Estimating Equivalent Mutants: Are we there yet?

Konstantin Kuznetsov*, Alessio Gambi†, Saikrishna Dhiddi‡ and Julia Hess§ and Rahul Gopinath§
Email: *konstantin.kuznetsov@cispa.de, †alessio.gambi@fh-krems.ac.at, ‡dhiddi01@ads.uni-passau.de, §rahul.gopinath@sydney.edu.au

***Abstract*—Mutation analysis systematically introduces artificial defects (mutants) into software to assess test suite quality. The number of mutants killed by a test suite is considered a good proxy for its fault revealing power. However, the reliability of this technique is hampered by *equivalent mutants*, i.e., mutants semantically equivalent to the original code, that cannot be detected by any test case.**

Statistical estimators for population, or species, size estimation have been recently proposed to alleviate this problem by estimating the number of "killable" mutants in the overall mutant population. While promising, the lack of a comprehensive study on their effective accuracy and reliability prevents its usage by the practitioners.

This paper makes the first step towards filling this gap and presents the results of a large-scale empirical study we conducted on the application of twelve widely known statistical estimators for population, or species, size for estimating the number of "killable" mutants in ten mature Apache projects.

Our results indicate that the statistical estimators we considered did not have sufficient predictive power to be useful to the practitioners, and the prediction is dependent on the particular program and test suite under evaluation.

***Index Terms*—estimation, mutation analysis, software testing**

I. Introduction

Mutation analysis is one of the premier means of assessing the quality of software test suites [1]. Evaluating a test suite with mutation analysis involves generating mutants *exhaustively* and evaluating them against the test cases comprising that test suite (Figure 1-right). Specifically, mutants are copies of the original code (Figure 1-left) in which simple, plausible faults, i.e., the mutations, are injected (Figure 2). Notably, mutations show strong similarities with real faults [2], [3] and are as difficult to find [4], [5].

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 *live* 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 its effectiveness in preventing faults [6].

Not all the generated mutants might be killable, no matter the test input used [7] (see Figure 2-right for an example). The amount of these so-called *equivalent mutants* cannot be established *a-priori* because there is no means to automatically prove that a mutant is an equivalent one in general [7], [8]. As a consequence, the mutation score might be inaccurate.

Can we assume that the number of equivalent mutants are always within a fixed percentage irrespective of the program? Unfortunately, the number of killable mutants is program-specific and can vary across projects drastically [9], [10].

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

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

```
1 def determinant(a,b,c,d):  def determinant(a,b,c,d):
2 - ad = a * d                 ad = a * d
3 + ad = a / d               - bc = b * c
4   bc = b * c               + bc = c * b
5   return ad - bc             return ad - bc
```

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

As a consequence, standard statistical estimators that do not consider program specific factors or features cannot accurately estimate the number of killable mutants [11].

Recently, researchers proposed the STADS framework [12] as a general framework capable of estimating the total number of software species when only a fraction has been discovered. It was recently proposed to adopt STADS framework to estimate the total number of killable mutants using the distribution of already killed mutants [13]. Specifically, the killable mutants are assumed to belong to a *population*, or *species*, whose size can be estimated by means of existing statistical estimators (e.g., [12], [14]). However, those estimators rely on assumptions about the distribution of the population and behave differently when such assumptions do not hold, and their ability to predict the number of killable mutants has not yet empirically evaluated. Hence, we conducted a large study on how well twelve statistical estimators from the literature predict killable mutants generated by the state-of-art mutation testing framework PIT[1] for ten mature and well-tested open-source projects (Table II).

This paper presents our study and summarizes its main findings, while the attached replication package [**?**] contains the whole dataset, results, and scripts to replicate the study.

In a nutshell, the main hypothesis of our study is that population, or species, size estimators are accurate and reliable. To verify this hypothesis, we manually classified live mutants obtained by running the existing (i.e., ORIGINAL) test suites as equivalent or killable, and checked whether the predictions made by the statistical estimators are not significantly different than the ones we obtained from the manual classification.

[1] https://pitest.org

We, however, found that the estimates made by the statistical estimators are often far from what we expected, i.e., they are indeed statistically different from the from manual estimates.

We critically questioned the setup of our study and identified two threats to the validity of our conclusions. Specifically, we argued that (1) the manual classification could be in error, hence the estimators might be closer to the truth than we saw, and (2) the way we sampled the data (i.e., by looking at the test classes killing mutants) could affect the quality of the estimates. To mitigate the first challenge, we extended our study with two test suites created using EVOSUITE[2], a state-of-the-art test generation tool, using fundamentally different configurations (i.e., random and guided by coverage).

—- ALESSIO HERE —-

The idea here is that predictions from ORIGINAL as well as EVOSUITE test suits should coincide as they predict the same quantity.

We found that in many cases, the difference between the predicted value from manual data and what was predicted by various estimators were quite close, often differing by less than a percentage. However, we observed that this happened when the mutants killed by the test suite was already quite close to the manual estimation of killable mutants. We also found that in several cases, the results seemed program dependent. Hence we wanted to investigate this.

Furthermore, it may be argued that the prediction of killable mutants based on EVOSUITE on a particular configuration is simply the maximum number of mutants that EVOSUITE configuration can kill if given an infinite amount of time, and EVOSUITE may be unable to kill certain killable mutants due to the limitations of the generation strategy and oracles that EVOSUITE configuration is able to apply. Hence, the prediction from EVOSUITE configurations and prediction from manual classification may not coincide.

To mitigate this threat, we came up with an innovative method to repartition our segments. We observed that because we were testing Java libraries, all our unit test cases were written as corresponding to specific Java classes. Within each class there were methods that tested different functionalities of the class. That is, we could consider each unit test class as a single test, and recompute the frequencies for mutant detection, resulting in the same number of mutants killed.

Given that we are estimating the same value (the number of mutants that can be killed by the strongest test suite) we expect the estimates to coincide or at least overlap in their confidence intervals, perhaps with the class level estimators resulting in larger confidence intervals.

However, our results show that none of the estimators are robust when such regrouping is applied. The estimators neither coincide, nor their confidence intervals overlap. Furthermore, the predicted number of mutants is almost always more than the actual number of mutants generated.

In summary, this paper makes the following contributions:

- A data set of 1016 live mutants sampled from ten mature open-source projects and manually classified into equivalent and non-equivalent by three researchers.
- The first application of species diversity estimators to the problem of estimating equivalent mutatnts.
- A comparison of twelve estimators for population size and species size estimation applied to the problem of estimating killable mutants.
- A method to compare the performance of species and population estimators even when the ground truth is not available.
- A data set of test suites generated manually and automatically fully evaluated using mutation analysis.

II. BIOMETRICS STATISTICAL ESTIMATORS

In biometrics [14] (Ecology), statistical estimators have been successfully used to address questions such as (1) *"How does one estimate the number of individuals that belong to a particular species in a given geographical area?"* (i.e., population size estimation), and (2) *"How does one estimate the total number of species that can be found in a given geographical area?"* (i.e., species richness estimation).

These estimators from biometrics have also been successfully used in other areas such as computer networks [15] to estimate the size of an unknown population. These estimators were also suggested for use in the estimation of residual defects using the STADS framework [16], and in particular, in estimation of equivalent mutants [12]. However, the precise details of how to apply such statistical estimators to the problem of equivalent mutant estimation has not been spelt out. In this section, we examine the basic ideas of these statistical estimators; instead, we provide a mapping between the concepts used in the estimators to the domain of mutation analysis in the next section.

Chao [14] argues that the problem of population size estimation is an instance of the (more general) problem of species richness estimation, and it has supplanted population size estimation; therefore, in our research we consider only species richness estimators.

In Section III, we describe how we use species richness estimators to estimate how many *live* mutants, i.e., mutants that survived mutation analysis, are indeed "killable". In the next sections, instead, we summarize the necessary concepts about species richness estimation (see Section II-A) and introduce the estimators we considered in our study to make the paper self-contained (see Section II-B and Section II-C).

A. Species Richness Estimation

The problem of estimating species richness was first formulated in *biometrics*. Estimating species richness is challenging because the geographical area to consider is often extremely large, making an exhaustive survey infeasible. To cope with that, ecologists resort to *sampling*: (1) they divide the area into a number of smaller *segments*, also called *sampling units*, that can be surveyed comprehensively; (2) they randomly select a number of sampling units to survey; and, (3) they exhaustively

[2]https://www.evosuite.org/

survey the selected sampling unit for the number of species found.

Sampling data might show a different distribution of species to 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 of the surveyed sampling units. Hence, the problem of species richness estimation with (partial) sampling becomes the problem of estimating the total number of species that is present in the entire geographical area including the species that are not found in any of the sampling units.

The essential idea exploited by species richness estimators is that the number of species that did not show up in any sampling unit can be estimated by looking at the next rarest species (e.g., *singleton* species detected only in a single sampling unit, *doubleton* species detected only in two sampling units) and the amount of individuals that belong to them.

Chao [14] 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. We present instances of both kinds of estimators in the following sections.

B. Incidence Sampling Estimators

Incidence sampling considers n sampling units randomly selected among all the available ones and assumes their independent [14]. In each sampling unit, the relevant (categorical) data is the presence of various species; hence, incidence sampling does not consider the number of individual that belong to each species found. After surveying all the n sampling units, incidence sampling reports the count of species that appear only once (q_1), twice (q_2), and so on (q_m). Using these values, the estimators predicts q_0, i.e., the number of species that never appeared during sampling.

1) F1F2 (Chao) estimator: Chao [17] provides an estimator for estimation of population of a species based on the incidence or abundance data. The *Chao* estimator is given by

$$\hat{S}_{chao} = \begin{cases} S_{obs} + \frac{n-1}{n} F_1^2/(2F_2), & \text{if } F_2 > 0. \\ S_{obs} + \frac{n-1}{n} F_1(F_1-1)/2, & \text{otherwise.} \end{cases}$$

where S_{obs} is the number of species observed in the sample, and F_1 is the frequency of singleton species — f_1 if abundance data, and q_1 if incidence data. Similarly F_2 is the number of doubleton species which is f_2 if abundance data and q_2 if incidence data. S_1 is the total number of species estimated. When bias corrected, the formula becomes

$$\hat{S}_{chao\text{-}bc} = S_{obs} + \frac{n-1}{n} \frac{F_1(F_1-1)}{2(F_2+1)}$$

An improved version is given by

$$\hat{S}_{iChao} = S_{chao} + \frac{n-3}{n} \frac{F_3}{4F_4} \times \max\left[F_1 - \frac{n-3}{n-1} \frac{F_2 F_3}{2F_4}, 0\right]$$

Note that Chao estimators are *lower bounds* for the species richness.

2) Chao estimator with bias correction: Chao [14] provides a different estimator when the detection probabilities of every species is equal.

$$S_{Chao1-bc} = S_{obs} + [(n-1)/n] f_1(f_1-1)/[2(f_2+1)]$$

Note that Chao [14] provides two additional estimators – lowerbound estimator and upperbound estimator with minor differences.

3) Jackknife estimator: The Jackknife estimator developed by Burnham et al. [18], [19] provides an alternative for Chao estimator. The formula is given by

$$S_{jack1} = S_{obs} + F_1 \left(\frac{n-1}{n}\right)$$

where as before F_1 is the number of singletons, and n is the number of individuals in the sample for abundance data, and the number of samples for incidence data. A second order Jackknife [20] that is far more robust to sampling bias [21] (but with larger standard error) is

$$S_{jack2} = S_{obs} + \frac{2n-3}{n} F_1 - \frac{(n-2)^2}{n(n-1)} F_2$$

with F_1 and F_2 interpreted as before. The Jackknife estimators are known to underestimate when the sample size is small, and overestimate when one has a large sample size [14]

4) Coverage estimator: The ACE (abundance coverage) and ICE (incidence coverage) estimators were developed for cryptographic analysis by Good and Turing. The formula is given by:

$$S_{ACE} = S_{abun} + \frac{S_{rare}}{\hat{C}_{rare}} + \frac{F_1}{\hat{C}_{rare}} \hat{\gamma}^2_{rare}$$

$$\hat{\gamma}^2_{rare} = \max\left[\frac{S_{rare}}{\hat{C}_{rare}} \frac{\sum_{i=1}^{k} i(i-1)F_i}{\left(\sum_{i=1}^{k} iF_i\right)\left(\sum_{i=1}^{k} iF_i - 1\right)} - 1, 0\right]$$

where S_{common} is species that occur more than k times, and S_{rare} occurs k times or less. The coverage estimators (*ACE* and *ICE*) generally under estimate, but the underestimation are corrected with the *ACE-1* and *ICE-1* estimators [14].

5) Bootstrap estimator: The Bootstrap estimator was described by Smith and van Belle [20] who gave the formula for Bootstrap estimate. The estmate is based on sampling n quadrats for which data is available. Given a sample of size n taken with replacement, the Bootstrap estimate of $S_{bootstrap}$ which is the estimated species count is given by

$$S_{bootstrap} = S_0 + \sum_{j=1}^{S_0} (1 - Y_j/n)^n$$

where Y_j is the number of quadrats where species j is detected.

6) Zelterman estimator: The Zelterman's estimate [22] provides the following formula for estimation of the missing zero count species.

$$\hat{f}_0^Z = n/[exp(\hat{\lambda}) - 1]$$

where $\hat{\lambda} = 2f_2/f_1$, and $n = f_1 + f_2 + ..f_m$ where m is the largest observed count.

C. Abundance Sampling Estimators

Abundance sampling, instead, works only on one ($n = 1$) sampling unit and considers as the relevant (numerical) data the count of individuals that belong to each species. After exhaustively surveying the sample unit, abundance sampling reports the count of species (f_k) for which the same number of individuals (k) has been found for various values of k. Using these values, the estimators predict f_0, the number of species for which sampling found no individuals, i.e., undetected species.

The second is the so called *abundance* sampling. In this kind of sampling, we sample n individuals from the population (the assumptions are that the samples are independent and random, but as the sampling in biometrics is rarely random or independent, the estimators are considered robust against violations of these assumptions [11]). From this sample of individuals, we tally the species of *each individual*. A species is called a *singleton* if there was only a single individual of that species in the sample. Similarly a *doubleton* if two individuals of that species were present in the sample. The number of singleton species is designated as f_1, and the number of doubleton species is designated as f_2 etc. The estimators try to estimate the number of f_0 species — that is the species that were not seen in the sample.

We next describe the **abundance sampling** estimators we used. Given the relatively low performance of these estimators even in biometrics, we only provide a brief outline.

1) Species Chao-Bunge: The Chao-Bunge model for species diversity was given by Chao and Bunge in 2002 [23] The formula for the estimator is given by

$$\hat{N} = D_a + \sum_{k=2}^{r} f_k/(1 - \frac{f1 \sum_{k=1}^{r} k^2 fk}{(\sum_{k=1}^{r} kfk)^2})$$

where D_a which is the number of observed species in the sample.

2) Species Unpmle: Species UNPMLE (Unconditional nonparametric maximum likelihood estimator) was provided by Norris and Pollock [24].

3) Species PNPMLE: Species PNPMLE (Penalized nonparametric maximum likelihood estimator) was provided by Wang and Lindsay [25].

4) Species PCG: Species PCG (Poisson-compound Gamma model with smooth nonparametric maximum likelihood estimation) was provided by Wang and Ji-Ping [26].

III. Mapping from biometrics to mutation analysis

We consider each nonequivalent mutant as a separate species, such that the estimation of species diversity results in the estimation of detectable mutants.

A. Mapping incidence sampling models

We consider a test case (either the test method or the test class) to be one sampling unit. Then, we collect the mutants detected by the test suite, and compute those mutants that were detected by a single test case (i.e. in one sampling unit), and those that were detected by two test cases etc.

B. Mapping abundance sampling models

We consider the test suite to be one big sample.u Next, we count each mutant detection by a test case as an individual of the species of that mutant. This provides us with the number of mutants that were detected by a single test case (singletons), those that were detected by two test cases etc.

C. Classes as logical test units

The idea here is that we consider each *test class* as a unit for testing. That is, irrespective of the number of methods in a test class that detected a mutant, we count only the unique number of classes that detected a mutant. For example, we reduce the results of the form ($m1$ killed by $T_1.ta$, $T_1.tb$, $T_2.tp$) where T_1,T_2 are classes and ta, tb, tp are methods in corresponding classes to ($m1$ killed by T_1, T_2) which ignores the test methods. We denote the set of tests thus derived as *class test set*.

D. Methods as logical test units

Here, a mutant is mapped to a single species. A logical unit is a single test method, and an individual in the population is an execution of a test method on a mutant.

IV. Methodology

We now detail the methodology followed for evaluating the statistical estimators introduced in Section II applied to the problem of equivalent mutants estimators as described in Section III.

A. Overview

The goal of our study is to understand whether statistical estimators for species richness estimation can provide actionable information to the mutation analysis practitioner for interpreting mutation analysis results. Hence, we looked for large, well-maintained, and well-tested open-source projects. We believe that such projects closely match the industrial projects where there is a high emphasis on adequate testing, and the question of whether live mutants are killable or equivalent is of high importance.

Our methodology consists of the following steps: (1) project selection and filtering (Section IV-B); (2) mutants generation and identification of live mutants (Section IV-C) to enable (3) manual classification of equivalent mutants (Section IV-D); (4) test suite generation (Section IV-E); and, (5) "killable" mutants estimation (Section IV-F).

The results from statistical estimators are evaluated in the following ways: 1) In comparison with estimates from manual classification. 2) Estimates from different test suites are compared against each other. 3) Estimates from method level estimators are compared against class level estimators.

B. Projects Selection and Filtering

To achieve a balance between generality and applicability of our findings, and to avoid uncontrolled variables, we focus on large open-source *Java* projects that do not depend on external resources (e.g., databases, backend servers) and use a standard build framework (i.e., *Apache Maven* [27]).

The libraries published by *Apache Commons* [28] fulfill our criteria. Those libraries focus on various aspects of reusable Java components. Those projects are large, mature, and well-maintained and must meet the quality criteria defined by the Apache foundation. At the time we conducted this study, Apache Commons contained the 41 projects reported in Table I. Among those, for our study, we selected only those that (1) could be built and tested successfully with minimal manual effort; (2) successfully completed mutation analysis; (3) have more than one release; and, (4) have a flat structure, i.e., they are packaged as a single module.

In Table I, we group the Apache Commons projects into two groups: the first group (top) contains the 22 projects that we could build, test, and analyze successfully, whereas the second group (bottom) contains the remaining projects. We further divide the latter group into two sub-groups: the group of "immature" projects (middle) that contains 3 projects that did not have any release at the time we conducted this study, and the group of projects that failed to meet our selection criteria (bottom). This last group contains 17 projects that either have a complex structure, fail to build or test, or break mutation analysis, although we put some effort into fixing existing problems.

Our methodology requires manual live mutants classification, which is notoriously expensive. Therefore, we had to restrict the scope of our study and limit the number of projects to consider. Specifically, we chose the 10 largest projects among the 22 selected from Apache Commons. We highlight the chosen projects in Table I and summarize their main properties in Table II.

TABLE I: List of considered Apache Commons projects

Project	Release	Commit Hash/Main Rejection Reason
commons-beanutils	1.9.4-RC2	32ceb2c92512d44f97638805e2f3fd9d70dfcfc6
commons-cli	1.4	f7153c3c10cc8060d3858a6a90d7ddccd0b3533e
commons-codec	1.13	beafa49f88be397f89b78d125d2c7c52b0114006
commons-collections	4.4	cab58b3a8093a2f6b84f12783a3fb358747310f7
commons-compress	1.18	b95d5cde4c68640f886e3c6802384fae47408a37
commons-configuration	2.5	dc00a04783ea951280ba0cd8318f53e19acb707f
commons-csv	1.7	a227a1e2fb61ff5f192cfd8099e7e6f4848d7d43
commons-dbcp	2.6.0	3e7fca08d3585aa7cf70045ee3ed607cbaf04baa
commons-dbutils	1.7	77faa3caefc24697b00bf370111f7307dae339b0
commons-digester	3.2	ec75748096c8639be7b4998d99c37bb2e3a9c206
commons-email	1.5	5516c487a5d03df520480d47f5c5ae5a3b3341d4
commons-exec	1.3	0b1c1ff0cba7145d396ace37805f0b59b3d52a9c
commons-fileupload	1.4	047f31576411beee69cf75584ae76531cc9ac753
commons-functor	1.0_RC1	62cd20998e706eafb48b35774aca7b6e1c657386
commons-imaging	1.0-alpha1-RC3	6f04ccc2cf8c867298579c355c6d88fd74da3e7b
commons-io	2.6-RC3	2ae025fe5c4a7d2046c53072b0898e37a079fe62
commons-lang	3.8_RC1	9801e2fb9fcbf7ddd19221e9342608940d778f8c
commons-math	3.6_RC2	95a9d35e77f70ffc9bd5143880c236a760b42005
commons-net	3.6	163fe46c019f5184016207c247cdff30ee740ecc
commons-pool	2.7.0	f4455dcb8afaf9ae7054589110f1082a7a8a282c
commons-validator	1.6	c4b93a7275b1172f69e53cd8cec7fc27c954a5be
commons-ognl	no releases	immature project
commons-geometry	no releases	immature project
commons-testing	no releases	immature project
commons-bcel	6.3.1	error during build
commons-vfs	2.4	complex structure and failures during build
commons-text	1.7	Error too many unapproved licenses
commons-logging	1.2	JVM crash during mutation
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	JVM crash
commons-jxpath	1.3	JVM crash
commons-scxml	2.0-M1	JVM Hangs
commons-rng	1.2	complex structure and errors during build
commons-rdf	0.5.0	complex structure and errors during build
commons-proxy	2.0 RC1	cyclic reference during build
commons-bsf	3.x-with-engines	complex structure and errors during build
commons-weaver	2.0	complex structure and errors during build
commons-jelly	1.0.1	errors during build
commons-jci	1.1	complex structure and errors during build

C. Mutants Generation and Live Mutants Identification

For mutation analysis, we used *PIT* [29] (v 1.4.9), with its *default*[3] mutation operators and the *killmatrix* option enabled. Since our study involves multiple test suites for each project, we will refer to the official test suite as the ORIGINAL test suite. Additionally, since mutation analysis is almost always used with unit test cases, we evaluated species richness estimators using unit tests only. We identified *live* mutants as those reported by PIT as `SURVIVED` after completing the mutation analysis of ORIGINAL. We then sampled those live mutants for the manual classification as described next.

D. Methodology for Live Mutant Classification

In our study, assessing the accuracy of species richness estimators requires the availability of ground truth data about equivalent mutants, which is not generally available. To ensure highly accurate data, we used manual classification (see results in Table III) and involved three researchers, all co-authors of this paper. The lead researcher (R_A) has ten years of experience in programming with Java and is deeply familiar with mutation analysis research. The other two researchers (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 using *commons-csv*, the smallest among the selected projects. Commons-csv's ORIGINAL test suite killed 78.11% mutants and left 116 mutants alive; we used all of those live mutants to train R_B and R_C.

We conducted the equivalent mutant classification between July 2019 and December 2019 (six months) and processed the projects in alphabetical order (as listed in Table III) one at the time. For the classification we adopted the following protocol: First, we randomly sampled 100 live mutants; we argue that classifying that many mutants is a good balance between classification effort (between one and thirty minutes per mutant) and representativeness of the obtained results. Next, we independently asked researchers R_B and R_C to classify live mutants as killable or equivalent. Killable mutants must be accompanied by a (hypothetical) unit test that could show the difference in behavior between that mutant and the original program. Likewise, equivalent mutants must be motivated with a convincing explanation. The result of the classification includes the particular mutant identifier, the label (killable or equivalent) along with how confident the researcher was in the classification (high, medium, low), and optional comments. Finally, we identified conflicting classifications (avg 4.9%) and let the researchers discuss and resolve them. During the discussion, the lead research R_A acted as the moderator while R_B and R_C could update their original classifications in light of the discussed argument. The discussion continued until the final classification was accepted unanimously.

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

TABLE II: Apache Commons projects — Projects and test suites measures

Project	LOC	Mutants	ORIGINAL				RANDOM				DYNAMOSA			
			Size	Stmt	Branch	% Kill	Size	Stmt	Branch	% Kill	Size	Stmt	Branch	% Kill
commons-collections	28955	8309	25011	86	81	75.21%	3954	67	59	57.30%	5659	74	69	93.41%
commons-compress	24015	9520	1047	84	76	65.41%	2610	63	45	44.31%	4306	70	57	49.33%
commons-configuration	28021	6163	2803	87	83	99.04%	2991	75	62	38.68%	4956	80	72	39.96%
commons-csv	1845	635	312	89	85	81.10%	365	78	63	59.53%	647	89	83	75.75%
commons-dbcp	14387	4184	1409	66	66	99.74%	1026	39	42	19.60%	2970	47	58	26.64%
commons-imaging	31320	11577	563	73	59	46.66%	2974	63	45	44.16%	4935	67	53	45.02%
commons-io	9984	3243	1316	90	88	94.13%	1677	72	65	58.20%	2822	81	79	86.68%
commons-lang	27646	13054	4114	95	91	65.66%	4500	73	60	71.64%	9027	89	86	86.47%
commons-math	100366	47876	6377	92	84	75.59%	11956	74	61	76.63%	17450	79	68	17.63%
commons-net	20053	5694	254	33	28	27.86%	1988	48	35	31.99%	3499	51	42	34.91%

TABLE III: Manual classification of live mutants. Misclassifications are reported under MER (Misclassification Error Rate).

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

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

E. Methodology for Test Generation

The test suite used for the mutation analysis defines the particular parts of the code, hence the mutants that can be covered and killed. That is, the test suite defines how live mutants are sampled. Because the ORIGINAL test suites contain unit tests that developers 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. This, in turn, would negatively affect the effectiveness of statistical estimators as they usually rely on independence and randomness assumptions.[1]

To reduce this risk and study species richness estimators more deeply, we generated two additional test suites for each of the selected projects using well-established techniques (see Table II). Specifically, we leveraged *EvoSuite* [30], to generate the RANDOM test suite, which does not target any specific testing goal, and the DYNAMOSA test suite, which aims to maximize coverage using the Dynamic Many-Objective Sorting Algorithm [31]. RANDOM has the lowest generation bias but might not be highly effective, whereas DYNAMOSA is more effective but might introduce bias being not completely random.

For generating both test suites, we followed the best practice set by the SBST unit testing tool competition [32]–[34] and let EVOSUITE generate unit tests for each of the classes in the selected projects for 60 seconds. We also allotted 120 sec for generating assertions and 300 sec for minimizing the tests.

EVOSUITE implements advanced techniques such as hybrid search, dynamic symbolic execution, weak mutation analysis, and testability transformation [35] as well as automated mock generation [36], to produce robust and effective test cases. Nevertheless, it occasionally fails to create valid (compilable and passing) unit tests and might also generate *flaky* tests [37]. Since those problematic tests prevent mutation analysis from producing reliable results, when we observed their occurrence, we eliminated them from RANDOM and DYNAMOSA.

F. Methodology for Estimation

As discussed in Section III, the application of species richness estimators for equivalent mutant estimation requires data about mutants (i.e., ID, description, location), test cases (i.e., name of test class and test methods), and test case executions (i.e., failed/passed test, name of killing test classes and test methods). Therefore, we executed all the available test cases against all the generated mutants for each project.

This epic effort (more than 2.5B test executions) was necessary to cope with some limitations of PIT that did not always identify all the mutants covered by the test cases; hence, it did not execute them. Additionally, because PIT did not always report the name of the test methods killing mutants, which is required to sample data using tests methods as sampling units, we resorted to using a different mutation testing infrastructure. Specifically, we chose *JUGE* [38], the open-source infrastructure developed for the SBST Java unit test competition, and extended it to (1) correctly report test method names and (2) parallelize the test executions across multiple computing nodes (we used a cluster of 50 nodes featuring heterogeneous hardware configurations). Noticeably, we configured JUGE to use the same PIT version used in the manual classification, hence we ensured to consider the same mutants across all the steps of our study.

After executing the tests, we used automated scripts to collect the required data from the various computing nodes and filter out observations related to invalid mutants (e.g., errors in loading classes, memory errors, JVM crashes) and mutants causing timeouts, [4] to reduce the risk of misunderstanding live, possibly "slow" mutants as trivially killed mutants. We

[1] **TODO** *This probably needs a more precise term*

[4] We conservatively granted each unit test a generous timeout of 10 seconds.

make available the entire dataset and the scripts to analyze test execution data in the replication package [**?**].

After collecting the necessary data, we estimated the live mutants species richness on the ten projects using all twelve estimators on each of the three test suites using both (test) classes and (test) methods as sampling units. This resulted in a whopping 720 estimates (Figure 3) that we evaluated against the results of the manual classification, as we describe in the next section.

V. Results & Discussion

We answered the following research question:

RQ1. Which statistical estimator (method level estimators, using ORIGINAL test suite) best predicts the ground truth (estimates from manual classification)?

To answer this question, we statistically determined the 95% confidence intervals of each estimator where each test method was considerd to be a single test case for the ORIGINAL test suite. We then searched for those estimators whose 95% confidence intervals overlapped with the 95% confidence intervals from manual classification consistently. From these, we wanted to identify those estimators that differed the least from manual classification. On inspection, we found that none of the statistical estimators had 95% confidence intervals that overlapped with 95% confidence intervals from manual classification consistently. Given that they do not overlap, the difference between the estimates and manual classification is *statistically significant*. Furthermore, the difference is larger than expected[2].

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

This suggested only two possibilites: (1) that the manual classification was in error, or (2) the estimate from ORIGINAL test suite was in error. While the manual classification could have errors, the likelyhood is limited considering that each classified mutant was verified by three researchers. On the other hand, a possible source of error for (2) is that the ORIGINAL test suites were biased due to manual creation. This can mean that the estimates from ORIGINAL test suite may have bias problems too.

To investigate this threat, we replace the ORIGINAL test suite with two test suites generated with EVOSUITE– (1) RANDOM and (2) DYNAMOSA. Our hypothesis here is that, by eliminating human bias, we would have an unbiased estimate that is closer to the ground truth.

RQ2. Which statistical estimator best predicts the ground truth (estimates from manual classification) when given unbiased test suites from EVOSUITE? To anseer this question, we again statistically determined the 95% CI from each estimator where each test method was considered a single test case for each test suite from EVOSUITE. We then searched for estimators where there was a consistent overlap between CIs from manual classification and estimates.

We again found that there was no single estimator that provided a consistent overlap between manual classification and the estimate. Furthermore, we also found that none of the estimators provided a consistent overlap between the estimates from both test suites.

That is, there is statistically significant difference between the estimates from different EVOSUITE test cases and manual classification, and between themselves. Furthermore, the difference is larger than expected[3].

> There difference between estimates from RANDOM and DYNAMOSA to the estimates from manual classification are statistically significant for every estimator examined. Their difference between each other is also statistically significant for every estimator examined.

Our results suggested a third possibility. It is possible that our mapping of a sampling unit to a single test method was wrong. To test this, we mapped the sampling unit to a test class. With this mapping, we had a complementary set of estimations to check.

RQ3. Which mapping of sampling units best predicts the ground truth (estimates from manual classification) when given unbiased test suites from EVOSUITE?

On comparing the confidence intervals from each of the estimators for class level sampling units to estimates from manual classification, we once again found that there is a statistically significant difference between the two. Furthermore, the estimates from method level sampling units and class level sampling units are also statistically significantly different.

> There difference between estimates from method level estimators and class level estimators to the estimates from manual classification are statistically significant for every estimator examined. Their difference between each other is also statistically significant for every estimator examined.

Given that class level sampling units are predicting the same quantity as that of the method level sampling units, this suggests the only remaining possibility. That the statistical estimation using species richness models is not applicable to the problem of equivalent mutant estimation.

A. What does this mean in practice?

One of the uses of mutation analysis is that it can be used to estimate the residual defect density [39]. That is, an accurate estimate of the remaining live mutants can provide evidence on the bugs that remains to be found. Similarly, STADS [16] is a framework that relies on the same species diversity estimates to provide accurate estimates on residual defect density. Our negative results from applying species discovery estimators to mutation analysis suggests that further study is needed before one can rely on species diversity estimators for estimating

[2] **TODO** *provide mean differences*

[3] **TODO** *provide mean here*

Fig. 3: method test data, with test suites indicated by color. ORIGINAL is red, RANDOM is blue and DYNAMOSA is green. Dashed lines represent killed mutants by respective test cases, and error bars represent estimates from mutant kills by each estimator. The solid black line is the total mutants generated, dashed black line the manual estimate, and gray dashed lines the error bars.

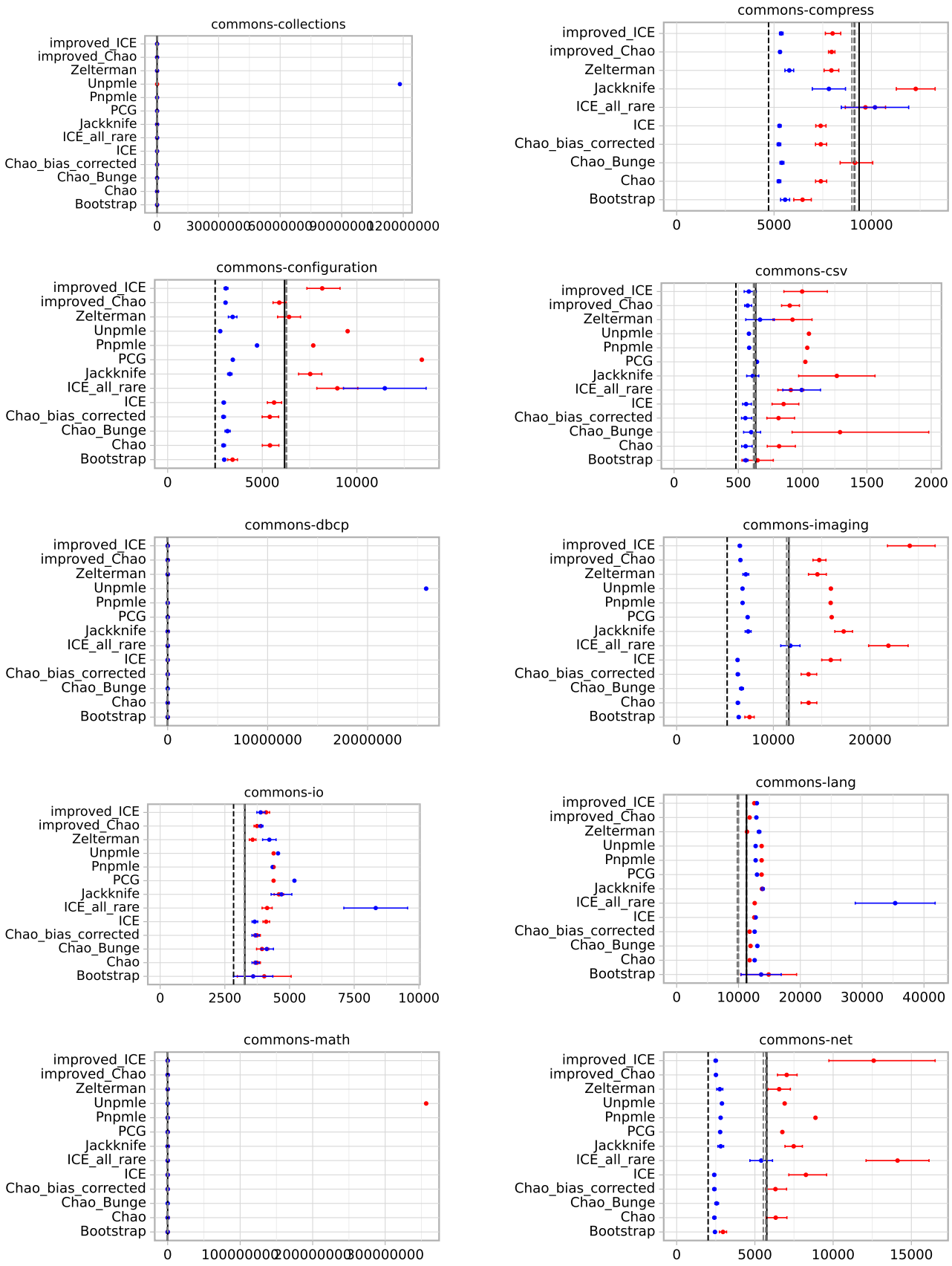

Fig. 4: Estimators from test DynaMOSA. The red is from *class test set* data, and blue from *method test set* data.

residual defects. Our recommendation to the testers is to continue to rely on sampling of a subset of live mutants for estimating the number of killable mutants within the required confidence interval.

Beyond mutation analysis, ours is also one of the first studies that evaluates the STADS framework [16] for estimation. A comprehensive evaluation in other fields such as residual defect estimation is difficult as it is very hard to get the ground truth on the number of defects a large program may still contain. Mutation analysis is a useful test bed for STADS because we have an upper bound on the number of killable mutants – the total number of mutants generated. However, we note that in many cases, the estimated number of killable mutants from class level estimators (including the error bars) lie beyond the total number of mutants generated.

Given the dismal performance of STADS on estimating the number of killable mutants, we believe that further empirical evidence and detailed analysis is needed to understand various assumptions made by species diversity models, and whether they are applicable to specific domains before they can be reliably used by the practitioners.

B. 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 (e.g., random) and well established and effective algorithms (e.g., DynaMOSA), 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 misclassifications 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.

Construct Validity. We are the first to apply species richness estimators to the problem of equivalent mutants; hence, our mapping of biometrics estimators to the mutation testing domain might not to capture important variables.

VI. Related Work

Mutation analysis is considered a primary way of evaluating test quality [1]; thus, mutation score is usually considered as a test suite adequacy metrics [2]–[5]. Unfortunately, equivalent mutants have vexed practitioners from the very beginning [7], and remains an open issue [40]. To address this issue, we proposed to estimate killable mutants using species richness estimators 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 classification of equivalent mutants. Acree's study [41] involved two competent software engineers experts in mutation analysis that classified[4] 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

[4] **TODO** *Can we report HOW MANY mutants they considered?*

documented various misclassifications (avg. 23%), suggesting that even manual classification may not be error-free. We also found misclassifications (see Table III, *Faults* column), but achieved better accuracy (less than 5% misclassifications on avg),[5] arguably because we trained the labelers and followed a structured and systematic classification protocol.

Other studies of note are by Yao et al. [42] on 18 C programs, and Grün et al. [10] (extended by Schuler et al. [43]) 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 [44]. 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.

We used statistical estimators for predicting killable mutants, others, instead, used them for estimating residual faults. For instance, Böhme [16] argued to use the same species richness estimators we studied for estimating residual defect density, while Nayak [45] and Voas and McGraw [46] modeled faults and residual defects as members of unknown populations and estimated their number via *capture-recapture* methods. Tohma et al. [47], instead, modeled the distribution of observed faults as hyper-geometric distribution to estimate the number of residual defects.

To tackle the problem of estimating equivalent mutants, other methods than statistical estimators have been proposed. For instance, Vincenzi et al. [48] proposed to estimate the (posterior) probability that specific mutation operators generate equivalent mutants, while Marsit et al. [49]–[51] 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.

VII. Conclusions

Equivalent mutants are one of the major stumbling blocks when using mutation score as an adequacy measure. The existing techniques to identify equivalent mutants are ineffective and identify only a limited number of such mutants. Consequently, the problem of accurately measuring mutation score remains open. Recent research advocates the use of statistical methods for estimating the number of equivalent mutants by modeling them as individuals of a (unknown) population. However, such biometrics estimators have never been evaluated empirically.

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

The research presented in this paper fills this gap by providing the first, large evaluation of these estimators on mutation analysis. First, we manually classified a statistically significant sample of live mutants and used it 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) class and (test) method granularity. Finally, we compared the resulting 720 estimates against the ground truth to measure the quality of species richness estimators on ten real-world projects across projects and test suites. Instead, 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 statistical estimators for species richness applied to the equivalent mutants problem do not produce consistent results across projects, test suites, and when computed using different sampling strategies. Therefore, they have way to go before they become truly usable by the practitioners.

References

[1] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon, and M. Harman, "Mutation testing advances: an analysis and survey," in *Advances in Computers*. Elsevier, 2019, vol. 112, pp. 275–378.

[2] M. Daran and P. Thévenod-Fosse, "Software error analysis: A real case study involving real faults and mutations," *ACM SIGSOFT Software Engineering Notes*, vol. 21, no. 3, pp. 158–171, 1996.

[3] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser, "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*, 2014, pp. 654–665.

[4] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?" in *Proceedings of the 27th international conference on Software engineering*, 2005, pp. 402–411.

[5] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using mutation analysis for assessing and comparing testing coverage criteria," *IEEE Transactions on Software Engineering*, vol. 32, no. 8, pp. 608–624, 2006.

[6] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE transactions on software engineering*, vol. 37, no. 5, pp. 649–678, 2010.

[7] T. A. Budd and D. Angluin, "Two notions of correctness and their relation to testing," *Acta informatica*, vol. 18, no. 1, pp. 31–45, 1982.

[8] H. G. Rice, "Classes of recursively enumerable sets and their decision problems," *Transactions of the American Mathematical Society*, vol. 74, no. 2, pp. 358–366, 1953.

[9] A. J. Offutt and W. M. Craft, "Using compiler optimization techniques to detect equivalent mutants," *Software Testing, Verification and Reliability*, vol. 4, no. 3, pp. 131–154, 1994. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.4370040303

[10] B. J. Grün, D. Schuler, and A. Zeller, "The impact of equivalent mutants," in *2009 International Conference on Software Testing, Verification, and Validation Workshops*. IEEE, 2009, pp. 192–199.

[11] N. J. Gotelli and R. K. Colwell, "Estimating species richness," *Biological diversity: frontiers in measurement and assessment*, vol. 12, pp. 39–54, 2011.

[12] M. Böhme, "Assurances in software testing: A roadmap," *CoRR*, vol. abs/1807.10255, 2018. [Online]. Available: http://arxiv.org/abs/1807.10255

[13] R. Gopinath, P. Görz, and A. Groce, "Mutation analysis: Answering the fuzzing challenge," 2022. [Online]. Available: https://arxiv.org/abs/2201.11303

[14] A. Chao and C.-H. Chiu, *Nonparametric Estimation and Comparison of Species Richness*. American Cancer Society, 2016, pp. 1–11. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470015902.a0026329

[15] N. Accettura, G. Neglia, and L. A. Grieco, "The capture-recapture approach for population estimation in computer networks," *Computer Networks*, vol. 89, pp. 107–122, 2015.

[16] M. Böhme, "Stads: Software testing as species discovery," *ACM Transactions on Software Engineering and Methodology*, vol. 27, no. 2, 7 2018.

[17] A. Chao, "Nonparametric estimation of the number of classes in a population," *Scandinavian Journal of statistics*, pp. 265–270, 1984.

[18] K. P. Burnham and W. S. Overton, "Estimation of the size of a closed population when capture probabilities vary among animals," *Biometrika*, vol. 65, no. 3, pp. 625–633, 1978. [Online]. Available: http://www.jstor.org/stable/2335915

[19] ——, "Robust estimation of population size when capture probabilities vary among animals," *Ecology*, vol. 60, no. 5, pp. 927–936, 1979. [Online]. Available: http://www.jstor.org/stable/1936861

[20] E. P. Smith and G. van Belle, "Nonparametric estimation of species richness," *Biometrics*, vol. 40, no. 1, pp. 119–129, 1984. [Online]. Available: http://www.jstor.org/stable/2530750

[21] J. Hortal, P. A. Borges, and C. Gaspar, "Evaluating the performance of species richness estimators: sensitivity to sample grain size," *Journal of animal ecology*, vol. 75, no. 1, pp. 274–287, 2006.

[22] D. Böhning, "Some general comparative points on chao's and zelterman's estimators of the population size," *Scandinavian Journal of Statistics*, vol. 37, no. 2, pp. 221–236, 2010.

[23] A. Chao and J. Bunge, "Estimating the number of species in a stochastic abundance model," *Biometrics*, vol. 58, no. 3, pp. 531–539, 2002.

[24] J. L. Norris and K. H. Pollock, "Non-parametric mle for poisson species abundance models allowing for heterogeneity between species," *Environmental and Ecological Statistics*, vol. 5, no. 4, pp. 391–402, 1998.

[25] J.-P. Z. Wang and B. G. Lindsay, "A penalized nonparametric maximum likelihood approach to species richness estimation," *Journal of the American Statistical Association*, vol. 100, no. 471, pp. 942–959, 2005.

[26] J.-P. Wang, "Estimating species richness by a poisson-compound gamma model," *Biometrika*, vol. 97, no. 3, pp. 727–740, 2010.

[27] Apache Software Foundation, "Apache Maven," https://maven.apache.org/.

[28] ——, "Apache Commons," http://commons.apache.org/.

[29] H. Coles, "Pit - real world mutation testing," https://pitest.org.

[30] G. Fraser and A. Arcuri, "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*, 2011, pp. 416–419.

[31] A. Panichella, F. M. Kifetew, and P. Tonella, "Reformulating branch coverage as a many-objective optimization problem," in *IEEE International Conference on Software Testing, Verification and Validation*. IEEE, 2015, pp. 1–10.

[32] X. Devroey, S. Panichella, and A. Gambi, "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, 2020, pp. 545–548. [Online]. Available: https://doi.org/10.1145/3387940.3392265

[33] S. Panichella, A. Gambi, F. Zampetti, and V. Riccio, "SBST tool competition 2021," in *14th IEEE/ACM International Workshop on Search-Based Software Testing, SBST 2021, Madrid, Spain, May 31, 2021*. IEEE, 2021, pp. 20–27. [Online]. Available: https://doi.org/10.1109/SBST52555.2021.00011

[34] A. Gambi, G. Jahangirova, V. Riccio, and F. Zampetti, "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, 2022, pp. 25–32. [Online]. Available: https://doi.org/10.1145/3526072.3527538

[35] G. Fraser and A. Arcuri, "Whole test suite generation," *TSE*, vol. 39, no. 2, pp. 276–291, 2013.

[36] A. Arcuri, G. Fraser, and J. P. Galeotti, "Generating TCP/UDP network data for automated unit test generation," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*. ACM, 2015, pp. 155–165. [Online]. Available: https://doi.org/10.1145/2786805.2786828

[37] Z. Fan, "A systematic evaluation of problematic tests generated by evosuite," in *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, J. M. Atlee, T. Bultan, and J. Whittle, Eds. IEEE / ACM, 2019, pp. 165–167. [Online]. Available: https://doi.org/10.1109/ICSE-Companion.2019.00068

[38] X. Devroey, A. Gambi, J. P. Galeotti, R. Just, F. M. Kifetew, A. Panichella, and S. Panichella, "JUGE: an infrastructure for benchmarking java unit test generators," *CoRR*, vol. abs/2106.07520, 2021. [Online]. Available: https://arxiv.org/abs/2106.07520

[39] A. Zeller, R. Gopinath, M. Böhme, G. Fraser, and C. Holler, "Mutation analysis," in *The Fuzzing Book*. CISPA Helmholtz Center for Information Security, 2022, retrieved 2022-01-11 09:26:37+01:00. [Online]. Available: https://www.fuzzingbook.org/html/MutationAnalysis.html

[40] L. Madeyski, W. Orzeszyna, R. Torkar, and M. Józala, "Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation," *IEEE Transactions on Software Engineering*, vol. 40, no. 1, pp. 23–42, 2014.

[41] A. T. Acree Jr, "On mutation," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, Georgia, 1980, gIT-ICS-80/12.

[42] X. Yao, M. Harman, and Y. Jia, "A study of equivalent and stubborn mutation operators using human analysis of equivalence," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 919–930.

[43] D. Schuler and A. Zeller, "(un-) covering equivalent mutants," in *2010 Third International Conference on Software Testing, Verification and Validation*. IEEE, 2010, pp. 45–54.

[44] R. Gopinath, I. Ahmed, M. A. Alipour, C. Jensen, and A. Groce, "Mutation reduction strategies considered harmful," *IEEE Transactions on Reliability*, vol. 66, no. 3, pp. 854–874, 2017.

[45] T. Nayak, "Estimating population size by recapture sampling," *Biometrika*, vol. 75, 03 1988.

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

[47] Y. Tohma, K. Tokunaga, S. Nagase, and Y. Murata, "Structural approach to the estimation of the number of residual software faults based on the hyper-geometric distribution," *IEEE transactions on software engineering*, vol. 15, no. 3, pp. 345–355, 1989.

[48] A. Vincenzi, E. Nakagawa, J. Maldonado, M. Delamaro, and R. Romero, "Bayesian-learning based guidelines to determine equivalent mutants." *International Journal of Software Engineering and Knowledge Engineering*, vol. 12, pp. 675–689, 12 2002.

[49] I. Marsit, M. N. Omri, and A. Mili, "Estimating the survival rate of mutants," in *ICSOFT*, 2017.

[50] I. Marsit, M. N. Omri, J. Loh, and A. Mili, "Impact of mutation operators on mutant equivalence." in *ICSOFT*, 2018, pp. 55–66.

[51] A. Ayad, I. Marsit, J. Loh, M. N. Omri, and A. Mili, "Estimating the number of equivalent mutants," in *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2019, pp. 112–121.

APPENDIX

A. ORIGINAL

B. RANDOM

C. Model

Here, we describe the model that we used to map species diversity estimations to mutation analysis. Assume that we have S mutants and the testsuite has T tests. We also have a *kill incidence matrix*[6] (W_{ij}) with S rows and T columns, where an element W_i contains 1 if the mutant i was killed by the test j, and 0 otherwise. The row sum of the incidence matrix $Y_i = \sum_{j=1}^{T} W_{ij}$ denotes the incidence-based frequency of the mutant i, $i = 1, 2, \ldots, S$.

Let Q_k denote the incidence-based frequency counts, i.e., the number of mutants that are detected by exactly k tests, $k = 0, 1, \ldots, T$. The total number of killed mutants is S_{obs}.

[6] which mutants were killed by which tests

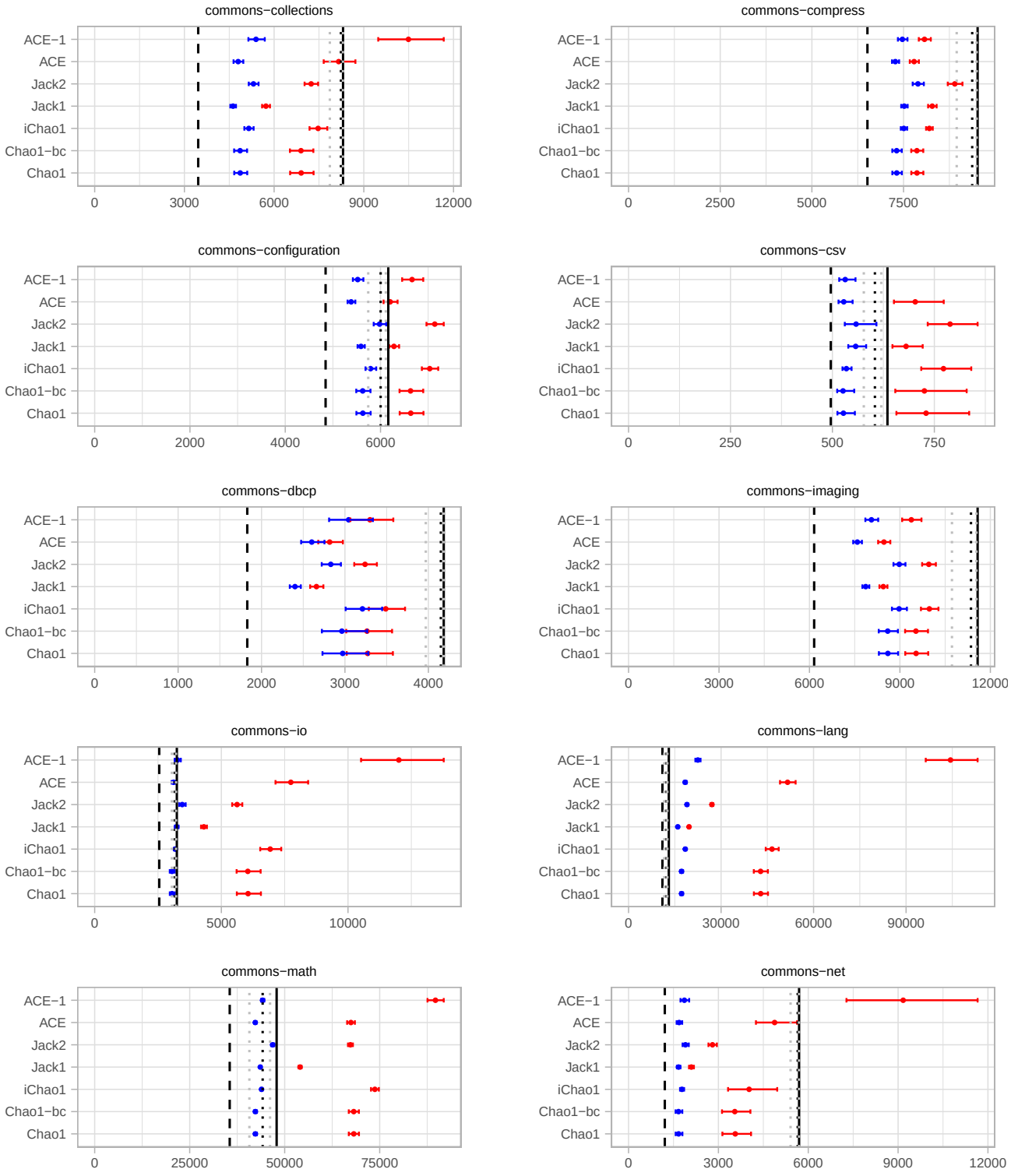

Fig. 5: Estimators from test ORIGINAL. The red is from *class test set* data, and blue from *method test set* data.

We assume that the detection probability of the i-th mutant by the j-th test is affected by two sources of heterogeneity. First, each mutant has its own unique kill rate π. Second, mutant detection by the j-th test ν_j may be affected by factors specific to that test: environment settings, input data, flakyness, etc.

Our model assumes that the detection probability of the i-th mutant in the j-th test is the product of the 'heterogeneity' effect and the test effect: $\pi_i \nu_j$. All factors that are involved in the test effects, $\{\nu_1, \nu_2, \ldots, \nu_T\}$ are modeled as random variables taken from an unknown probability density function $h(\nu)$.

Here, we regard the mutant effects $\pi_1, \pi_2, \ldots, \pi_S$ as fixed parameters, but treat the test effects $\nu_1, \nu_2, \ldots, \nu_T$ as random variables. The conditional probability distribution of W_{ij} given ν_j is a Bernoulli random variable with probability of success $\pi_i \nu_j$. That is,

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

The marginal distribution for the incidence-based frequency Y_i for the i-th mutant follows a binomial distribution:

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

where $\lambda_i = \pi_i \int \nu h(\nu) d\nu$. The frequency Y_i is a binomial random variable with detection probability ${\lambda_i}^{y}_{i}$.

D. 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.