac{S_{infreq}}{\hat{C}_{infreq}}\frac{T}{T-1}\frac{\sum_{i=1}^{k} i(i-1)Q_i}{\left(\sum_{i=1}^{k} iQ_i\right)\left(\sum_{i=1}^{k} iQ_i - 1\right)} - 1, 0\right]$$

S_{freq} is the number of species that occur more than k times, while S_{infreq} is those that do. A $k = 10$ cut-off is recommended. We also considered the original coverage estimator as firstly proposed in [35], which is equivalent to the `ICE` with $k = 0$ (`ICE-k0`). Gotelli [28] provided a version (referred as `ICE-1`) for highly heterogeneous communities, which underestimates less.

For all the aforementioned estimators, there is an analytical way of constructing the confidence interval.

2.2.4 Bootstrap estimator [53]. The Bootstrap estimator is based on resampling T sampling units with replacement from a set of initially observed T sampling units a sufficient number of times m.

$$S_{bootstrap} = avg_m(S_{obs} + \sum_{j=1}^{S_{obs}}(1 - Y_j/T)^T)$$

Y_j is the number of sampling units where species j is detected.

2.2.5 Zelterman estimator [10]. Zelterman's estimator (referred as `Zelterman`) is frequently used in Social Sciences, in particular in illicit dug use research. It is defined as:

$$S_{zelterman} = S_{obs} + S_{obs}/[(1+\lambda)^T - 1]$$

where $\hat{\lambda} = 2f_2/(m-1)f_1$.

2.2.6 Other estimators. The estimators mentioned below are not intended to be directly applied to our model. Still, with the slight adaptation they can be used for our purposes. They have have complex formulation; hence, we only introduce them.

Along with the incidence based estimators, there is also a class of species richness estimators that work with abundance data. Abundance sampling considers each individual to be a sampling unit and recognizes as the relevant (numerical) data the number of individuals that belong to each species. We can transform our data into abundance form by summing up values of the incidence matrix along the test axis, though, loosing information about the number of samples (tests). This way, we get a vector of counts for each mutant. Besides, sometimes estimates assume Poisson rather than Binomial distribution of counts, which can be considered the limiting case when the number of tests tends to infinity (i.e., is too big). Therefore, we additionally analyse `Chao Bunge`, `UNPMLE`, `PNPMLE`, and `PCG` estimators. The Chao Bunge estimator, proposed by Chao and Bunge [15], utilizes the gamma-Poisson model in which species are detected in the sample according to a Poisson process and rates of the processes follow a gamma distribution. The Unconditional nonparametric maximum likelihood estimator (UNPMLE) was provided by Norris and Pollock [41]. The Penalized nonparametric maximum likelihood estimator (PNPMLE) was provided by Wang and Lindsay [58]. The Poisson-compound Gamma model with smooth nonparametric maximum likelihood estimation (PCG) was provided by Wang and Ji-Ping [57].

3 METHODOLOGY

Our methodology consisted of the following steps: (1) test subjects selection (Section 3.1); (2) test suite generation (Section 3.2); and, (3) killable mutants estimation (Section 3.3). After estimating the killable mutants for each test suite, sampling strategy, and selected project, we needed ground truth data about killable mutants to assess the estimation accuracy. Since such data was not available, we collected it by (4) collecting survived mutants (Section 3.4) and (5) manually classify them as killable or equivalent (Section 3.5).

3.1 Test Subjects Selection

The choice of our test subjects was guided by the following considerations: (1) We consider manual mutant classification as the most important part of our study; hence, we wanted to ensure that our classifiers could understand easily what a program fragment does, thus closely matching real world settings, in which programmers work on well understood projects. (2) We wanted to reduce the work involved in setting up our experiments on different computing infrastructures, thus increasing the reproducibility of our results. Hence, we focused on large open-source *Java* projects that do not depend on external resources (e.g., databases), and built using a standard tool such as Maven [6]. We found that the libraries published by *Apache Commons* [5] met our criteria. In particular, they are written to an exacting standard, follows common syntactic and semantic guidelines as in large enterprise projects, and our classifiers after some initial training could understand their functioning reasonably well. At the time we conducted this study, Apache Commons contained the 41 projects (Table 1). Among those, for our study, we selected only projects that (1) could be built and tested

successfully with minimal effort; (2) completed mutation analysis successfully; (3) are released more than once; and, (4) are packaged as a single Maven module.

For the sake of clarity, in Table 1 we group the Apache Commons projects into two groups: the first group (top) contains the 21 projects (name, release, and commit hash) that we could build, test, and analyze successfully; the second group (bottom) contains the projects (name, release, and rejection note) that we discarded. Specifically, we discarded three projects that had no official release at the time we conducted this study and 17 projects that have a complex structure, fail to build, do not pass the available tests, or break mutation analysis even after some effort into fixing them.

Since our methodology requires manual mutant classification, which is expensive, we had to restrict the scope of our study; hence, we selected as test subjects the ten largest projects fulfilling our criteria. We highlight these projects in Table 1 and summarize their main properties in Table 2.

3.2 Test Suites Generation

Because the Original test suites in each project contain unit tests that developers manually wrote to check application- and domain-specific requirements, one can argue that using the Original test suites might result in a biased sampling of live mutants. To reduce this risk, we generated two additional test suites for each test subject. Specifically, we leveraged EvoSuite [24] to generate Random, a test suite that does not target any specific testing goal, and DynaMOSA, a test suite that maximizes coverage using the Dynamic Many-Objective Sorting Algorithm [44]. Consequently, we argue that the former test suite has the lowest generation bias, whereas the latter might introduce some bias being not completely random. For generating both test suites, we followed the best practice from SBST testing tool competition [23, 25, 45] and let EvoSuite generate unit tests for each class for 60 seconds, generate assertions for 120 seconds, and minimize the tests for 300 seconds.

3.3 Killable Mutants Estimation

We executed all the available test suites against all the generated mutants for each project, resulting in the execution of more than 2.5B test methods. We noticed that PIT (1) did not always identify all the mutants covered by the test cases; hence, it did not execute them; and (2) it did not always report the name of the test methods killing mutants, which is required by one of our sampling strategies. For the test execution, we chose *JUGE* [22], the open-source infrastructure developed for the SBST Java unit test tool competition, and extended it to address PIT's limitations and heavily parallelize the test executions across multiple computing nodes (we used a cluster of 50 nodes featuring heterogeneous hardware configurations).

After executing the tests, we filtered out faulty samples related to invalid and timed-out mutants. To reduce the risk of misclassifying "slow" mutants as killed, we conservatively granted each unit test a generous timeout of 10 seconds. Finally, we estimated the live mutant species richness on the ten test subjects using the twelve estimators on each of the three test suites using both test class and test method sampling. The estimates are plotted along with the 95% confidence intervals.

Table 1: List of considered Apache Commons projects

Project	Release	Commit Hash/Note
commons-beanutils	1.9.4-RC2	32ceb2c925
commons-cli	1.4	f7153c3c10
commons-codec	1.13	beafa49f88
commons-collections	4.4	cab58b3a80
commons-compress	1.18	b95d5cde4c
commons-configuration	2.5	dc00a04783
commons-csv	1.7	a227a1e2fb
commons-dbcp	2.6.0	3e7fca08d3
commons-dbutils	1.7	77faa3caef
commons-digester	3.2	ec75748096
commons-email	1.5	5516c487a5
commons-exec	1.3	0b1c1ff0cb
commons-fileupload	1.4	047f315764
commons-functor	1.0_RC1	62cd20998e
commons-imaging	1.0-alpha1-RC3	6f04ccc2cf
commons-io	2.6-RC3	2ae025fe5c
commons-lang	3.8_RC1	9801e2fb9f
commons-math	3.6_RC2	95a9d35e77
commons-net	3.6	163fe46c01
commons-pool	2.7.0	f4455dcb8a
commons-validator	1.6	c4b93a7275
commons-ognl	no releases	immature project
commons-geometry	no releases	immature project
commons-testing	no releases	immature project
commons-bcel	6.3.1	fail build
commons-vfs	2.4	complex structure
commons-text	1.7	fail build
commons-logging	1.2	fail mutation analysis
commons-jexl	v4.0-snapshot.4	failing tests
commons-crypto	1.0.0	failing tests
commons-jcs	2.2.1-RC4	failing tests
commons-chain	1.2	fail mutation analysis
commons-jxpath	1.3	fail mutation analysis
commons-scxml	2.0-M1	fail mutation analysis
commons-rng	1.2	complex structure
commons-rdf	0.5.0	complex structure
commons-proxy	2.0 RC1	fail build
commons-bsf	3.x-with-engines	complex structure
commons-weaver	2.0	complex structure
commons-jelly	1.0.1	fail build
commons-jci	1.1	complex structure

3.4 Survived Mutants Collection

To assess the quality of the statistical estimators, we needed a reliable approximation of killable and equivalent mutants in the ten test subjects; therefore, we opted to sample and manually classify the mutants that survived mutation analysis. As the first step, we used PIT [20] (v 1.4.9) to evaluate the *Original* test suites and identified mutants that PIT marked `SURVIVED` or `NOT_COVERED`. Specifically, we configured PIT to use its *default*[1] mutation operators and the *killmatrix* option enabled. Since mutation analysis is almost always used with unit tests, we did not consider any integration test that might appear in the *Original* test suite in this step.

[1] https://pitest.org/quickstart/mutators/

Table 2: Apache Commons projects — Projects and test suites measures

Project			Total	ORIGINAL				RANDOM				DYNAMOSA			
Id	Name	kLOC	Mutants	Size	Stmt	Branch	Kill	Size	Stmt	Branch	Kill	Size	Stmt	Branch	Kill
1	commons-net	20	5764	254	33	28	28%	1988	48	35	32%	3499	51	42	35%
2	commons-math	100	47881	6377	92	84	76%	11956	74	61	77%	17450	79	68	18%
3	commons-lang	28	13061	4114	95	91	66%	4500	73	60	72%	9027	89	86	86%
4	commons-io	10	3273	1316	90	88	94%	1677	72	65	58%	2822	81	79	87%
5	commons-imaging	31	11597	563	73	59	47%	2974	63	45	44%	4935	67	53	45%
6	commons-dbcp	14	4230	1409	66	66	99%	1026	39	42	20%	2970	47	58	27%
7	commons-csv	2	635	312	89	85	81%	365	78	63	60%	647	89	83	76%
8	commons-configuration	28	6279	2803	87	83	99%	2991	75	62	39%	4956	80	72	40%
9	commons-compress	24	9566	1047	84	76	65%	2610	63	45	44%	4306	70	57	49%
10	commons-collections	29	8309	25011	86	81	75%	3954	67	59	57%	5659	74	69	93%

3.5 Manual Mutants Classification

Running PIT on the ORIGINAL test suites left thousands of surviving mutants. Since we could not exhaustively classify all of those in a reasonable time, we randomly sampled 100 survived mutants for each project and used those for the manual classification. We argue that classifying so many mutants is a good balance between classification effort (estimated to be between one and thirty minutes per mutant) and representativeness of the obtained results. Nevertheless, the whole manual classification took six months.[2]

Given the sampled mutants, we adopted a structured classification protocol that involved three classifiers. The lead classifier (R_A) has ten years of experience in programming with Java and is an expert in mutation analysis. The other two classifiers (R_B and R_C), instead, have experience in programming (between three and five years) but were unfamiliar with mutation analysis before the start of this study. To cope with that, we organized a pilot study for training R_B and R_C using *commons-csv*, the smallest among the selected projects, and all the 116 mutants that survived the analysis.

After the initial training, the classification considered one project at a time in alphabetical order (as listed in Table 3) and required researchers R_B and R_C *independently* to classify all 100 sampled mutants. Killable mutants must be accompanied by a (hypothetical) unit test that could show the difference in behavior between that mutant and the original program. Similarly, equivalent mutants must be motivated with a convincing explanation. As a result, for each sampled mutant, we collected a label (killable or equivalent) along with how confident the classifier was in the classification (high, medium, low), and optional comments. Between R_A and R_B we achieved a Cohen's Kappa of 0.48 (moderate agreement).

As final step, we identified conflicting classifications (avg 4.9%) and let the three classifiers discuss and resolve them. During the discussion, the lead classifier R_A acted as the moderator while R_B and R_C could update their original classifications in light of the discussed arguments. The discussion continued until the final classification was accepted unanimously.

Given the proportion of killable mutants in a sample, we obtained the population proportion—and thus an estimate of the total number of killable mutants—with population proportion confidence intervals [43]. The estimates from manual classification are the 95% confidence intervals based on random sampling.

[2]The manual classification ran between July 2019 and December 2019.

Table 3: Manual classification of live mutants.

Project	Sampled	Misclass.		Equivalent
		R_B	R_C	
commons-net	100	0	4	1
commons-math	100	10	0	8
commons-lang	100	7	3	12
commons-io	100	3	2	5
commons-imaging	100	0	2	0
commons-dbcp	100	0	0	1
commons-csv*	116	3	23	9
commons-configuration	100	0	2	4
commons-compress	100	2	4	1
commons-collections	100	3	2	1

*We used commons-csv as a pilot to train researchers R_B and R_C.

4 RESULTS & DISCUSSION

Our study seeks answer to three main research questions investigating the ability of frequency based statistical estimators to predict the number of killable mutants in a project using biased (RQ1) and unbiased (RQ2) test suites as well as the robustness of such estimators against changes in sampling strategy (RQ3).

RQ1. Which statistical estimator best matches the estimates from manual classification using manually written test suites? To answer this question, we considered the statistical estimators computed using test methods for the ORIGINAL test suite as sampling unit and determined their 95% confidence intervals. Next, we checked whether the statistical estimators are significantly different from the manual classification, i.e., their 95% confidence interval did not overlap with the 95% confidence intervals from manual classification. We computed by how much they differ using normalized mean difference (MD) computed as the difference between the point estimate and manual classification estimate normalized by the manual classification estimate, i.e., $\frac{E-ME}{ME}$ ($\times$ 100 for percentage). We removed any estimate that resulted in negative CI upperbound, point estimate, or no estimate at all. We also removed any estimate that resulted in ∞ in upper CI. The results are given in Table 4.

Inspecting Table 4 reveals that in the majority of the cases the statistical estimators were significantly different from the manual

(a) ICE-k0 (b) Zelterman

(c) Chao Bunge (d) Jackknife

(e) Chao (f) iChao

(g) ICE (h) ICE-1

(i) UNPMLE (j) Bootstrap

(k) PNPMLE (l) PCG

Figure 3: Killable mutants estimated by species richness estimators. Each plot shows the performance of a single estimator across all projects, test suites (ORIGINAL is red, RANDOM is blue, and DYNAMOSA is green), and using class sampling (dotted lines) and method sampling (solid lines). For reference, estimates from manual classification are reported in solid purple. Projects (Y axis) numbered according to Table 2 column 1. The estimates or CIs going beyond 200x of the generated mutants are not shown.

Table 4: Comparison of *mean difference (MD)* between method estimators across subjects—ORIGINAL test suites

Estimator	Valid Estimates	CI Overlaps	Mean	Stdev
ICE-k0	8	0	19.2	20.6
Zelterman	9	1	15.4	12.5
Chao Bunge	9	3	22.2	43.8
Jackknife	9	0	15.3	10.6
Chao	9	1	16.9	16.1
iChao	9	2	16.0	14.5
ICE	9	1	18.1	16.1
ICE-1	9	2	16.8	12.8
UNPMLE	**10**	**5**	**11.1**	**9.8**
Bootstrap	10	0	18.5	19.9
PNPMLE	9	1	17.8	16.2
PCG	7	4	8.0	10.2

Table 5: Comparison of *mean difference (MD)* between method estimators across subjects—RANDOM test suites

Estimator	Valid Estimates	CI Overlaps	Mean	Stdev
ICE-k0	9	1	81.2	71.3
Zelterman	**10**	**1**	**32.6**	**19.6**
Chao Bunge	10	1	42.5	42.6
Jackknife	10	0	33.1	18.3
Chao	10	1	37.1	21.2
iChao	10	0	36.1	20.0
ICE	10	1	36.7	19.6
ICE-1	10	0	36.5	18.2
UNPMLE	9	2	37.8	17.7
Bootstrap	10	2	36.7	23.1
PNPMLE	10	1	34.0	20.2
PCG	10	2	33.7	21.4

classification and their mean absolute difference varied considerably. Out of ten test subjects, UNPMLE behaved consistently with manual classification 5 times (MD 11.1%, Stdev. 9.8%). However, we note that the confidence intervals were very large in many cases for UNPMLE (Fig. 3). The other estimators were significantly different (less than 5 CI overlaps) than manual classification.

> There is a statistically significant difference between the estimates from ORIGINAL test suites and manual classification for every estimator examined.

RQ2. Which statistical estimator best predicts the estimates from manual classification when given unbiased test suites?
To answer this question, we considered the statistical estimators computed using test methods for RANDOM and DYNAMOSA test suites as sampling unit and determined their 95% confidence intervals. Next, we checked whether the estimators' 95% confidence intervals consistently overlapped with the one from manual classification, and computed their mean absolute difference with respect to manual classification. The results are given in Table 5 for RANDOM, and in Table 6 for DYNAMOSA.

Table 6: Comparison of *mean difference (MD)* between method estimators across subjects—DYNAMOSA test suites

Estimator	Valid Estimates	CI Overlaps	Mean	Stdev
ICE-k0	9	4	77.9	77.5
Zelterman	10	1	39.2	20.6
Chao Bunge	10	1	40.8	22.3
Jackknife	10	1	39.3	24.9
Chao	10	1	39.1	24.9
iChao	10	0	39.5	23.0
ICE	10	1	39.2	24.8
ICE-1	10	1	39.8	22.9
UNPMLE	7	2	39.5	26.8
Bootstrap	**10**	**3**	**39.1**	**24.8**
PNPMLE	10	1	49.5	42.6
PCG	9	1	42.4	40.0

Table 5, reveals that the overlaps between confidence intervals from manual classification and RANDOM was considerably fewer than ORIGINAL. The smallest difference in MD was observed for Zelterman (mean 32.6%, Stdev. 19.6%). For DYNAMOSA, the situation was slightly better, ICE-k0 overlapped with manual classification estimates 4 times out of 10. The smallest difference in MD, with least variation was observed for Bootstrap (mean 39.1% Stdev. 24.8%).

Comparing the estimators across RANDOM and DYNAMOSA test suites, we also observed that none behaved consistently, i.e., their 95% confidence intervals did not overlap most of the times.

> The difference between estimates from RANDOM and DYNAMOSA to the estimates from manual classification is statistically significant for every estimator examined. Likewise, estimates from RANDOM are always significantly differently than the ones from DYNAMOSA.

RQ3. Which mapping of sampling units best predicts the estimates from manual classification?
To answer this question, we considered the statistical estimators computed using test methods and test classes as sampling unit for all the tests suites and determined their 95% confidence intervals. Next, we checked whether the estimators' 95% confidence intervals consistently overlapped with the one from manual classification, and computed their mean absolute difference with respect to manual classification. The results are given in tables Table 7, Table 8, and Table 9. These results are also plotted as dotted lines in Fig. 3.

From Table 7, we see that Bootstrap had the best overlap with CI (6 times), and mean difference (mean 16.9%, Stdev. 19.2%). For RANDOM and DYNAMOSA, the situation was the same as before with the manual classification and estimates confidence intervals not overlapping a majority of the time. Furthermore, we also found that the estimates from method level sampling units and class level sampling units are also statistically significantly different.

Table 7: Comparison of *mean difference (MD)* between class estimators across subjects—ORIGINAL test suites

Estimator	Valid Estimates	CI Overlaps	Mean	Stdev
ICE-k0	9	1	35.2	51.4
Zelterman	9	1	36.7	51.8
Chao Bunge	7	0	22.8	19.1
Jackknife	10	0	41.2	53.5
Chao	9	1	32.4	42.5
iChao	9	1	35.9	49.5
ICE	9	0	41.2	57.9
ICE-1	8	0	40.2	43.5
UNPMLE	6	0	31.9	31.8
Bootstrap	**10**	**6**	**16.9**	**19.2**
PNPMLE	8	1	64.9	34.8
PCG	6	1	43.9	39.5

Table 8: Comparison of *mean difference (MD)* between class estimators across subjects—RANDOM test suites

Estimator	Valid Estimates	CI Overlaps	Mean	Stdev
ICE-k0	8	1	61.1	65.1
Zelterman	10	4	25.4	39.0
Chao Bunge	4	1	27.3	26.0
Jackknife	10	2	33.1	44.7
Chao	10	2	26.3	33.3
iChao	10	2	27.7	40.7
ICE	10	1	29.7	29.9
ICE-1	10	3	52.4	61.1
UNPMLE	7	3	34.9	63.9
Bootstrap	10	2	30.9	23.0
PNPMLE	**9**	**4**	**23.6**	**24.9**
PCG	7	3	24.3	29.2

Table 9: Comparison of *mean difference (MD)* between class estimators across subjects—DYNAMOSA test suites

Estimator	Valid Estimates	CI Overlaps	Mean	Stdev
ICE-k0	10	3	49.3	46.4
Zelterman	10	1	20.7	16.3
Chao Bunge	5	1	34.6	42.4
Jackknife	10	0	37.3	25.6
Chao	**10**	**2**	**19.9**	**14.0**
iChao	10	0	22.5	14.6
ICE	10	1	28.0	15.4
ICE-1	10	1	52.4	42.4
UNPMLE	7	0	35.5	19.4
Bootstrap	10	4	36.9	20.9
PNPMLE	10	1	55.7	44.5
PCG	8	0	57.7	51.7

The difference between estimates from test method and test class level estimators to the estimates from manual classification are statistically significant for every estimator examined. Furthermore, the difference between each other is statistically significant for every estimator examined.

4.1 What Does This Mean in Practice?

Given that using various test suites and sampling strategies always produced inconsistent and unsatisfactory results for all the considered estimators despite the underlying sampled quantity, i.e., killable mutants, did not change, our (current) conclusion is that frequency based statistical estimators are not applicable to the problem of estimating killable mutants.

Our negative results suggest a need of further empirical evidence and additional analysis to understand the impact of various assumptions made by those estimators and their applicability to mutation analysis. In other words, despite the progress in using those estimators to assess residual risk from the structural coverage point of view, for mutation analysis we are not there yet!

4.2 Threats to Validity

External Validity. Our study was conducted on a limited number of programs from a specific open-source repository and using a single test generator; hence, our findings may not generalize to other projects and test suites generated by other means. To reduce this risk, we selected multiple projects from Apache Commons and used EvoSuite in two exemplary configurations. Apache Commons projects are popular, implement different functionalities, and are comparable to well run industrial projects. EvoSuite, instead, implements standard baselines and well established and effective algorithms that generate test suites with remarkably different features. We use a single run of both the test generator (randomized) on our classes. While multiple runs are required for statistical confidence of the results, we note that our approach is similar to the one adopted by established biometrics studies that draw conclusions from single sampling campaigns.

Internal Validity. The test suites automatically generated did not always achieve high coverage, hence, they can lead to larger uncertainty in the final estimation of equivalent mutants. However, we note that this situation is similar to the one currently faced by software practitioner. Our analysis can be subject to bugs, sampling errors, and mutant mis-classifications that might bias our results. For instance, the test generator could not generate unit tests for all the classes under analysis, hence failing to sample their mutants. We tried to mitigate this risk by reviewing the code of JUGE and our scripts, cross-checking the results, and use the largest possible subset of classes for which test generation succeeded. It is possible that our manual classification is biased. We tried to mitigate it by cross-checking between three classifiers.

Construct Validity. We are the first to apply frequency based statistical models to the problem of killable mutants estimation; hence, our mapping of statistical estimators to the mutation testing domain might not to capture important variables. We tried to mitigate this threat by adopting different sampling strategies.

5 RELATED WORK

Mutation analysis is considered a primary way of evaluating test quality [47]; thus, mutation score is usually considered as a test suite adequacy metrics [3, 4, 21, 34]. Unfortunately, equivalent mutants have vexed practitioners from the very beginning [11], and remains an open issue [36] that affects also residual risk estimation. To address this issue, we proposed to estimate killable mutants

using frequency based statistical estimators that have been applied recently to estimate reachable coverage and conducted a large empirical study that included 10 mature open-source Java projects, 3 types of test suites, and the manual classification of 1016 live mutants involving 3 researchers.

A few previous studies also relied on the manual mutants classification. Acree's study [2] involved two competent software engineers experts in mutation analysis that classified live mutants from four COBOL programs. Similar to our study, Acree used manually written tests for eliminating a large chuck of mutants; however, differently from us, the labelers had no exposure to the programs under analysis and focused on small COBOL programs. During his study, Acree documented various misclassifications (avg. 23%), suggesting that even manual classification has errors. We also found misclassifications (see Table 3, *Misclass.* column), but achieved better accuracy (less than 5% misclassifications on avg),[3] arguably because we trained the labelers and followed a structured and systematic classification protocol.

Other studies of note are by Yao et al. [59] on 18 C programs, and Grün et al. [30] (extended by Schuler et al. [51]) on 7 Java programs. Both studies involved a single researcher but classified a different amount of live mutants, 1, 194 Yao et al. and 140 Grün et al. Yao et al.'s study estimated that 77% of all mutants are killable, while Grün et al.'s reported that 45% of the *classified* live mutants are equivalent. Unfortunately, none of those studies reported the misclassification rate. Compared to those studies, our manual classification required (modulo the number of mutants) the same amount of time, but involved twice as many researchers. We also considered real-world, more complex, and arguably more difficult to evaluate, projects, and a sufficiently representative set of mutation operators [26]. Finally, we studied manually written and automatically generated unit test suites, that covered more mutants and resulted in estimating a higher number (generally above 90%) of killable mutants.

Papadakis et al. [46] is a study similar to ours, with more numerous subjects. However, compared to that study, we consider larger programs and different test suites, do not employ selective mutation, whose limits have been discussed empirically and theoretically [26, 27], and we employ mechanisms such as conflict identification and resolution, to reduce manual classification error proneness. We also note that Papadakis et al. requires manual classification, however limited. This may not be feasible in many cases where the testers may not be program experts. Furthermore, evaluating residual risk may be conducted by people who are not involved in either testing or development (such as the end-user). Hence, alternative means of estimating killable mutants is required.

We used statistical estimators for predicting killable mutants, others, instead, used them for estimating residual faults. For instance, Böhme [9] argued to use the same species richness estimators we studied for estimating residual defect density, while Nayak [40] and Voas and McGraw [56] modeled faults and residual defects as members of unknown populations and estimated their number via *capture-recapture* methods. Tohma et al. [54], instead, modeled the distribution of observed faults as hyper-geometric distribution to estimate the number of residual defects.

[3]This measure does not consider the results of `commons-csv` that we used for training the labelers.

To estimate equivalent mutants, other methods have been proposed. Vincenzi et al. [55] proposed estimating the (posterior) probability that specific mutation operators generate equivalent mutants. Marsit et al. [7, 37, 38] proposed using information theory to measure the intrinsic *redundancy* in programs as a proxy for mutants equivalence. Despite their potential benefits and promising initial results, none of those methods has been empirically evaluated yet.

6 CONCLUSIONS AND FUTURE WORK

Accurately estimating the number of killable mutants is crucial for estimating the residual risk and the effectiveness of test generators. While a sound and complete classifier for killable and equivalent mutants is impossible, recent advances in statistical estimation using frequency based estimators gave us hope that one could at least estimate the number of killable mutants. Such estimators have been used successfully in related domain for instance to estimate reachable coverage and reachable computer systems in a network.

This research provides the first, large evaluation of these estimators on mutation analysis. First, we manually classified a statistically significant sample surviving mutants and used them as a ground truth. Next, we evaluated more than a hundred thousand mutants using manually and automatically generated test suites and estimated the killable mutants in each project and for each test suite using twelve state-of-the-art species richness estimators. We estimated killable mutants using data sampled at test method and test class granularity. Finally, we compared the resulting 720 estimates against the ground truth to measure the quality of the statistical estimators on ten real-world projects across projects and test suites. We compared the estimates across sampling strategies to understand whether the estimators produce significantly different results despite estimating the same quantity. Our results show that the considered statistical estimators applied to the killable mutants estimation do not produce consistent results across projects, test suites, and when computed using different sampling strategies. Therefore, they have way to go before practitioners can reliably use them.

We recommend a few directions for future research:

Better models. The first question is of course whether we can get better performance from identifying the unique aspects of mutant distribution, and correct our models and the estimators for that. Doing so would require us to identifying the mutant distribution characteristics and developing models that can incorporate such information.

Investigate more variables. To improve estimation, incorporate coverage spectra and other information either by modifying the test sample or directly into the model to see if such information can help estimate better killable mutants.

Evaluating mutant classifiers. Finally, given that we have a large set of manually classified mutants, we can use them to evaluate the current killable and equivalent mutant classifiers such as coverage based [52], machine learning based [39, 49], or other [48] classifiers.

7 DATA AVAILABILITY

The replication package [1] contains data about manual classification of mutants, test suites, kill matrices, and scripts to compute and plot the estimations. We will release data publicly upon acceptance.

REFERENCES

[1] [n.d.]. https://anonymous.4open.science/r/chaos-replication-848B.

[2] Allen Troy Acree Jr. 1980. *On Mutation.* Ph.D. Dissertation. Georgia Institute of Technology, Atlanta, Georgia. GIT-ICS-80/12.

[3] James H Andrews, Lionel C Briand, and Yvan Labiche. 2005. Is mutation an appropriate tool for testing experiments?. In *Proceedings of the 27th international conference on Software engineering*. 402–411.

[4] James H Andrews, Lionel C Briand, Yvan Labiche, and Akbar Siami Namin. 2006. Using mutation analysis for assessing and comparing testing coverage criteria. *IEEE Transactions on Software Engineering* 32, 8 (2006), 608–624.

[5] Apache Software Foundation. [n.d.]. Apache Commons. http://commons.apache.org/.

[6] Apache Software Foundation. [n.d.]. Apache Maven. https://maven.apache.org/.

[7] A. Ayad, I. Marsit, J. Loh, M. N. Omri, and A. Mili. 2019. Estimating the Number of Equivalent Mutants. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 112–121.

[8] Marcel Böhme. 2018. Assurances in Software Testing: A Roadmap. *CoRR* abs/1807.10255 (2018). arXiv:1807.10255 http://arxiv.org/abs/1807.10255

[9] Marcel Böhme. 2018. STADS: Software testing as species discovery. *ACM Transactions on Software Engineering and Methodology* 27, 2 (1 7 2018). https://doi.org/10.1145/3210309

[10] Dankmar Böhning. 2010. Some general comparative points on Chao's and Zelterman's estimators of the population size. *Scandinavian Journal of Statistics* 37, 2 (2010), 221–236.

[11] Timothy A Budd and Dana Angluin. 1982. Two notions of correctness and their relation to testing. *Acta informatica* 18, 1 (1982), 31–45.

[12] K. P. Burnham and W. S. Overton. 1978. Estimation of the Size of a Closed Population when Capture Probabilities vary Among Animals. *Biometrika* 65, 3 (1978), 625–633. http://www.jstor.org/stable/2335915

[13] K. P. Burnham and W. S. Overton. 1979. Robust Estimation of Population Size When Capture Probabilities Vary Among Animals. *Ecology* 60, 5 (1979), 927–936. http://www.jstor.org/stable/1936861

[14] Anne Chao. 1984. Nonparametric estimation of the number of classes in a population. *Scandinavian Journal of statistics* (1984), 265–270.

[15] Anne Chao and John Bunge. 2002. Estimating the number of species in a stochastic abundance model. *Biometrics* 58, 3 (2002), 531–539.

[16] Anne Chao and Chun-Huo Chiu. 2016. *Nonparametric Estimation and Comparison of Species Richness.* American Cancer Society, 1–11. https://doi.org/10.1002/9780470015902.a0026329 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470015902.a0026329

[17] Anne Chao and Chun-Huo Chiu. 2016. Species richness: estimation and comparison. *Wiley StatsRef: statistics reference online* 1 (2016), 26.

[18] Anne Chao, SM Lee, and SL Jeng. 1992. Estimating population size for capture-recapture data when capture probabilities vary by time and individual animal. *Biometrics* (1992), 201–216.

[19] Chun-Huo Chiu, Yi-Ting Wang, Bruno A Walther, and Anne Chao. 2014. An improved nonparametric lower bound of species richness via a modified good–turing frequency formula. *Biometrics* 70, 3 (2014), 671–682.

[20] Henry Coles. [n.d.]. PIT - Real world mutation testing. https://pitest.org.

[21] Murial Daran and Pascale Thévenod-Fosse. 1996. Software error analysis: A real case study involving real faults and mutations. *ACM SIGSOFT Software Engineering Notes* 21, 3 (1996), 158–171.

[22] Xavier Devroey, Alessio Gambi, Juan Pablo Galeotti, René Just, Fitsum Meshesha Kifetew, Annibale Panichella, and Sebastiano Panichella. 2021. JUGE: An Infrastructure for Benchmarking Java Unit Test Generators. *CoRR* abs/2106.07520 (2021). arXiv:2106.07520 https://arxiv.org/abs/2106.07520

[23] Xavier Devroey, Sebastiano Panichella, and Alessio Gambi. 2020. Java Unit Testing Tool Competition: Eighth Round. In *ICSE '20: 42nd International Conference on Software Engineering, Workshops, Seoul, Republic of Korea, 27 June - 19 July, 2020*. ACM, 545–548. https://doi.org/10.1145/3387940.3392265

[24] Gordon Fraser and Andrea Arcuri. 2011. Evosuite: automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 416–419.

[25] Alessio Gambi, Gunel Jahangirova, Vincenzo Riccio, and Fiorella Zampetti. 2022. SBST Tool Competition 2022. In *15th IEEE/ACM International Workshop on Search-Based Software Testing, SBST@ICSE 2022, Pittsburgh, PA, USA, May 9, 2022*. IEEE, 25–32. https://doi.org/10.1145/3526072.3527538

[26] Rahul Gopinath, Iftekhar Ahmed, Mohammad Amin Alipour, Carlos Jensen, and Alex Groce. 2017. Mutation reduction strategies considered harmful. *IEEE Transactions on Reliability* 66, 3 (2017), 854–874.

[27] Rahul Gopinath, Amin Alipour, Iftekhar Ahmed, Carlos Jensen, and Alex Groce. 2016. On the limits of mutation reduction strategies. In *Proceedings of the 38th International Conference on Software Engineering*. ACM.

[28] Nicholas J Gotelli and Anne Chao. 2013. Measuring and estimating species richness, species diversity, and biotic similarity from sampling data. (2013).

[29] Nicholas J Gotelli and Robert K Colwell. 2011. Estimating species richness. *Biological diversity: frontiers in measurement and assessment* 12 (2011), 39–54.

[30] Bernhard JM Grün, David Schuler, and Andreas Zeller. 2009. The impact of equivalent mutants. In *2009 International Conference on Software Testing, Verification, and Validation Workshops*. IEEE, 192–199.

[31] Joseph R Horgan and Aditya P Mathur. 1996. Software testing and reliability. In *Handbook of software reliability engineering*. 531–566.

[32] Joaquín Hortal, Paulo AV Borges, and Clara Gaspar. 2006. Evaluating the performance of species richness estimators: sensitivity to sample grain size. *Journal of animal ecology* 75, 1 (2006), 274–287.

[33] Yue Jia and Mark Harman. 2010. An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering* 37, 5 (2010), 649–678.

[34] René Just, Darioush Jalali, Laura Inozemtseva, Michael D Ernst, Reid Holmes, and Gordon Fraser. 2014. Are mutants a valid substitute for real faults in software testing?. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 654–665.

[35] Shen-Ming Lee and Anne Chao. 1994. Estimating population size via sample coverage for closed capture-recapture models. *Biometrics* (1994), 88–97.

[36] L. Madeyski, W. Orzeszyna, R. Torkar, and M. Józala. 2014. Overcoming the Equivalent Mutant Problem: A Systematic Literature Review and a Comparative Experiment of Second Order Mutation. *IEEE Transactions on Software Engineering* 40, 1 (2014), 23–42.

[37] Imen Marsit, Mohamed Nazih Omri, JiMing Loh, and Ali Mili. 2018. Impact of Mutation Operators on Mutant Equivalence.. In *ICSOFT*. 55–66.

[38] Imen Marsit, Mohamed Nazih Omri, and Ali Mili. 2017. Estimating the Survival Rate of Mutants. In *ICSOFT*.

[39] Muhammad Rashid Naeem, Tao Lin, Hamad Naeem, and Hailu Liu. 2020. A machine learning approach for classification of equivalent mutants. *Journal of Software: Evolution and Process* 32, 5 (2020), e2238. https://doi.org/10.1002/smr.2238 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2238 e2238 smr.2238.

[40] Tapan Nayak. 1988. Estimating Population Size by Recapture Sampling. *Biometrika* 75 (03 1988). https://doi.org/10.2307/2336441

[41] James L Norris and Kenneth H Pollock. 1998. Non-parametric MLE for Poisson species abundance models allowing for heterogeneity between species. *Environmental and Ecological Statistics* 5, 4 (1998), 391–402.

[42] A. Jefferson Offutt and W. Michael Craft. 1994. Using compiler optimization techniques to detect equivalent mutants. *Software Testing, Verification and Reliability* 4, 3 (1994), 131–154. https://doi.org/10.1002/stvr.4370040303 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/stvr.4370040303

[43] R Lyman Ott and Micheal T Longnecker. 2015. *An introduction to statistical methods and data analysis.* Cengage Learning.

[44] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2015. Reformulating branch coverage as a many-objective optimization problem. In *IEEE International Conference on Software Testing, Verification and Validation*. IEEE, 1–10.

[45] Sebastiano Panichella, Alessio Gambi, Fiorella Zampetti, and Vincenzo Riccio. 2021. SBST Tool Competition 2021. In *14th IEEE/ACM International Workshop on Search-Based Software Testing, SBST 2021, Madrid, Spain, May 31, 2021*. IEEE, 20–27. https://doi.org/10.1109/SBST52555.2021.00011

[46] Mike Papadakis, Marcio Delamaro, and Yves Le Traon. 2014. Mitigating the effects of equivalent mutants with mutant classification strategies. *Science of Computer Programming* 95 (2014), 298–319.

[47] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. 2019. Mutation testing advances: an analysis and survey. In *Advances in Computers*. Vol. 112. Elsevier, 275–378.

[48] Mike Papadakis and Yves Le Traon. 2013. Mutation Testing Strategies Using Mutant Classification. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing* (Coimbra, Portugal) *(SAC '13)*. Association for Computing Machinery, New York, NY, USA, 1223–1229. https://doi.org/10.1145/2480362.2480592

[49] Samuel Peacock, Lin Deng, Josh Dehlinger, and Suranjan Chakraborty. 2021. Automatic Equivalent Mutants Classification Using Abstract Syntax Tree Neural Networks. In *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 13–18. https://doi.org/10.1109/ICSTW52544.2021.00016

[50] Henry Gordon Rice. 1953. Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.* 74, 2 (1953), 358–366.

[51] David Schuler and Andreas Zeller. 2010. (Un-) covering equivalent mutants. In *2010 Third International Conference on Software Testing, Verification and Validation*. IEEE, 45–54.

[52] David Schuler and Andreas Zeller. 2013. Covering and uncovering equivalent mutants. *Software Testing, Verification and Reliability* 23, 5 (2013), 353–374.

[53] Eric P. Smith and Gerald van Belle. 1984. Nonparametric Estimation of Species Richness. *Biometrics* 40, 1 (1984), 119–129. http://www.jstor.org/stable/2530750

[54] Yoshihiro Tohma, Kenshin Tokunaga, Shinji Nagase, and Yukihisa Murata. 1989. Structural approach to the estimation of the number of residual software faults based on the hyper-geometric distribution. *IEEE transactions on software engineering* 15, 3 (1989), 345–355.

[55] Auri Vincenzi, Elisa Nakagawa, José Maldonado, Márcio Delamaro, and Roseli Romero. 2002. Bayesian-Learning Based Guidelines to Determine Equivalent Mutants. *International Journal of Software Engineering and Knowledge Engineering* 12 (12 2002), 675–689. https://doi.org/10.1142/S021819400200113X

[56] Jeffrey M. Voas and Gary McGraw. 1997. *Software Fault Injection: Inoculating Programs against Errors.* John Wiley & Sons, Inc., USA.

[57] Ji-Ping Wang. 2010. Estimating species richness by a Poisson-compound gamma model. *Biometrika* 97, 3 (2010), 727–740.

[58] Ji-Ping Z Wang and Bruce G Lindsay. 2005. A penalized nonparametric maximum likelihood approach to species richness estimation. *J. Amer. Statist. Assoc.* 100, 471 (2005), 942–959.

[59] Xiangjuan Yao, Mark Harman, and Yue Jia. 2014. A study of equivalent and stubborn mutation operators using human analysis of equivalence. In *Proceedings of the 36th International Conference on Software Engineering*. 919–930.

[60] Andreas Zeller, Rahul Gopinath, Marcel Böhme, Gordon Fraser, and Christian Holler. 2023. When To Stop Fuzzing. In *The Fuzzing Book*. CISPA Helmholtz Center for Information Security. https://www.fuzzingbook.org/html/WhenToStopFuzzing.html Retrieved 2023-01-07 15:26:49+01:00.

A CHALLENGES FACED

During our empirical evaluation, we faced a number of challenges, and here is a list of some of the tougher challenges we faced.

Flacky tests. Flaky tests were a major source of problems during our evaluation. In particular, tests that run and report no failure during manual runs sometimes fail during PIT runs for extracting coverage.

Residual Errors in Classification. We found that even after three fold confirmation, one of the mutants marked as equivalent was actually killed by an automatically generated test case.

PIT bugs The PIT version we used sometimes resulted in corrupted XML files as output. Furthermore, PIT would sometimes fail to detect bugs due to its incorrect coverage heuristics. We found that some such tests would detect mutants.

Sandboxes When PIT was used in programs that contained file IO, it often corrupted the file system, overwriting unintended files.

Bugs in R packages Some of the statistical estimation packages had bugs in them, which we found thanks to implementing the estimation formulas ourselves and cross checking the results.

Limitations of EvoSuite The EvoSuite test generator would not run on all programs we used.

B CHARTS

B.1 Original

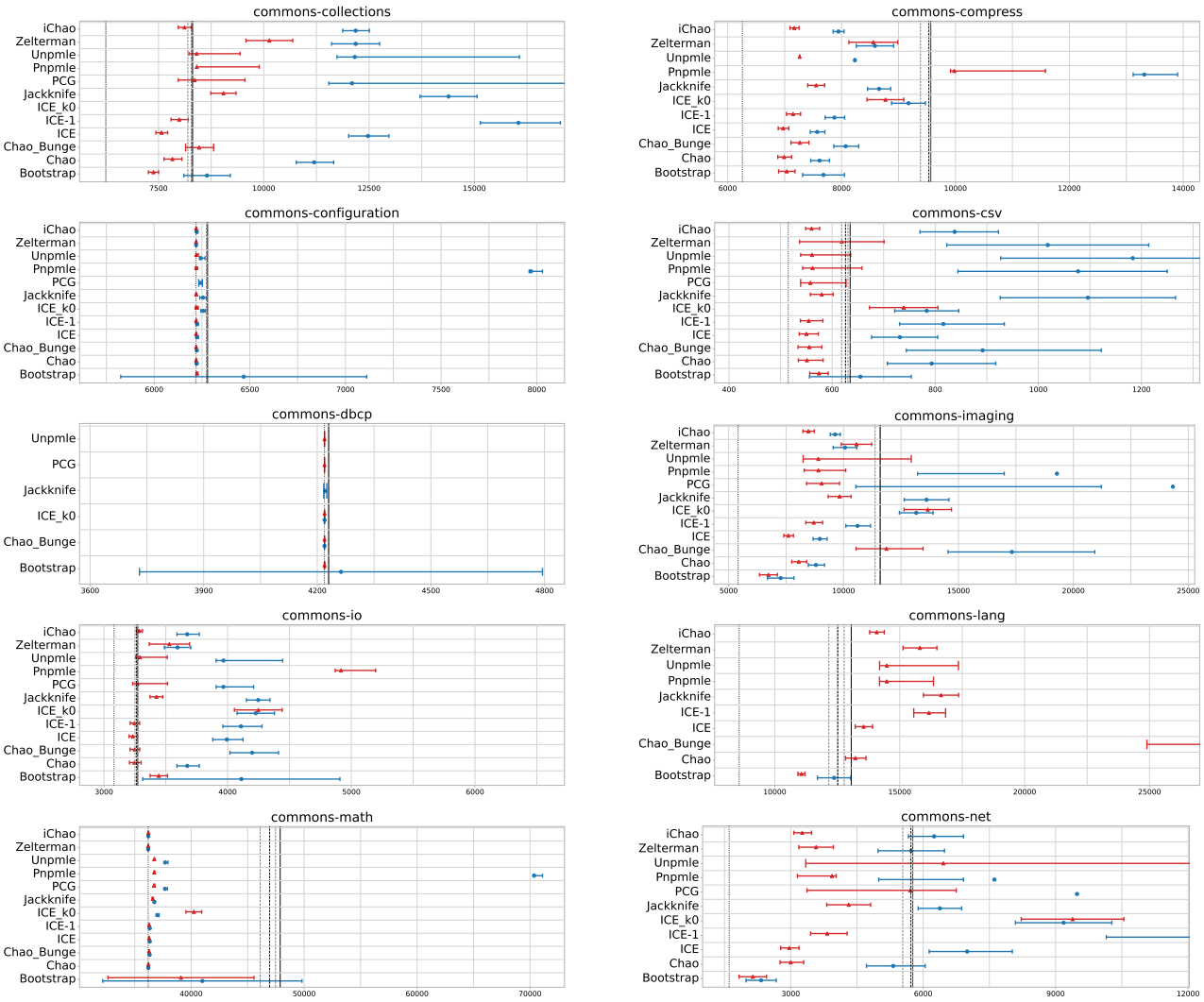

Figure 4: Estimators from test Original. The blue is from *class test set* data, and red from *method test set* data. Dotted black line corresponds to the number of killed mutants, dashed black line corresponds to the number of manually estimated killed mutants, dashed gray lines define the confidence interval of manual estimation.

B.2 Random

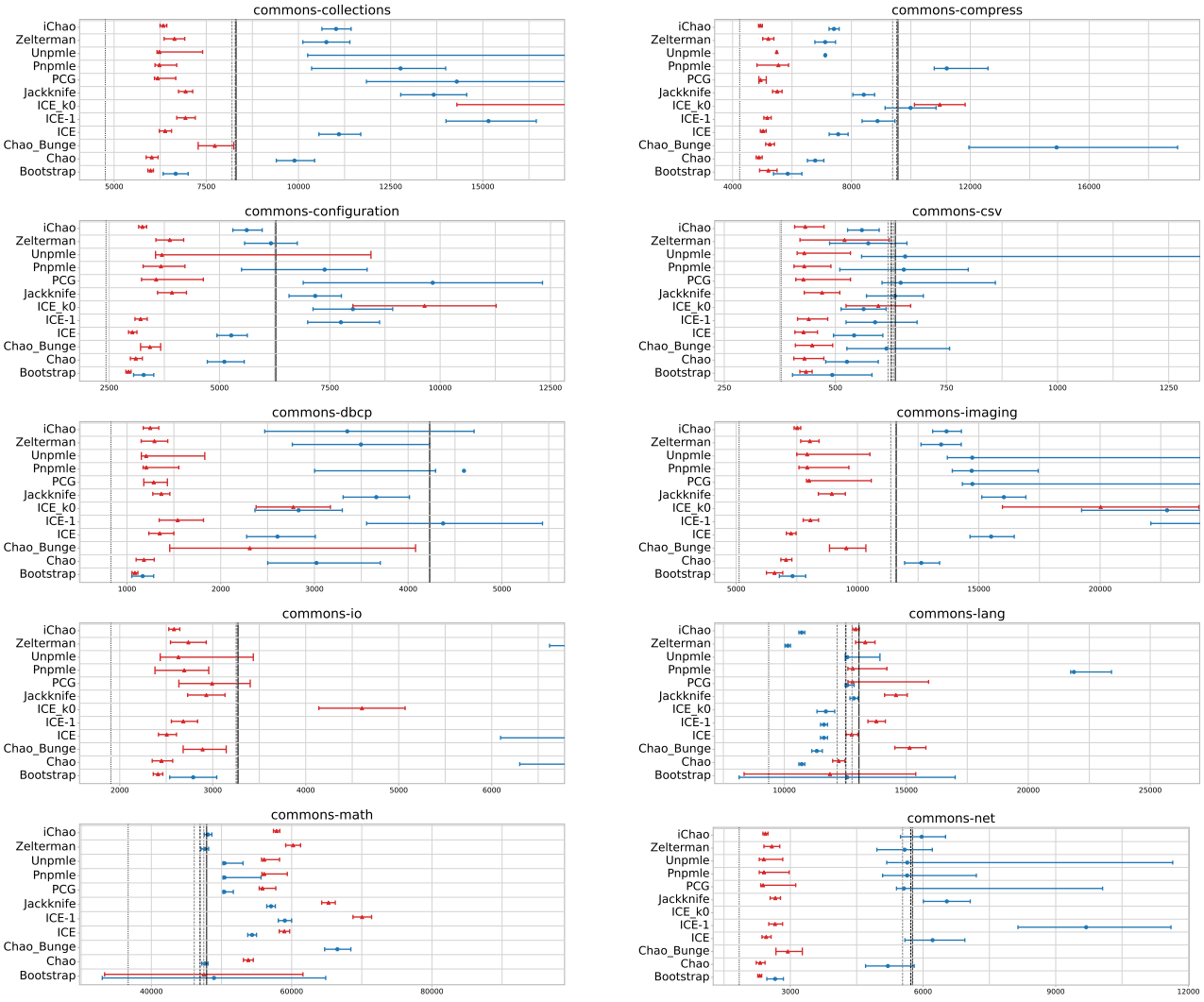

Figure 5: Estimators from test Random. The blue is from *class test set* data, and red from *method test set* data. Dotted black line corresponds to the number of killed mutants, dashed black line corresponds to the number of manually estimated killed mutants, dashed gray lines define the confidence interval of manual estimation.

B.3 DynaMOSA

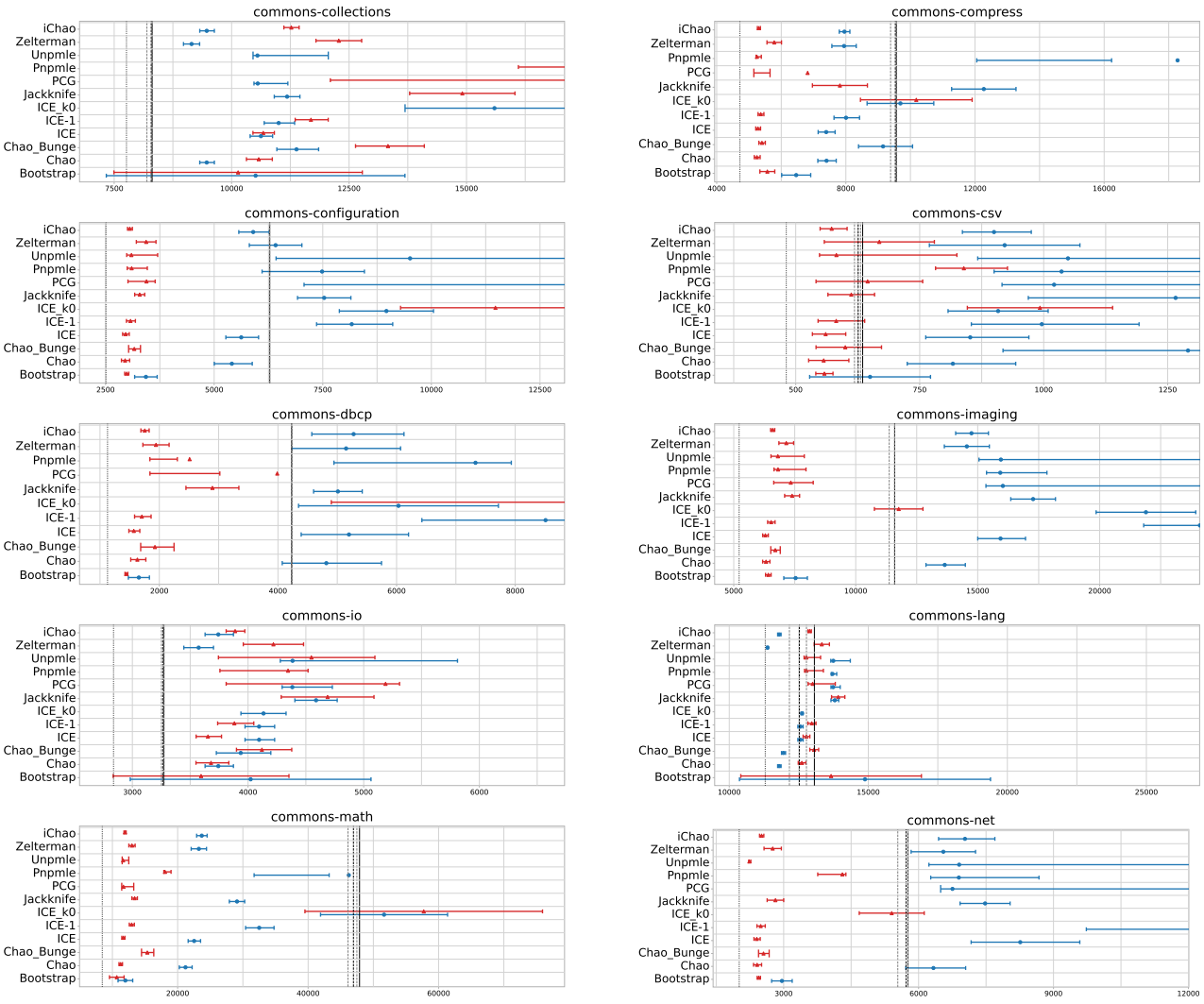

Figure 6: Estimators from test DynaMOSA. The blue is from *class test set* data, and red from *method test set* data. Dotted black line corresponds to the number of killed mutants, dashed black line corresponds to the number of manually estimated killed mutants, dashed gray lines define the confidence interval of manual estimation.