STADS: Software Testing as Species Discovery

MARCEL BÖHME, National University of Singapore and Monash University, Australia

A fundamental challenge of software testing is the statistically well-grounded *extrapolation* from program behaviors observed during testing. For instance, a security researcher who has run the fuzzer for a week has currently *no* means (1) to estimate the total number of *feasible* program branches, given that only a fraction has been covered so far; (2) to estimate the additional time required to cover 10% more branches (or to estimate the coverage achieved in one more day, respectively); or (3) to assess the residual risk that a vulnerability exists when no vulnerability has been discovered. Failing to discover a vulnerability does not mean that none exists—even if the fuzzer was run for a week (or a year). Hence, testing provides *no formal correctness guarantees*.

In this article, I establish an unexpected connection with the otherwise unrelated scientific field of *ecology* and introduce a statistical framework that models Software Testing and Analysis as Discovery of Species (STADS). For instance, in order to study the species diversity of arthropods in a tropical rain forest, ecologists would first sample a large number of individuals from that forest, determine their species, and extrapolate from the properties observed in the sample to properties of the whole forest. The estimations (1) of the total number of species, (2) of the additional sampling effort required to discover 10% more species, or (3) of the probability to discover a new species are classical problems in ecology. The STADS framework draws from over three decades of research in ecological biostatistics to address the fundamental extrapolation challenge for automated test generation. Our preliminary empirical study demonstrates a good estimator performance even for a fuzzer with adaptive sampling bias—AFL, a state-of-the-art vulnerability detection tool. The STADS framework provides *statistical correctness guarantees* with quantifiable accuracy.

CCS Concepts: • **Security and privacy → Penetration testing**; • **Software and its engineering → Software testing and debugging**;

Additional Key Words and Phrases: Statistical guarantees, extrapolation, fuzzing, stopping rule, code coverage, species coverage, discovery probability, security, reliability, measure of confidence, measure of progress

ACM Reference format:
Marcel Böhme. 2018. STADS: Software Testing as Species Discovery. *ACM Trans. Softw. Eng. Methodol.* 27, 2, Article 7 (June 2018), 52 pages.
https://doi.org/10.1145/3210309

1 INTRODUCTION

The development of automated and practical approaches to vulnerability detection has never been more important. The recent worldwide WannaCry cyber-epidemic clearly demonstrates the

Dr. Böhme conducted this research at the National University of Singapore and has since moved to Monash University.
This research was partially supported by a grant from the National Research Foundation, Prime Minister's Office, Singapore, under its National Cybersecurity R&D Program (TSUNAMi project, No. NRF2014NCR-NCR001-21) and administered by the National Cybersecurity R&D Directorate.
Author's address: M. Böhme, Rm 131, 25 Exhibition Walk, Clayton VIC 3800, Australia; email: marcel.boehme@acm.org.

https://doi.org/10.1145/3210309

vulnerability of our well-connected software systems. WannaCry exploits a *software vulnerability* on Windows machines to gain root access on a huge number of computers all over the world. The ransomware uses the root access to encrypt all private data, which is released only if a ransom is paid. Hospitals had to shut down because life-saving medical devices were infected [127].

In 2017, a company's cost of cyber attacks worldwide was on average US $11.7 million, which is a 22.7% increase from the preceding year [98]. In February 2017, a bug was discovered in the HTML parser of Cloudflare, a company that offers performance and security services to about 6 million customer websites (including OKCupid and Uber). The bug leaked information, including private keys and passwords [125]. In July 2017, a hacker stole US $31 million from Ethereum, a blockchain-based platform, exploiting a vulnerability in the implementation of a protocol that was formally verified to be cryptographically sound [126]. To discover software vulnerabilities *at scale*, we need automated testing tools that can be used in practice, that work by the push of a button.[1]

Automated software testing (or fuzzing) has been an extremely successful automated vulnerability detection technique in practice. Our own fuzzers [7, 9, 10, 84] discovered 100+ bugs and more than 40 vulnerabilities in large security-critical software systems. Fuzzers, such as AFL [100], Libfuzzer [111], syzkaller [122], Peach [119], Monkey [115], and Sapienz [72], are now routinely used as automated testing and vulnerability detection techniques in large companies, such as Google [118], Microsoft [114], Mozilla [116], and Facebook [105]. The 2004 DARPA Grand Challenge inspired substantial research in self-driving cars, which are now a reality. The 2016 DARPA *Cyber* Grand Challenge [104], the world's first machine-only hacking tournament with $3.75 million in prize money, will arguably provide a similar push of research in advanced automated vulnerability detection. A *fuzzer* generates and executes program inputs, while a *dynamic analysis* (e.g., injected program assertions [91, 94]) identifies test executions that expose a vulnerability.

1.1 Extrapolation: A Fundamental Challenge of Automated Testing

A fundamental challenge of software testing is the statistically well-grounded *extrapolation* from program behaviors observed during testing. Harrold [58] established the "development of techniques and tools for use in *estimating, predicting, and performing testing*" as a key research pointer in her roadmap for testing. In an invited article on the future of software testing, Bertolino [5] corroborates that "we will need to make the process of testing more effective, *predictable* and effortless." In a recent IEEE Computer Society seminar, Whalen [99] argued that currently "there is *no sound basis to extrapolate* from tested to untested cases." Unlike automated verification, fuzzing does not allow one to make universal statements over program properties [38].[2]

No formal guarantees. If a verifier terminates without a counterexample, it formally guarantees the absence of vulnerabilities for *all* inputs. In contrast, a fuzzer perpetually generates random inputs and checks whether any of those exposes a vulnerability. Clearly, if the fuzzer generates a vulnerability-exposing input, a vulnerability exists. Yet, failing to expose a vulnerability does *not* mean that none exists. In fact, Hamlet and Taylor [56] argue that no matter how long the fuzzer is run (e.g., a year), if no vulnerability is discovered, we cannot report with *any* degree of confidence that none exists. So then, *what is the utility of a fuzzing campaign that exposes no vulnerabilities?*

[1] We concretely position this work within the software security domain and leverage the appropriate terminology. We take this position due to the practical impact and the recent, considerable traction of automated testing in the security domain. The security domain also provides a more compact terminology: "Fuzzing" instead of "automated software testing," "fuzzer" instead of "testing tool," "fuzzing campaign" instead of "execution of the testing tool," and so forth. Nevertheless, the central concepts that we present in this article apply to automated software testing and analysis in general.

[2] The emphasis (in *italics*) in all three quotes within this paragraph was added by the author.

No cost-effectiveness analysis. Suppose a security researcher has run the fuzzer for 1 week and exercised 60% of all program branches. Today, she has no means to estimate how much longer it would take to achieve, say, 70% coverage, or how much coverage would be achieved after, say, 1 more week. Perhaps the program is just very difficult to fuzz. However, there exists no formal measure of fuzzability either that would allow one to estimate the resources needed to achieve acceptable progress during a fuzzing campaign. In fact, our security researcher has no means to determine whether the fuzzer *can* even achieve 70% branch coverage at all. Some branches may just not be feasible. Perhaps 100% of feasible branches have already been covered. In that case, how should a security researcher judge the *campaign's progress toward completion*? In practice, exactly when to abort a campaign is mostly a judgment call that requires experience and guesswork. Bertolino [5] highlights the need for techniques to assess cost-effectiveness: "We would also need to be able to incorporate estimation functions of the cost/effectiveness ratio of available test techniques. The key question is: given a fixed testing budget, how should it be employed most effectively?"

No smart scheduling. The lack of oversight has consequences not only for individual security researchers but also for large multinational companies. For instance, Google Security has heavily invested in a large-scale fuzzing infrastructure called OSS-Fuzz, which is now generating some 10 *trillion* test inputs per day for more than 50 security-critical open-source software projects [118]. Each project is assigned roughly the same time budget. This is a waste of resources since fuzzing campaigns for certain programs stop making any progress after only a few hours, while campaigns for other programs continue to make progress for days on end. For now, there is no automated mechanism to measure how far a fuzzing campaign has progressed toward completion. Hence, no *smart* scheduling strategies for fuzzing campaigns have been developed yet.

Currently, a security researcher has no means to estimate the progress of the current fuzzing campaign toward completion or the confidence that the campaign inspires in the program's correctness. At any time in the campaign, the researcher has no means to gauge (let alone predict) the expected return on investment: how much more would he or she learn if he or she continued the campaign?

1.2 An Unexpected Connection with Ecology

In this article, I establish an unexpected connection with the scientific field of *ecology*, a branch of biology that deals with the relations of organisms to one another and to their physical surroundings. I argue that methodologies to estimate the number of species in an assemblage[3] provide an ideal statistical framework within which one can assess and extrapolate the progress of a fuzzing campaign toward completion and the confidence it inspires in the program's correctness. I conduct a preliminary empirical evaluation and outline future research directions to tailor and improve these methodologies for the requirements of automated software engineering and security.

Discovery in testing. My *key observation* is that automated software testing and analysis are about *discovery*. A fuzzer generates test inputs by sampling from the program's input space, and thus discovers properties about the program's behavior. Depending on the concrete objective, discovery means to find new bugs or vulnerabilities [8], to exercise interesting program paths [100], to cover new coverage goals, to kill stubborn mutants [66], to explore new program states [6, 87], to report unexpected information flows [73], or to explore new event sequences [115].

Discovery in ecology. Similarly, ecologists are concerned with the discovery of species in an assemblage. For instance, in order to study the biodiversity of arthropods in a tropical rain forest

[3] An *assemblage* is a group of individuals belonging to a number of different species that occur together in space and time. For example, all birds that live on an island today form an assemblage, all plants currently on Earth form an assemblage,and so forth.

Fig. 1. Species of arthropods (i.e., "bugs") discovered during ecological surveys in Singapore and Malaysia. The diversity and richness of arthropod species in tropical rain forests are notoriously difficult to assess due to the immense sampling effort that is required. According to a recent estimate [4], there are 6.1 million tropical arthropod species (high richness), most of which are rare (high diversity). *Photo Credit*: Marcel Böhme with the permission from Lee Kong Chian Natural History Museum, Singapore.

(Figure 1), ecologists would first sample a large number of individuals from that forest and determine their species. However, since sampling effort is necessarily limited, the sample is usually incomplete. The sample may contain several abundant species and miss many rare species. Biostatisticians spent the last three decades constructing a well-grounded statistical framework within which they can extrapolate, with quantifiable accuracy, from properties of the sample to properties of the complete assemblage (e.g., arthropod diversity in the tropical rain forest) [13, 21, 36].

STADS framework. My key observation allows us to model software testing and analysis as discovery of species (STADS). Consequently, STADS provides direct access to a rich statistical framework in ecology. Within the STADS framework, security researchers can leverage methodologies to accurately estimate the degree to which a software has been tested and to extrapolate, with quantifiable accuracy, from the behavior observed during testing to the complete program behavior. We show that an estimate of the probability to discover a new species provides a *statistical correctness guarantee*. Moreover, we present novel methodologies to assess *campaign completeness* (i.e., the progress of an ongoing campaign toward completion), *cost effectiveness* (e.g., the additional resources required to achieve an acceptable completeness), and *residual risk* that a vulnerability exists when none has been discovered.

Terminology. A *fuzzer* generates test inputs for a program. In STADS, a *test input* corresponds to an individual or sampling unit. A *dynamic analysis* identifies the *species* for an input. For instance, the AFL [100] instrumentation identifies the path exercised by an input; AddressSanitizer [91] identifies the memory error exposed by an input (if at all). A species is *rare* if only a few generated test inputs belong to that species, while a species is *abundant* if a large number of test inputs belong to that species. The *relative abundance* of a species describes the probability to generate a test input that belongs to that species. The program's *input space* represents the assemblage. The set of test inputs generated throughout a fuzzing campaign corresponds to the survey sample. We refer to Chao and Collwell (2017) [21], Chao and Lou [24], and Collwell et al. [34] for recent

reviews of the literature on the pertinent models and estimators spanning three decades of research in ecology.

Hypothesis. I hypothesize that within STADS, rare species that have been discovered explain the species within the fuzzer's search space that remain undiscovered. Intuitively, it is the "difficulty" to discover a rare species—measured by the total number of test inputs that needed to be generated before discovering the rare species—that provides insights on the discovery of undetected species that are evidently much rarer. A similar hypothesis is underpinning the nonparametric biostatistics in ecology [18]. In order to test this central hypothesis, we need to establish the accuracy of existing estimators and extrapolators from ecology within the STADS framework.

Species richness S. Estimating the total number of species S in the assemblage is a classical problem in ecology. If an ecologist samples n individuals and discovers $S(n)$ species, then $(S - S(n))$ species remain *undetected*. In order to quantify the species richness of the complete assemblage, nonparametric estimators $\hat{S}$ have been developed that become more accurate as sampling effort n increases [16, 17]. For instance, recently ecologists estimated the total number of species on Earth as 8.7 million [79], while only 14% have been discovered despite two centuries of taxonomic classification. In STADS, an estimate $\hat{S}$ of the asymptotic total number of species allows us to estimate the proportion $\hat{G} = S(n)/\hat{S}$ of all $\hat{S}$ species that have been discovered. For instance, we could estimate the *feasible* branch coverage, i.e., the proportion of actually feasible branches covered so far. The species coverage G can be used to assess *campaign completeness*, i.e., how much progress has been made toward completion. It could also be used to devise *smart scheduling strategies* for fuzzing campaigns that automatically abort a campaign that has reached a certain degree of completeness $\hat{G}$ and schedule the next one.

Discovery probability $U(n)$. In ecology, the discovery probability $U(n)$ measures the probability to discover a new species with the $n + 1$'th generated test input. The discovery probability can be estimated accurately and efficiently from the sample alone [49]. In the STADS framework, if the dynamic analysis is able to identify vulnerabilities, then the discovery probability U provides a *statistical guarantee* that no detectable vulnerability exists if none has been discovered. In other words, security researchers can use the STADS statistical framework for residual risk assessment. In ecology, the sample coverage $C = 1 - U$ quantifies the completeness of the sample, i.e., the proportion of individuals in the assemblage whose species is represented in the sample. Sample coverage is routinely used to choose the most accurate estimator for other quantities, such as species richness S [12], and to compare attributes of species across different assemblages [24].

Extrapolating species discovery $S(n + m^*)$ and $U(n + m^*)$. An extrapolation allows one to assess the tradeoff between investing more resources and gaining more insight. In ecology, there exist methodologies to quantify this return on investment. In STADS, a security researcher can use these methodologies to make an informed decision whether to continue or abort a fuzzing campaign. Suppose the client requires a statistical guarantee of $U(n + m^*) = 10^{-8}$ as an upper bound on the probability that the fuzzer finds a vulnerability in the program. The researcher can estimate the additional fuzzing effort m^* that is required to achieve *that* degree of confidence in the program's correctness. Suppose a fuzzer has achieved a statement coverage of $G(n) = 60\%$. Within STADS, the statistically well-grounded extrapolation allows one to estimate the coverage $G(n + m^*)$ that would be achieved if m^* more test inputs were generated.

1.3 Contributions

This article addresses the fundamental challenge of statistically well-grounded extrapolation both (1) *spatially* (i.e., from behaviors observed during fuzzing to *all* program behaviors) and (2) *temporally* (i.e., if the campaign was continued for some more time). We provide the first general

statistical model of STADS. For the first time, practitioners can use well-researched methodologies from ecology to make informed decisions about the fate of a fuzzing campaign and quantify what has been learned about the program. Within STADS, researchers can, for the first time, formally define novel metrics and identify or develop their estimators to investigate interesting properties of software, fuzzing campaign, and fuzzer.

- A fuzzer's effectiveness and efficiency may be measured and compared across other fuzzers. *Effectiveness* is determined by the number of species within the fuzzer's search space. *Efficiency* is determined by the number of species discovered per generated test input.
- A campaign's completeness, cost-effectiveness, and residual risk may be assessed as it is ongoing. *Campaign completeness* can be judged by the species coverage $G(n)$ or the sample coverage $C(n) = 1 - U(n)$. *Cost-effectiveness* can be assessed via extrapolation of the species discovered $S(n + m^*)$ or confidence achieved $U(n + m^*)$ if m^* additional test inputs were generated. The campaign's *residual risk* can be assessed via the discovery probability $U(n)$.
- The difficulty to fuzz a program (i.e., *software fuzzability*) can be estimated from the relative species abundance distribution. Intuitively, as the proportion of rare species increases, the difficulty to discover species increases as well.

The *primary contribution* of this article is the STADS model, which establishes the connection with ecology to provide access to a rich statistical framework that can address the challenges in automated software testing and analysis. However, due to space limitations, we can only present and investigate some pertinent aspects of the STADS framework. Specifically, this article makes the following secondary contributions.

- **Hypothesis.** I hypothesize that rare species that have been discovered explain the species within the fuzzer's search space that remain undiscovered. This hypothesis underpinning the STADS framework is *tested successfully* in our empirical study. Estimators and extrapolators that are based on rare species (i.e., singleton and doubleton species) demonstrate good performance for automated software testing and analysis. Within the STADS framework, we make *no assumptions* about the total number, relative abundance distribution, or location of species within the program's input space.
- **STADS models.** The *multinomial model*—where each input belongs to exactly one species—is integrated into the STADS framework and empirically evaluated. For instance, an input can execute only one path, exercise only one method call sequence, compute only one final output, and crash only at one program location. The *Bernoulli product model*—where each input belongs to one or more species—is integrated into the STADS framework. For instance, a single input can exercise multiple coverage goals (e.g., program statements, branches, or methods), kill multiple mutants, witness multiple information flows, violate multiple assertions, expose multiple bugs, and traverse multiple program states. For both models, we provide an *extensive survey* of ecological methodologies to estimate and extrapolate relevant quantities within the STADS framework and show how these methodologies can solve hard problems that have been long-standing in automated software engineering.
- **Evaluation.** In order to conduct an empirical evaluation of the multinomial model within the STADS framework, we fuzz six security-critical open-source programs for a cumulative 8.2 months using the popular, state-of-the-art fuzzer AFL [100]. The evaluation of two estimators ($\hat{G}(n)$ [16], $\hat{U}(n)$ [49]) and one extrapolator $\hat{S}(n + m^*)$ [92] demonstrates a reasonably low bias and high precision. We find that, despite the adaptive sampling bias of AFL, the methodologies are *statistically consistent*, meaning that bias decreases and

precision increases as more test inputs are generated. The estimate for one fuzzing campaign is fairly *representative* for other fuzzing campaigns of the same length.[4]

The STADS framework exhibits some peculiar features that make the direct application of existing ecologic methodologies more challenging: one has to deal with extremely *large populations* containing a *huge number of species* (e.g., millions of program branches), where *most species are rare*. Sampling strategies of feedback-directed fuzzers are (intentionally) subject to adaptive bias. For instance, in search-based software testing (SBST) [75, 76], the species discovered by future test inputs depend on the "fitness" of past test inputs. We point out many opportunities to identify, improve, tailor, and develop novel methodologies that address the peculiarities of the STADS model and sketch solutions to correct the adaptive bias of feedback-directed fuzzers.

1.4 Outline

The remainder of this article is structured as follows. Section 2 illustrates the main technical challenges and contributions using a practical motivating example. Section 3 introduces the STADS framework and multinomial model more formally and explains how the model relates to automated testing tools in practice. Sections 4 and 5 follow with a survey and discussion of estimation and extrapolation in the multinomial model of the STADS framework, respectively. In Section 6, we provide a preliminary empirical evaluation of the estimators and extrapolators within the multinomial model. In Section 7, we extend the STADS framework to account for inputs that can belong to multiple species by introducing the Bernoulli product model. In Section 8, we survey the relevant related literature. After an extended discussion of the peculiarities of the STADS framework and opportunities for future research in Section 9, we conclude in Section 10.

2 MOTIVATING EXAMPLE

We introduce the main ideas of our statistical framework of STADS using the following motivating example. We ran the fuzzer American Fuzzy Lop (AFL) for *1 week* on the program libjpeg-turbo compiled with AddressSanitizer (ASAN). *AFL* [100] is the state-of-the-art fuzzer for automated vulnerability detection. *Libjpeg-turbo* [112] is a popular, security-critical image parsing library that is used in many browser and server frameworks. *ASAN* [91] is a dynamic analyzer that identifies buffer overflows and other memory-related errors and vulnerabilities. We use that fuzzing campaign to illustrate the challenges and opportunities of automated testing and analysis in general.

Path discovery. While the true objective of AFL is to discover a maximal number of errors, it is an unlikely measure of progress; errors are (thankfully) rather sparse in the program's input space. Instead, the more immediate (and measurable) goal of AFL is to explore paths.[5] AFL's compiler-wrapper `afl-gcc` instruments the program such that each path yields a different *path-id*. ASAN instruments the program such that it *crashes* for inputs exposing a memory-related error. Hence, AFL's *concrete testing objective* is to discover a maximal number of paths and crashes.

Species discovery. In ecology, researchers sample individuals from an assemblage and identify their species to gain insights about the species richness and diversity of the assemblage. AFL's *fuzzer* `afl-fuzz` generates and executes test inputs for the instrumented program by applying random mutation operators at random points in a random seed file. In other words, AFL is a (biased) stochastic process that samples test inputs from the program's input space. Our *assemblage* is the

[4]More specifically, an estimate is fairly representative for other fuzzing campaigns where the same program is fuzzed for the same time using the same fuzzer and seed corpus (if any).

[5]To address path explosion, AFL clusters paths that exercise the same control-flow edges and do not yield substantially different hit counts for each edge [10]. Effectively, AFL reports the number of discovered path *clusters* rather than the number of discovered paths. For simplicity, we stick to the AFL terminology.

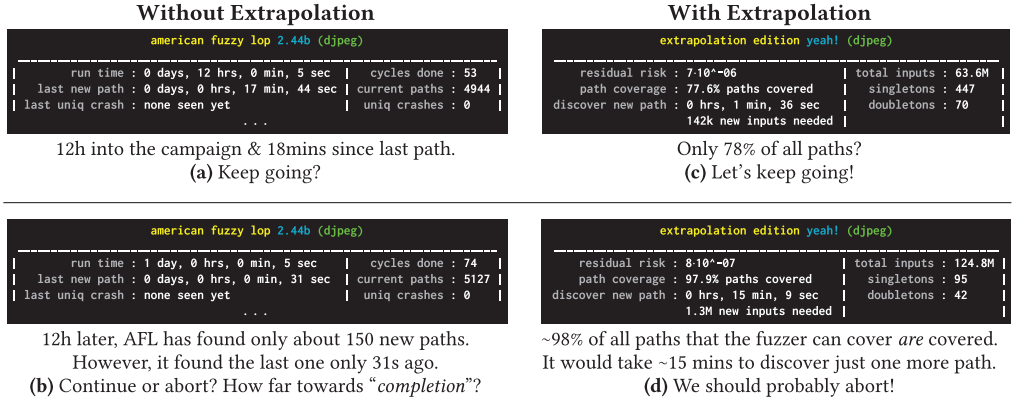


Fig. 2. The left-hand side ("without extrapolation") shows the first few lines of AFL's retro-style UI (AFL v2.44b). Specifically, it shows the pertinent information for the fuzzing campaign (a) at 12 hours and (b) at 24 hours. The right-hand side ("with extrapolation") shows our extension with estimates of the residual risk (i.e., the probability to discover a (crashing) path with the next input that is generated), the path coverage (i.e., the proportion of paths discovered), and the time or test inputs needed to discover the next path—for the fuzzing campaign (c) at 12 hours and (d) at 24 hours.

program's input space.[6] Our *individual* is a discrete input. Our *sample* is the set of all test inputs that have been generated throughout the current campaign. In this example, our *species* is the tuple (*path-id*, *crashing*) where *crashing* is true if the input crashes the program and false otherwise. ASAN and `afl-gcc` together form the *dynamic analysis* that identifies the species for a program input. The *general testing objective* is always to discover a maximal number of species.

Challenges. Figure 2(a) shows the progress for our fuzzing campaign after the passage of 12 hours—just like a security researcher might see it. In 12 hours, AFL has generated ~63 million (63M) test inputs and completed 53 cycles through the seed inputs. AFL has discovered about 5 thousand (5k) paths, and about 18 minutes (18 min) have passed since the discovery of the most recent path. Since the security researcher is given only the total number of paths, he or she cannot make an informed decision concerning the progress of the fuzzing campaign toward completion. About 18 minutes have passed since the last discovery of a new path. So, the researcher might reckon that the probability to discover a new path is very low. However, as we will see below, the time since the last discovery is rather *unreliable* and often changes several times per minute by up to *four orders of magnitude*. No crashes have been found. At 12 hours, the security researcher has no handle on the progress of the fuzzing campaign toward completion or on the correctness of the program.

Figure 2(b) shows the progress for our fuzzing campaign after 24 hours. The security researcher has learned that the number of discovered paths has not increased substantially in the last 12 hours. He or she may (or may not) decide to discontinue the fuzzing campaign based on this observation alone. However, the most recent path was found only a few seconds ago. So, he or she might be swayed to continue for at least a few more hours. Still, no crashes have been found. Even after 24 hours, the security researcher has no definite handle on making an informed decision about the completeness of the fuzzing campaign or how confident he or she can be in the correctness of the program.

[6]This is grossly simplified. Technically, our assemblage is the set of all program inputs that AFL is capable of generating using the available seed files and mutation operators. All statistical claims will hold only over AFL's search space.

2.1 Assessing Residual Risk Using the Discovery Probability

"Testing can be used to show the presence of bugs, but never to show their absence."
Edsger Dijkstra (1970) [38]

Finding no vulnerabilities in a (long-running) fuzzing campaign does not mean that none exists. A *residual risk assessment* would allow us to quantify the confidence the campaign inspires in the correctness of the program. In fact, our STADS framework provides *statistical guarantees* about the absence of vulnerabilities with quantifiable accuracy (e.g., 95% confidence intervals). In order to assess the residual risk, we suggest to estimate the probability U to discover a new species with the next generated test input. If the dynamic analysis, as in our motivating example, is able to identify vulnerabilities, then undiscovered vulnerabilities correspond to undiscovered species. Hence, the discovery probability U provides an upper bound on the probability to discover a new vulnerability with the next input that is generated. From this perspective, I argue that testing can be used to show that bugs are absent with a *certain likelihood* $(1 - U)$ that can be estimated efficiently and accurately *during* a fuzzing campaign, with a likelihood that increases over the course of a campaign.

In ecology, the *discovery probability* U gives the proportion of individuals in the assemblage whose species are *not* represented in the sample. In our motivating example, the discovery probability gives the proportion of all inputs in the input space that exercise yet undiscovered paths. We could say U represents how much of the program behavior remains untested. The *inverse of the discovery probability* $1/U$ provides the number of test inputs that we can expect to generate before discovering a new (path) species. The *sample coverage* $C = 1 - U$ is the complement of the discovery probability and effectively quantifies the degree of confidence that the fuzzing campaign inspires in the correctness of the program. In our example, at least $C\%$ of all inputs that AFL is capable of generating are expected to execute without crashes.

Out of the box, AFL already reports the time since the last discovery of a new species (Figure 2(a+b); `last new path`). This time to last discovery can be used as an estimate of the expected time to the next discovery. However, as we will see shortly, this estimate is very unreliable. Given the number m of test inputs that have been generated in the time since the last discovery, we can compute the *empirical discovery probability* as $\hat{U}_{\text{emp}} = 1/m$. However, the discovery probability thus estimated changes by orders of magnitude in a matter of seconds.

In Figure 3, we can see several estimators of the current discovery probability in an ongoing fuzzing campaign: **(a)** the empirical probability (i.e., $1 - 1/m$, where m is the number of test inputs needed to discover the most recent path), **(b)** the rolling median (i.e., the median empirical probability for the discovery of the $N = 11$ most recent paths), and **(c)** the Good-Turing estimator that is available in our STADS framework. Figure 3(a) shows the *empirical discovery probability* $\hat{U}_{\text{emp}}$ 1 day and 7 days into the fuzzing campaign, respectively. Unlike the sample coverage $C = 1 - U$, the discovery probability U can be represented on a log-scale. For instance, $t = 100$ *hours* into the fuzzing campaign, we find the empirical probability at about $2 \cdot 10^{-8}$. In other words, it took about $(2 \cdot 10^{-8})^{-1} = 50$ *million* test inputs to discover the next path. However, the empirical probability changes quite substantially in a matter of seconds. Particularly in the first 24 hours, the change can be over four orders of magnitude (Figure 3(a), top).

In signal processing, quick but large swings are often addressed with a moving average, the mean value of a set of N successive points. However, the moving average is susceptible to extreme events. Instead, the *moving median is more robust*, i.e., the median value of a set of N successive points. As we can see in Figure 3(b), the swings of the moving median $\hat{U}_{\text{mm}}$ are still quite substantial,

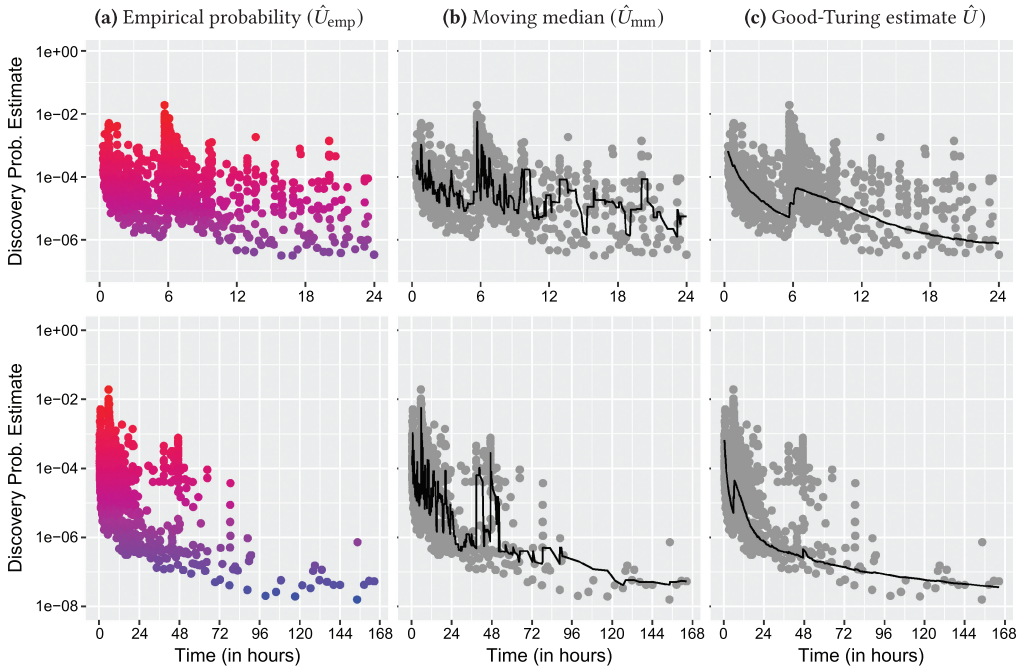


Fig. 3. Estimating the current discovery probability, i.e., the probability that a generated input discovers a previously undiscovered path over 24h (top) and 168h (bottom).

between one and three orders of magnitude. The moving median is right-aligned, meaning that the discovery probability estimate at time t is computed as the median of the N empirical values just preceding t. Hence, the moving median also generally *overestimates* the discovery probability. Increasing N to smooth the swings would only *increase the bias*. Moreover, $\hat{U}_{mm}$ is *not consistent*, meaning that $\hat{U}_{mm}$ is *not* guaranteed to approach the true discovery probability as sampling effort increases. So, the median (and mean) of the last N empirical probabilities $\hat{U}_{mm}$ is also an unreliable estimator of the current discovery probability (Figure 3(b); $N = 11$).

Main Hypothesis. *Almost all information about number and relative abundance of species that remain undiscovered is in the number and relative abundance of rare species that have been discovered.*

The main hypothesis of the STADS framework applied to our motivating example is that the number and "size" of paths that AFL has exercised only once or twice throughout the fuzzing campaign contains almost all information about the paths that are yet to be explored. Specifically, we denote as *singletons* those paths that are exercised by exactly one generated test input. Similarly, we denote as *doubletons* those paths that are exercised by exactly two generated test inputs. In Figure 2(d), we can see that one day into the fuzzing campaign after generating 125 million test inputs, there are still 95 singletons and 42 doubletons (~3% of discovered paths). This is one singleton for every 1.3 million generated test inputs. Clearly, it would require at least as many new test inputs to discover the next *undiscovered* path. In fact, this is the main insight of the Good-Turing estimator of the discovery probability. The *Good-Turing estimator* [49] is computed as the number of singletons divided by the number of samples (i.e., generated test inputs). The Good-Turing estimator is used across many disciplines of science, including rare event estimation [80], cryptanalysis [48], computational linguistics [43], and biology [24].

The *main hypothesis* of the STADS model *holds* for our motivating example. The proportion of generated inputs that exercise singleton paths accurately predicts the current discovery probability. In the bottom of Figure 3(c), we can see that the Good-Turing estimate $\hat{U}$ is not subject to huge swings like both empirical estimators. In fact, it was formally shown that (1) the estimator's accuracy strictly increases as the sample size (i.e., number of generated test inputs) increases [95], (2) its convergence to the true value is also reasonably fast [132], (3) its mean squared error is reasonably low [89], and (4) its performance is close to the best natural estimator for *any* distribution [81].

Figure 2 shows the discovery probability estimate $\hat{U}$, just like a security researcher might see it if he or she uses our AFL extension. $\hat{U}$ is shown under `residual risk` because the discovery probability provides an upper bound on the probability of discovering a vulnerability with the next input that is generated. Even if no crashing path has been detected in a very long running fuzzing campaign, there always exists a residual risk that an unexplored crashing path might be discovered in the future when more resources are being invested.

Twelve hours into the fuzzing campaign, the discovery probability is shown as $\hat{U} = 7 \cdot 10^{-6}$ (Figure 2(c)). The discovery probability is estimated as f_1/n, where f_1 is the number of *singletons* ($f_1 = 447$) and n is the number of *total inputs* ($n = 63.6 \cdot 10^6$). Depending on which residual risk is deemed acceptable, the security researcher can use the discovery probability to decide whether to continue or abort the fuzzing campaign. In fact, 12 hours later, 1 day into the fuzzing campaign, the discovery probability has decreased by one order of magnitude ($\hat{U} = 8 \cdot 10^{-7}$; Figure 2(d)).

We can use the discovery probability to compute other descriptive statistics, which the security researcher can use for his or her decision. For instance, the fuzzing effort has also increased by an order of magnitude: while it took only 1.5 minutes to discover a new path 12 hours into the fuzzing campaign, he or she can expect it takes 15 minutes to discover a new path 24 hours into the fuzzing campaign.

2.2 Assessing the Completeness of the Fuzzing Campaign

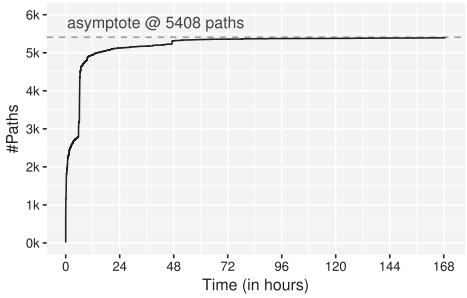
> "Currently, there is no sound basis to extrapolate from tested to untested cases."
> *Michael Whalen on the Future of V&V [99]*

AFL shows the number of paths that have been discovered in the current sampling campaign (Figure 2(a+b)). However, without an estimate of the number of paths that remain undiscovered, a security researcher cannot judge whether this is close or far from the discovery of *all* paths.

Within the STADS framework, we define *species coverage* G as the proportion of the asymptotic total number of species that have been discovered. In our motivating example, the *path coverage*—which is one kind of species coverage—gives the proportion of paths that have been discovered in the current fuzzing campaign. Hence, path coverage is a measure of the progress of the current fuzzing campaign toward completion. Unlike measures of code coverage, where the total number of elements is (assumed to be) known a priori, path coverage is more difficult to measure since the total number of paths is *unknown*. Currently, a security researcher has no means to compute the path coverage at any point in the fuzzing campaign.

Figure 4 shows the number of paths $S(n)$ that AFL discovered in libjpeg-turbo as the number of generated test inputs n increases.[7] We can see that the number of paths discovered approaches an asymptote, which we estimate to be at $\hat{S} = 5{,}408$ paths (using the *Chao1*-estimator of species richness [16]). The *asymptote* represents the total number of paths that the same fuzzer can discover for the same program given unlimited time. Essentially, we estimate the y-intercept $\hat{S}$ of the asymptote and yield $\hat{G}(n) = S(n)/\hat{S}$.

[7]For convenience, Figure 4(a) actually shows $S(n)$ over *time*.



Subject	: libjpeg-turbo [112]
Test driver	: djpeg
Fuzzer	: AFL [100]
Dynamic analysis	: ASAN [91]
Fuzzing time	: 168 hours
Generated inputs	: $973 \cdot 10^6$
Discovered paths	: 5392 (99.7%)
Est. total paths	: 5408
Unique crashes	: 0

Fig. 4. Example fuzzing campaign. (a) Number of paths discovered over time. (b) Descriptive statistics.

Figure 2 shows the path coverage estimate, just like a security researcher might see it if he or she uses our AFL extension. Twelve hours into the fuzzing campaign, AFL is estimated to have achieved 77.6% path coverage for the test driver of libjpeg-turbo (Figure 2(c)). This clearly indicates that the researcher should continue the fuzzing campaign in order to explore a greater percentage of paths. Twelve hours later, 1 day into the fuzzing campaign, the path coverage has increased to 97.7% (Figure 2(d)). At this point, he or she might decide to abort the fuzzing campaign if he or she feels that the time that AFL would require to explore the remaining paths is too high. In fact, 2 days later (i.e., 3 days into the fuzzing campaign), the path coverage has increased only to 99.1%, and 6 days later (i.e., 7 days into the fuzzing campaign) to 99.7%. Basically, spending six (6) *times* more hours fuzzing libjpeg-turbo only increased the path coverage by 2 percentage points. Clearly, the security researcher benefits tremendously from a measure such as path coverage when judging the progress of the fuzzing campaign toward completion.

We estimate the path coverage $\hat{G}$ after n test inputs were generated as follows:

$$\hat{G}(n) = S(n) \Bigg/ \left(S(n) + \frac{n-1}{n} \frac{f_1^2}{2f_2} \right),$$

where the denominator is the *Chao1* estimator [16] of species richness (i.e., of the total number of paths), $S(n)$ is the current number of paths discovered, f_1 is the number of singletons, $f_2 > 0$ is the number of doubletons, and n is the number of test inputs generated. Path coverage can be estimated *very efficiently and scalably* (i.e., independent of the size of the fuzzed program). In fact, AFL only needs to maintain the number of singletons f_1 and doubletons f_2 (Figure 2). Empirically, we find that the *accuracy* of the estimate increases as the number of generated inputs increases.

2.3 Extrapolating the Completeness of the Fuzzing Campaign

In automated software testing, we lack methodologies to predict how much more code coverage can be achieved if the fuzzer is run only for so much longer. In other words, we lack estimators of return on investment. For instance, Figure 5(a) shows the statement coverage that AFL has achieved 1 minute into the fuzzing campaign.[8] Even from the plot, the reader may find it difficult to estimate whether the coverage will remain at 60% or continue to increase to 70% within the next minute (i.e., at 2 minutes). In the following, we show how estimators from ecology can be used within our STADS framework to extrapolate statement coverage if more resources were invested.

So far, we have defined species based on the path that an input exercises, and whether it crashes or not. In the following, we allow an input to belong to multiple species where the set of species

[8]We measured statement coverage using `gcov` as a proportion of all *executable* statements.

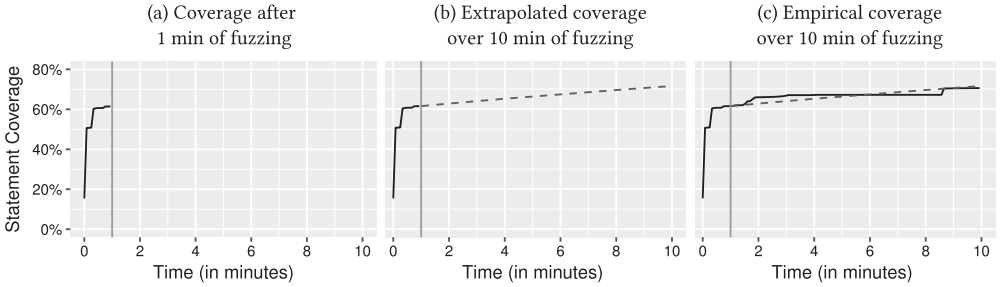


Fig. 5. Code coverage extrapolation: how much coverage is achieved if the fuzzer is run 2x, 5x, or 10x longer? We choose this particular interval because it does not seem quite obvious how the coverage would develop. Several hours into the fuzzing campaign, the general trend appears more predictable, which is why we can increase the extrapolation intervals from minutes to hours in Figure 1.

Table 1. Coverage Extrapolation from Time T to T'

			Statistical Extrapolation	
Current Time T	*Empirical Cov. @ T*	*Future Time T′*	*Extrapolated Cov. @ T′*	*Empirical Cov. @ T′*
1 min	61.5%	2 min	62.7%	65.9%
1 min	61.5%	5 min	66.3%	67.1%
1 min	61.5%	10 min	71.5%	70.5%
10 min	70.5%	15 min	71.7%	83.3%
10 min	70.5%	30 min	75.1%	83.7%
1 hour	87.2%	90 min	87.4%	87.5%
1 hour	87.2%	2 hours	87.6%	87.6%
10 hours	96.6%	15 hours	96.7%	96.7%
10 hours	96.6%	20 hours	96.9%	97.1%
1 day	97.1%	1.5 days	97.2%	97.2%
1 day	97.1%	2 days	97.8%	98.6%

Extrapolating the statement coverage of libjpeg-turbo at various points T into the fuzzing campaign for various times T' in the future. The *bias* of our estimate can be computed as the difference between the extrapolated with the empirical value of the statement coverage at time T'.

for an input t is given by the program statements that t covers. In our motivating example, the code coverage tool `gcov` [107] forms the dynamic analysis that identifies the statements covered by an input. Notice that statement coverage is just another kind of species coverage.

Figure 5(b) shows the extrapolation of the statement coverage within the first 10 minutes of the fuzzing campaign. The statement coverage is forecasted to increase by 9 percentage points if the security researcher invests nine times more minutes into the fuzzing campaign. Normally, Chao and Jost [24] would suggest to extrapolate only within twice the sample size (i.e., up to twice the length of the current fuzzing campaign). However, in this case the extrapolation is fairly accurate even within 10 times the sample size as we can see by overlaying the empirical values in Figure 5(c). Figure 1 shows more estimates of the statement coverage in the future. Despite the extrapolation

up to 24 hours into the future, the coverage estimate is within ±1 percentage points of the empirical value in eight out of 11 cases. Between 10 and 15 minutes, there is a sudden coverage increase by 12 percentage points that is explained by the adaptive sampling of AFL. The extrapolation was not able to forecast this sudden increase. Otherwise, our computed estimates are all fairly accurate.

Given that n test inputs have been generated and $S(n)$ of S statements have been covered in the fuzzing campaign, within the STADS framework we extrapolate the number of covered statements $S(n + m^*)$ when m^* more test inputs have been generated as follows [24]:

$$\hat{S}(n + m^*) = S(n) + \hat{Q}_0 \left[1 - \left(1 - \frac{Q_1}{n\hat{Q}_0 + Q_1} \right)^{m^*} \right],$$

where $\hat{Q}_0 = S - S(n)$ is the number of uncovered statements and Q_1 is the number of statements that are executed by exactly one generated test input. Since AFL only needs to maintain Q_1 (in addition to S, $S(n)$, and n), code coverage can be extrapolated *very efficiently and scalably* (i.e., independent of the size of the fuzzed program). The accuracy decreases as m^* increases. However, a 95% confidence interval that allows one to assess the decrease of accuracy is available via statistical bootstrapping [24].

Note that the STADS statistical framework also allows us to extrapolate other quantities such as discovery probability and other kinds of species coverage.

3 AUTOMATED SOFTWARE TESTING AND ANALYSIS AS DISCOVERY OF SPECIES

In the following, we present our statistical framework of automated STADS. Let $\mathcal{P}$ be the program that we wish to fuzz. We call as $\mathcal{P}$'s *input space* $\boldsymbol{\mathcal{D}}$ the set of all inputs that $\mathcal{P}$ can take. As inputs, we consider command line parameters, files, event sequences, messages, data streams, and databases, but also other objects that impact program behavior that are not normally considered inputs, such as environment variables, configuration, values returned from system calls, thread schedules, and so on. Let $\mathcal{F}$ be a stochastic process that samples inputs $t \in \boldsymbol{\mathcal{D}}$. We call $\mathcal{F}$ *fuzzer* and the sampling of inputs *test input generation.*

3.1 Search Space of the Fuzzer

Most fuzzers operate on a *restricted search space* $\boldsymbol{\mathcal{D}}' \subseteq \boldsymbol{\mathcal{D}}$ such that $\mathcal{F}$ is effectively unable to generate *all* inputs that $\mathcal{P}$ can take. For instance, a fuzzer might generate test inputs that are "valid" w.r.t. a prespecified input model [84], yet there might be programs that exhibit a vulnerability only for an invalid input. A fuzzer might generate inputs only up to a certain maximum size, yet there might be programs that exhibit a vulnerability only for substantially larger test inputs. Hence, the test inputs that are generated within a fuzzing campaign are necessarily random only within the capabilities of the fuzzer. For instance, the search space for CSmith [131] is a subset of all C programs (rather than a random sequence of UTF8-characters).

Hence, estimates that are derived from methodologies in the STADS framework hold only w.r.t. the fuzzer's search space and the tested program $\mathcal{P}$. The search space is specified either explicitly by the input model, grammar, or protocol that is used (and/or induced [61]) during fuzzing to reduce the fuzzer's search space [2, 46, 84], or implicitly by the fuzzer's inherent limitations to generate certain inputs. Most fuzzers do not also fuzz a program's environment (OS, architecture, current date, etc.), further restricting the behaviors that $\mathcal{P}$ can exhibit when fuzzed by $\mathcal{F}$.

In ecology, the sampling might also operate on a restricted search space [69]. For instance, certain areas may not be accessible. A net may not trap species smaller than its mesh. A light trap may not lure light-insensitive species.

3.2 Species Identification

Suppose the fuzzer's search space $\mathcal{D}'$ can be subdivided into S individual subdomains $\{\mathcal{D}_i\}_{i=1}^{S}$, called *species*. All inputs belong to at least one species and multiple inputs can belong to the same species. Specifically, all inputs belong to the same *species* that share the same discrete property of the program $\mathcal{P}$. For instance, we could consider each input that covers the same program statement to belong to the same species. Depending on the specific objective (e.g., "Cover all statements!"), we can choose a suitable species identification and devise a sampling strategy that can discover a maximal number of species. The *concrete fuzzing objective* is thus encoded by the way the species is identified for a certain input.

In our STADS framework, a *dynamic analysis* identifies the specific species to which a generated test input belongs. For C programs, the gcov coverage-tool [107] identifies the statements and branches an input covers; the AFL-instrumentation [100] identifies the specific path an input exercises; and AddressSanitizer [91] identifies the type of vulnerability an input exposes. Notice that some objectives require the dynamic analysis to identify a single species for each input (e.g., the path an input exercises), while others require it to identify multiple species for a single input (e.g., the statements an input covers). Notice also, we require *deterministic execution*: the same input, executed an arbitrary number of times, must always belong to the same species. However, conceptually we can integrate nondeterministic programs by considering all potential nondeterministic actions as part of the program's input space. For instance, a test input for a concurrent program also specifies a specific thread interleaving. A test input for an interactive program (e.g., an Android app) also specifies a state it must start from.

3.3 Fuzzing Campaigns

The fuzzer $\mathcal{F}$ generates n test inputs and is said to *discover* a species $\mathcal{D}_i$ when $\mathcal{D}_i$ is sampled for the first time. The *general fuzzing objective* is then to discover a maximal number of species. Let p_i be the probability with which $\mathcal{F}$ samples species $\mathcal{D}_i$ for $i : 1 \leq i \leq S$ at any point during the fuzzing campaign. Note that the STADS framework fully accounts for *arbitrary fuzzer heuristics*, including the sampling from the operational distribution [130], as long as the fuzzer does not change the sampling strategy adaptively throughout the fuzzing campaign. For instance, if a fuzzer generates more "typical" program inputs by sampling from the program's operational distribution—because the software engineer deems the detection of bugs that could also be found by a customer as more important—then all statistical claims derived from the STADS framework strictly hold w.r.t. that fuzzer within the stipulated confidence bounds. This fuzzer is simply more likely (greater p_i) to discover an "operational" bug $\mathcal{D}_i$ than a fuzzer without that heuristic, *for all fuzzing campaigns*.

Within our statistical framework, we must assume that the *relative species abundances* $\mathbf{p} = \{p_i\}_{i=1}^{S}$ does not substantially change during the fuzzing campaign. However, in practice, this assumption might not hold. We say that (feedback-directed) fuzzers where $\mathbf{p}$ changes during the fuzzing campaign have an *adaptive sampling bias*. For instance, a *coverage-directed fuzzer* retains generated test inputs that previously discovered a new species, and fuzzes those in addition to the initially provided seeds [100]. This allows one to sample a "neighboring" species $\mathcal{D}_i$ with a greater probability p_i (thus also increasing the efficiency of the fuzzer [10]). Yet, the rate at which new species are discovered is consistently decelerating throughout a fuzzing campaign. Hence, the rate at which the probabilities p_i change is also decelerating and the magnitude of the change decreases such that *the adaptive bias reduces* as more test inputs are generated. We investigate the adaptive bias empirically in Section 6 and provide an extended discussion in Section 9.5.

Within our STADS statistical framework, we do *not* assume that the fuzzer has any information about the species identification. Specifically, the "location" and relative abundance p_i of a specific

(undiscovered) species $\mathcal{D}_i$ as well as the total number of inputs N and the total number of species S are a priori *unknown*. Within a single *fuzzing campaign*, the fuzzer generates n test inputs and discovers $S(n)$ species. A species $\mathcal{D}_i$ is *discovered* when $\mathcal{F}$ generates the first test input t that belongs to $\mathcal{D}_i$, i.e., $t \in \mathcal{D}_i$. During a fuzzing campaign, the fuzzer generates X_i test inputs that belong to species $\mathcal{D}_i$, $\sum_{i=1}^{S} X_i = n$. Only species where $X_i > 0$ are marked as discovered.

There are two pertinent measures that *characterize a program*. The *species richness* S quantifies the total number of species, such as the number of statements, paths, vulnerabilities, information flows, and so forth in the program. In contrast, the *species evenness* J quantifies how "even" the relative abundances $\mathbf{p}$ are distributed. Formally, we compute *species evenness* J using Pielou's evenness index [85]:

$$J = \frac{H}{H_{\max}}, \qquad \text{where } H = -\sum_{i=1}^{S} p_i \ln p_i \quad \text{is Shannon's diversity index} \tag{1}$$

$$\text{and} \qquad H_{\max} = \ln S \qquad \text{is the max. possible value of } H. \tag{2}$$

Note that $0 \leq J \leq 1$. Shannon's diversity index is also known as Shannon entropy. Both quantities S and J can be illustrated by the following example. In Case #1, half of all inputs might exercise one path, while another half might exercise another. In Case #2, 90% of all inputs might exercise one path and only 10% another. Both cases have the same total number of paths ($S_1 = S_2 = 2$) but feature a very different evenness ($J_1 = 1$, $J_2 = 0.47$). The asymptotic total number of species S is important to determine how many more species we can expect to discover in a fuzzing campaign. The species evenness J is important to choose the right testing tool. If J is very low, symbolic execution-based fuzzers [15, 31] might be more appropriate than random fuzzers [100, 119].[9]

There are two pertinent measures that *characterize a fuzzing campaign*, species coverage and discovery probability. We define as *species coverage* G the proportion of species that have been discovered after generating n test inputs:

$$G(n) = \frac{S(n)}{S}. \tag{3}$$

Examples of species coverage are path coverage in our motivating example, where the total number of species S is *not* known and must be estimated, and code coverage, where S *is* indeed known. We define as *discovery probability* U the proportion of inputs that belong to species that remain undiscovered after generating n test inputs:

$$U(n) = 1 - \frac{\sum_{i=1}^{S} p_i I(X_i > 0)}{\sum_{i=1}^{S} p_i} = \frac{\sum_{i=1}^{S} p_i I(X_i = 0)}{\sum_{i=1}^{S} p_i}, \tag{4}$$

where $I(A)$ is the indicator function, i.e., $I(A) = 1$ if the event A occurs, and $I(A) = 0$ otherwise. We define as *sample coverage* C the complement of the discovery probability. In ecology, the sample coverage is the proportion of individuals in the assemblage whose species is represented in the sample. In automated software testing, it essentially quantifies the proportion of (tested *and* untested) program inputs that stress program behaviors that have already been tested before. In Section 4.1, we show that the estimate $\hat{U}$ of the discovery probability provides an upper bound on the probability to expose a vulnerability, whence the sample coverage estimate measures the confidence that a fuzzing campaign inspires in the correctness of the program. The sample coverage that is achieved in a fuzzing campaign depends on the species evenness J and the number of test

[9] For an extended discussion of blackbox versus whitebox testing efficiency, see Böhme and Paul [8].

inputs n that are generated. Intuitively, the lower the evenness, the more test inputs n a fuzzing campaign must generate to expect a reasonably high sample coverage C.

3.4 Main Hypothesis

I hypothesize that within the STADS statistical framework, the rare species that have been discovered throughout a fuzzing campaign explain the species within the fuzzer's search space that remain undiscovered. Intuitively, it is the *difficulty to discover a rare species*, measured by the total number of test inputs that needed to be generated before discovering the rare species, that provides insights on the difficulty of discovering yet undetected (but detectable) species.

Main Hypothesis. *Almost all information about number and relative abundance of undiscovered species within the fuzzer's search space is in the number and relative abundance of rare species that have already been discovered.*

A species $\mathcal{D}_i$ is considered *rare* if $1 \leq X_i \leq \kappa$, where κ is an arbitrary but very small constant. In fact, almost all estimators and extrapolators presented in this article are functions of the number of singleton and doubleton species (i.e., $\kappa = 2$).

The same hypothesis is underpinning the nonparametric biostatistics in ecology. Chao and Chui argue that "abundant species (which are certain to be detected in samples) contain almost no information about the undetected species richness, whereas rare species (which are likely to be either undetected or infrequently detected) contain almost all the information about the undetected species richness" [18]. Therefore, most nonparametric estimators and extrapolators are based on counts of rare species. In order to *test the main hypothesis*, we need to establish the accuracy of these estimators and extrapolators within the STADS framework.

This hypothesis is the reason for the great scalability of the STADS framework. For most estimates, the fuzzer needs to record only the number of rare species that have been discovered. Hence, the computation of the estimates scales easily to very large programs in our experiments.

3.5 The Multinomial Model: One Input, Single Species

Some concrete fuzzing objectives require one to identify a single species for each input. For instance, an input can execute only one path [47], exercise only one method call sequence, compute only one final output [87], or crash only at one program location; a single input either exposes a vulnerability or does not expose a vulnerability. In ecology, one individual can also belong only to a single species. A researcher samples individuals from the assemblage at various random locations and records for each detected species the number of occurrences. When individuals are sampled, ecologists call the collected data as *abundance data* and utilize the *multinomial model* [34, 63]. In the multinomial model, within STADS, a generated test input is considered as *individual*.[10]

Define the *abundance frequency count* f_k as the number of species that contain exactly k test inputs that were generated throughout the current fuzzing campaign, $0 \leq k \leq n$. More formally, $f_k = \sum_{i=1}^{S} I(X_i = k)$, where $I(A)$ is the indicator function, i.e., $I(A) = 1$ if event A occurs and $I(A) = 0$ otherwise. Hence, $n = \sum_{k=1}^{n} k f_k$ and $S(n) = \sum_{k=1}^{n} f_k$. The abundance frequency count f_0 represents the number of undiscovered species. We call f_1 the number of *singleton species* and f_2 the number of *doubleton species*. The input space contains S nonoverlapping subdomains, where the probability that the fuzzer generates a test input that belongs to species $\mathcal{D}_i$ is p_i for $i : 1 \leq i \leq S$.

[10] In Section 7, we extend the STADS framework to include the Bernoulli product model, where a generated test input is considered as a *sampling unit*. In the STADS framework, the Bernoulli model describes concrete fuzzing objectives that require one to identify one or more species for a single input.

Note that $\sum_{i=1}^{S} p_i = 1$. The multinomial probability distribution has the probability mass function

$$P(X_1 = x_1, \ldots, X_S = x_S) = \frac{n!}{x_1! \ldots x_S!} p_1^{x_1} p_2^{x_2} \ldots p_S^{x_S}. \quad (5)$$

From Equation (5), we can see that the number of generated test inputs X_i that belong to species $\mathcal{D}_i$ is a *sufficient statistic*, meaning that no other statistic that can be calculated from the same sample provides any additional information as to the value of the (estimated) parameter. This renders the abundance frequency counts f_k, which are defined from X_i, as suitable components for the estimators and extrapolators of fuzzing progress. As Colwell et al. [34] point out, the multinomial model assumes that the sampling procedure itself does not substantially alter the probabilities $(p_1, p_2, \ldots, p_S)$. The authors provide more details about the multinomial model and its utility in the ecologic context. The case where multiple species can be identified for a single input is explained by the Bernoulli product model [21] and discussed in Section 7.

4 ESTIMATING RESIDUAL RISK AND CAMPAIGN COMPLETENESS

Our model of STADS provides access to a rich statistical framework in ecology. This unexpected connection between two otherwise unrelated fields of research provisions software testing with methodologies to accurately estimate how much we have seen and to extrapolate from the seen to the unseen. In this section, we focus on the *estimation* of how much has been tested and how much more there is. We show that an estimate of the probability to discover a new species can provide a *statistical guarantee* that no (detectable) vulnerability exists that has not already been discovered. Moreover, we present novel methodologies to assess *campaign completeness* (i.e., the progress of an ongoing campaign toward completion).

4.1 Discovery Probability and Sample Completeness

In the STADS framework, the *discovery probability* $U(n)$ measures the current probability to discover a new species with the $n+1$'th generated test input, where n is the number of test inputs that have been generated throughout the current fuzzing campaign (i.e., $U(0) = 1$). If the dynamic analysis is able to identify vulnerabilities, then the discovery probability U provides a *statistical guarantee* that no detectable vulnerability exists if none has been discovered. In other words, security researchers can use the STADS statistical framework for residual risk assessment.

The concept of discovery probability might seem to require advance knowledge of the true relative species abundance $\{p_i\}_{i=1}^{S}$ during a fuzzing campaign. However, the discovery probability can be *very accurately and efficiently* estimated using only information contained in the single, uncompleted fuzzing campaign itself, as long as the number of generated test inputs is reasonably large [49, 89]. Hence, the concept of discovery probability finds application across many fields of science, such as rare event estimation [80], cryptanalysis [48], computational linguistics [43], biology [24], actuarial science, and so on.

In the STADS framework, the *sample coverage* $C(n) = 1 - U(n)$ measures the probability that the $n+1$'th generated test input belongs to an already discovered species. In other words, we know the species for $C\%$ of program inputs in the fuzzer's search space. Sample coverage also directly measures *sample completeness*, i.e., how complete the sample is w.r.t. the remaining undiscovered species in the assemblage. Hence, in ecology, sample coverage is routinely used to choose the most accurate estimator [12] and to compare attributes of species across assemblages [24]. In software testing, sample coverage can also be used to assess the progress of the current fuzzing campaign toward completion *without* the need to estimate $\hat{S}$ the total number of species. If the fuzzer has exposed no vulnerabilities, the sample coverage quantifies the *degree of confidence* that the fuzzing campaign inspires in the correctness of the program.

The *inverse of the discovery probability* $1/U(n)$ gives the number of test inputs that we can expect to generate before discovering a previously undiscovered species. Given the number of test inputs generated per unit time Δ, we can derive the expected time until discovery as $1/(\Delta \cdot U(n))$.

Estimation in STADS. In the multinomial model, the Good-Turing estimator estimates the probability to generate a test input that belongs to an undiscovered species. Thus, using the Good-Turing estimator [49], the estimate of the *discovery probability* $\hat{U}(n)$ is obtained as

$$\hat{U}(n) = \frac{f_1}{n}, \tag{6}$$

where f_1 is the number of singletons and n is the total number of generated test inputs. According to Good [48], the Good-Turing estimators were developed by Alan Turing during World War II while breaking Enigma codes. Good and Turing showed that their estimator can be accurately and efficiently computed only from the sample itself [49]. Moreover, the estimator is *strongly consistent*, meaning that its accuracy strictly increases as the sample size (i.e., number of generated test inputs) increases [95]. Zhang and Zhang [132] prove *asymptotic normality* of the Good-Turing estimator, meaning that the convergence to the true value is also reasonably fast. Robbins [89] showed that the *mean squared error* of the Good-Turing estimator is less than $1/n$, which indicates that it is quite accurate if n is large. In theory, it is assumed that the probability p_i to sample a species $\mathcal{D}_i$ follows a binomial distribution. However, in practice, the Good-Turing estimator seems to perform close to the *best natural estimator* for *any* distribution [81].

Statistical guarantee. We show that the *Good-Turing estimate* $\hat{U}(n)$ *of the discovery probability* provides an upper bound on the probability that an error in the fuzzer's search space remains undiscovered given that no error has been exposed after generating n test inputs.[11] Depending on the dynamic analysis, the fuzzer's search space is partitioned by the species identified for each input. Inputs that belong to the same species share the same input subdomain. Suppose the *progress-based dynamic analysis* partitions the input space according to the concrete fuzzing objective (e.g., based on the path or the statements that are exercised as in our motivating example). There are S subdomains $\mathcal{A} = \{\mathcal{D}_i\}_{i=1}^{S}$ in the fuzzer's search space. Further suppose that an *error-based dynamic analysis* partitions the same search space into T subdomains $\mathcal{B} = \{\mathcal{E}_j\}_{j=1}^{T}$. A partitioning is *error based* if all inputs that belong to the same species homogeneously either do or do not expose an error [8]. One could imagine error-based partitioning as black-and-white regions in the restricted input space of the program, where the black regions contain inputs that expose an error. In practice, a dynamic analysis, such as ASAN [91], would identify *some* test input executions that expose an error. Hence, the statistical guarantees hold *modulo* the dynamic analyzer's capability to identify an error-exposing input.[12] Let a *combined dynamic analysis* be derived by intersecting the progress- and error-based partitioning. The joint partitioning yields R species $\mathcal{AB} = \{\mathcal{D}_i \bigcap \mathcal{E}_j \mid \mathcal{D}_i \in \mathcal{A}, \mathcal{E}_j \in \mathcal{B}\}/\emptyset$, where $\cdot/\cdot$ is the difference operation to remove "empty" species and $R \leq S + T$. Notice that the number of singletons f_1 and doubletons f_2 for the progress-based analysis $\mathcal{A}$ are also the number of singletons and doubletons for the combined analysis $\mathcal{AB}$. Assuming that no error has been exposed throughout the fuzzing campaign, all error-*exposing* species in $\mathcal{AB}$ are clearly still among the undiscovered ones. Since the estimate U of the discovery probability denotes the proportion of inputs that belong to *undiscovered* species for $\mathcal{AB}$ (and $\mathcal{A}$), it provides an upper bound on the proportion of inputs exposing an error. A similar argument can be constructed trivially for the Bernoulli product model.

[11] A vulnerability is just a special case of an error.

[12] Similarly, in software verification, the formal guarantees are valid only *modulo* the provided specification. However, like a dynamic analysis in software testing, a specification may be *incomplete* (i.e., does not allow to detect *all* vulnerabilities; a.k.a. false negatives) or *incorrect* (i.e., reports vulnerabilities when there are none; a.k.a. false positives).

Quantifying accuracy. In the STADS framework, approximate estimators of the *variance* and the associated *confidence interval* can be derived with an asymptotic approach [26, 132]. We also note that one must account for the resulting *missing* probability mass when estimating the relative species abundance p_i for each discovered species $\mathcal{D}_i$, e.g., to estimate species evenness J [85]. In the multinomial model, the estimator $\hat{p}_i = X_i/n$ would evidently overestimate p_i. This can be remedied with an approach called *smoothing* [43, 49].

Scalability. In practice, the computation of the discovery probability estimate $\hat{U}(n)$ is efficient and easily scales with program size (i.e., with #species S). The fuzzer needs to store information only about doubleton and singleton species in addition to the number of generated test inputs and the number of discovered species. In the statistical programming language *R*, the `goodTuring`-function of the `edgeR`-package [90] implements Good-Turing estimation, while the `goodTuringProportions`-function implements the Good-Turing smoothing procedure. The `spadeR`- and `iNext`-packages [26, 62] for *R* compute the improved discovery probability estimator. The `iNext`-package also provides 95% confidence intervals.

4.2 Species Coverage

In ecology, species richness measures the number of species in the assemblage. In the STADS model, we define species coverage as the proportion of species in the assemblage that have been discovered throughout the fuzzing campaign. Hence, with an estimate of species richness $\hat{S}$, we can compute the current species coverage $\hat{G}(n) = S(n)/\hat{S}$ to assess the current progress of the fuzzing campaign toward completion. At the basis of most estimators is the observation that the species discovery curve *decelerates* over time as the number n of generated test input increases [8]. At the beginning of the fuzzing campaign, many species are discovered in a short time. Later, it takes more and more time to discover the next undiscovered species. For our motivating example, this deceleration can be observed in Figure 4. In fact, the discovery curve appears to approach an asymptote, which is estimated at 5,408 paths (using the *Chao1*-estimator [16]). The asymptotic total number of species is our estimation target. In the following, we review various estimators $\hat{S}$. We refer to Colwell et al. [34] for a more extensive review of available methodologies. An empirical and simulation-based comparison of several estimators was conducted by Hortal et al. [60].

During their investigations of the species discovery curve in what we now call the STADS model, Böhme and Paul [8] suggest fitting an exponential curve to extrapolate how many species we can expect to discover in a given time budget. *Curve fitting* would also allow us to determine the asymptotic total number of species [35, 86]. However, curve-fitting approaches are not based on any statistical sampling model, which prevents us from effectively evaluating the variance of the resulting asymptote. Moreover, different functional forms may manifest the same goodness of fit but yield vastly different estimates of the asymptote, which calls into question the statistical soundness of this approach.

Sampling-theory-based approaches build upon a statistical foundation and can be broadly distinguished into parametric and nonparametric frameworks [70]. In the parametric framework, it is assumed that the relative species abundances $\{p_i\}_{i=1}^{S}$ follow a statistical model with one or two parameters (e.g., Poisson process [42]). However, parametric models usually require extensive numerical procedures and work well only when the correct distribution is already known [32]. Yet, in software testing and analysis, just like in ecology, the distribution is often unknown. The most effective estimators of the total number of species are *sampling theory based and nonparametric* [19]. Here, we can distinguish jacknife, coverage-based, and Chao1/2-type estimators.

Jackknife estimators were developed to reduce the bias of a biased estimator and allow one to compute variance and confidence intervals for the estimate [14, 83, 97]. The current number of

discovered species $S(n)$ is obviously a negatively biased estimator of the total number of species S. In the multinomial model of the STADS framework, the first-order jackknife estimator $\hat{S}_{jk1}$ corrects this bias by assuming that the number of undiscovered species equals the number of singletons f_1:

$$\hat{S}_{jk1} = S(n) + \frac{n-1}{n} f_1 \tag{7}$$

$$\approx S(n) + f_1. \tag{8}$$

In the multinomial model of the STADS framework, the second-order jackknife estimator $\hat{S}_{jk2}$ for which the estimated number of unseen species is in terms of singletons and doubletons has the form

$$\hat{S}_{jk2} = S(n) + \frac{2n-3}{n} f_1 - \frac{(n-2)^2}{n(n-1)} f_2 \tag{9}$$

$$\approx S(n) + 2f_1 - f_2. \tag{10}$$

Burnham and Overton [14] provide higher orders of the jackknife estimators. All jacknife estimators can be expressed as linear combinations of frequencies and thus variances can be obtained.

Chao1-type estimators provide a lower bound for the total number of species rather than a point estimate [16]. When there are a large number of undiscovered species, it will be statistically impossible to obtain a good estimate of species richness. Hence, a good lower bound is often more practical than an imprecise point estimate. Chao [16] derived such a lower bound called *Chao1* for the multinomial model:

$$\hat{S}_{\text{Chao1}} = \begin{cases} S(n) + \frac{n-1}{n} \frac{f_1^2}{2f_2} & \text{if } f_2 > 0 \\ S(n) + \frac{n-1}{n} f_1(f_1-1)/2 & \text{if } f_2 = 0 \end{cases} \tag{11}$$

$$\approx \begin{cases} S(n) + f_1^2/(2f_2) & \text{if } f_2 > 0 \\ S(n) + f_1(f_1-1)/2 & \text{if } f_2 = 0, \end{cases} \tag{12}$$

where in the current fuzzing campaign n is the total number of test inputs generated, $S(n)$ is the total number of species discovered, and f_1 and f_2 are the abundance frequency counts for singleton and doubleton species, respectively.

Very recently, Chao et al. [20] showed that $\hat{S}_{\text{Chao1}}$ is an *unbiased point estimator* as long as very rare species (i.e., undetected and singleton species) have approximately equal relative abundance. If very rare species are unevenly distributed and the sample size is not sufficiently large, the available data do not contain sufficient information, and it is only reasonable to provide a good lower bound estimate of species richness S.

An *improved lower bound* can be obtained from tripleton and quadrupleton species, respectively. Chui et al. [32] derived the improved lower bound called *iChao1* for the multinomial model:

$$\hat{S}_{\text{iChao1}} = \hat{S}_{\text{Chao1}} + \frac{n-3}{n} \frac{f_3}{4f_4} \times \max\left(f_1 - \frac{n-3}{n-1} \frac{f_2 f_3}{2f_4}, 0\right) \tag{13}$$

$$\approx \hat{S}_{\text{Chao1}} + \frac{f_3}{4f_4} \times \max\left(f_1 - \frac{f_2 f_3}{2f_4}, 0\right), \tag{14}$$

where f_3 and f_4 are the frequency counts for tripleton and quadrupleton species, respectively.

Coverage-based estimators utilize sample coverage, the proportion of inputs belonging to discovered species, to estimate the total number of species [25, 32]. As we have seen earlier, sample coverage, as the complement of the discovery probability, can be very accurately and efficiently estimated from the frequency counts alone, as long as the number of test inputs generated in the

fuzzing campaign is reasonably large [49]. As Chao and Chiu [19] point out, coverage-based estimators might be appropriate when there are many rare species, i.e., where $0 \ll |\{p_i \mid p_i \ll \frac{1}{S}, 1 \leq i \leq S\}| \lesssim S$ and $|\cdot|$ gives the cardinality of the set. However, for lack of space, we are adjourning to future work the discussion and evaluation of the ACE and ACE-1 estimators [25, 28].

Species coverage. In the STADS framework, we compute the estimate $\hat{G}$ of the species coverage that has been achieved in the campaign by dividing the number of currently discovered species $S(n)$ by the estimated total number of species $\hat{S}$. If S is known, then $\hat{S} = S$. For instance, in our motivating example, path coverage is computed w.r.t. an estimated total number of species, while statement coverage is computed w.r.t. the known total number of statements. Both statement and path coverage are examples of species coverage, only that the same inputs are assigned to a different kind of species.

Quantifying accuracy. The *variance* and 95% *confidence intervals* for the estimators in the STADS framework can be derived by the standard statistical approximation method [17] or using bootstrapping [21]. Hortal et al. [60] find that estimator accuracy strongly depends on the species evenness J and on the completeness $C = 1 - U$ of the sample. Effectively, the accuracy of the estimate improves as the discovery probability U decreases or species evenness J increases.

Scalability. In practice, the computation of all sampling-theoretic, nonparametric estimators of species coverage is efficient and easily scales with program size (i.e., with #species S). In most cases, the fuzzer needs to store information only about doubleton and singleton species in addition to the number of generated test inputs and discovered species. In the statistical programming language *R*, the `ChaoSpecies`-function of the `SpadeR`-package [26] implements several estimators of the total number of species. The `ChaoSpecies`-function also reports 95% confidence intervals.

5 EXTRAPOLATION OF SPECIES DISCOVERY

An extrapolation allows one to assess the tradeoff between investing more time and gaining more insight. We discuss novel methodologies from ecology to quantify this return on investment. Specifically, using extrapolation in the model of STADS, the security researcher can answer the following questions:

(1) Given that in the current fuzzing campaign n test inputs have been generated and the researcher has time to generate only m^* more test inputs, how much species coverage $\hat{G}(n + m^*)$ and residual risk $\hat{U}(n + m^*)$ can he or she expect to achieve?
(2) Given that in the current fuzzing campaign n test inputs have been generated and the security researcher would like to achieve a specific species coverage G^*, how many more test inputs m_{G^*} can he or she expect to generate before achieving G^* (i.e., m_{G^*} s.t. $\hat{G}(n + m_{G^*}) = G^*$)?

Using these extrapolators, a security researcher can make an informed decision whether to continue or abort a fuzzing campaign. Suppose the client requires a statistical guarantee (i.e., discovery probability) of 10^{-8} as the upper bound of the probability that the fuzzer finds a vulnerability in the program. The researcher can estimate the effort that is required to achieve that degree of confidence in the correctness of the program.

5.1 Estimating Progress Toward Completion within a Given Time Budget

In our STADS statistical framework, there are several estimators of the expected number $\hat{S}(n + m^*)$ of discovered species if the reference sample of size n was augmented by $m^* > 0$ more individuals (i.e., if m^* *more* test inputs were generated) [37, 50, 92, 93]. Chao and Jost [24] provide an overview. In the multinomial model, Shen et al. [92] proposed the following sampling-theoretic extrapolator

based on the asymptotic total number of species:

$$\hat{S}(n+m^*) = S(n) + \hat{f}_0 \left[1 - \left(1 - \frac{f_1}{n\hat{f}_0 + f_1} \right)^{m^*} \right], \tag{15}$$

where for the current fuzzing campaign, n is the number of generated test inputs, $S(n)$ is the number of discovered species, $\hat{f}_0 = \hat{S} - S(n)$ is the expected number of undiscovered species, and f_1 is the number of singletons.

The rule of thumb is to keep the extrapolation within twice the sample size (i.e., $m^* \leq n$) [34]. However, recently Orlitzki et al. (2016) [82] introduced the *provable* extrapolation of the discovered species for m^* all the way up to $n \cdot [\log(n) - 1]$ additional test inputs. This shows that the number of discovered species can be estimated for a population $\log(n)$ times larger than that observed. The authors go on to show that this is also the largest possible estimation range and that the estimators' mean-square error is optimal up to constants for any m^*.

In the multinomial model of the STADS framework, we can derive the expected discovery probability $\hat{U}(n+m^*)$ if m^* more test inputs were generated by recognizing that the discovery probability is only the difference in the number of discovered species between this and the next generated test input $U(n) = S(n+1) - S(n)$. Hence,

$$\hat{U}(n+m^*) = \hat{S}(n+m^*+1) - \hat{S}(n+m^*) \tag{16}$$

$$= \frac{f_1}{n} \left(\frac{n\hat{f}_0}{n\hat{f}_0 + f_1} \right)^{m^*+1}, \tag{17}$$

where $\hat{f}_0 = \hat{S} - S(n)$ is the expected number of undiscovered species, f_1 is the number of singleton species, and f_2 is the number of doubleton species.

Quantifying accuracy. In the STADS model, confidence intervals for the estimators $\hat{S}(n+m^*)$ and $\hat{U}(n+m^*)$ can be derived using the bootstrap method [24, 34]. In ecology, the rule of thumb is to keep the extrapolation within twice the sample size (i.e., $m^* \leq n$) [34]. The reason can be illustrated with the following example. Intuitively, the accuracy of $\hat{S}(n+10)$ is better when $n = 1{,}000$ than it is when $n = 1$. First, the extrapolator performs better at $n = 1{,}000$ because more information is available. Second, the margin of error is also reduced because the species discovery curve decelerates substantially (cf. Figure 4(a)).

Scalability. In practice, the extrapolation of species coverage $\hat{G} = \hat{S}(n+m^*)/\hat{S}$ and of the discovery probability $\hat{U}(n+m^*)$ is efficient and easily scales with program size (i.e., with #species S). The fuzzer needs to store information only about doubleton and singleton species in addition to the number of generated test inputs n and the number of discovered species $S(n)$. In the statistical programming language *R*, the `iNext`-package [62, 109] computes the extrapolation and also provides 95% confidence intervals.

5.2 Estimating Number of Inputs Needed to Discover a Given Proportion of Species

Chao et al. [22] developed a nonparametric method for estimating the number of further test inputs that would need to be generated in order to achieve an arbitrary species coverage G^*. Formally, to reach a fraction G^* of estimated total number of species $\hat{S}$ where $\hat{G}(n) < G^* < 1$, in the multinomial model of the STADS framework the required number m_{G^*} of further test inputs is estimated as

$$m_{G^*} \approx \frac{nf_1}{2f_2} \log \left[\frac{\hat{f}_0}{(1-G^*)\hat{S}} \right], \tag{18}$$

where in the current fuzzing campaign n is the number of generated test inputs, $S(n)$ is the number of discovered species, $\hat{f}_0 = \hat{S} - S(n)$ is an estimate of the number of undiscovered species, and f_1 and f_2 are the abundance frequency counts for singleton and doubleton species, respectively.

Accuracy. In the STADS model, confidence intervals for the estimators can be derived using the bootstrap method [22]. Given the estimate $\hat{G}(n)$ of current species coverage, we suggest that $\hat{G}(n) \leq G^* \leq 0.5 + \frac{\hat{G}(n)}{2}$ to keep the accuracy within a reasonable range. This suggestion is a variant of the rule of thumb stated in Section 5.1 that $m_{G^*} \leq n$ [34].

Scalability. In practice, computing m_{G^*} is efficient and easily scales with program size (i.e., with #species S). The fuzzer needs to store information only about doubletons and singletons in addition to the number of generated test inputs and discovered species. The logarithm of a 32-bit floating-point number in Equation (18) can be computed efficiently with a typecast, a bit shift, and a subtraction operation [124]. All other basic mathematical operations require one CPU step each. In the statistical programming language *R*, the number of test inputs required to discover a certain proportion of all species can be estimated with the `num.samples.required`-function in the `sprex`-package [121].

6 EMPIRICAL EVALUATION

6.1 Research Objectives

The main objectives of this preliminary empirical evaluation are as follows:

(1) To *test my main hypothesis* that within the model of automated STADS, the rare species that have been discovered throughout a fuzzing campaign explain the species within the fuzzer's search space that remain undiscovered.
(2) To *evaluate ecologic estimators and predictors* for the multinomial model in STADS. Specifically, we evaluate the *Chao1* estimator $\hat{S}$ of species richness [16]; the predictor $\hat{S}(n + m^*)$ by Shen, Chao, and Feng [92] of the number of species that would be discovered if m^* more test inputs were generated; and the Good-Turing estimator $\hat{U}(n)$ [49] of the discovery probability.
(3) To *investigate the impact of the adaptive sampling bias* of a feedback-directed fuzzer. An underlying assumption of most methodologies in the STADS framework is that the probability p_i to generate a test input that belongs to species $\mathcal{D}_i$ does *not* change substantially during the fuzzing campaign. However, it does for feedback-directed fuzzers, such as AFL.

We use path coverage as one kind of species coverage (1) because path coverage is the main measure of progress for our extension of AFL [100] (see Section 2), the fuzzer used for our experiments, and (2) because path coverage satisfies the conditions of the multinomial model (one species per input). We employ the *Chao1*-estimator [16] to estimate the asymptotic total number of paths $\hat{S}$ and the *Good-Turing*-estimator [49] to estimate the discovery probability. We estimate path coverage as $\hat{G}(n) = S(n)/\hat{S}$. To extrapolate the number of paths discovered in the subsequent fixed-time interval, we compute the average number of tests generated per unit time and leverage the sampling-theoretic estimator $\hat{S}(n + m^*)$ proposed by Shen et al. [92]. For our evaluation, we use established measures of estimator accuracy. The *bias* of an estimator measures the mean difference of the estimate to the true value of the estimation target, while the *precision* measures the variance of the estimates. Specifically, we ask the following research questions:

RQ.1 Can path coverage $\hat{G}(n)$ be used to effectively estimate the progress of a fuzzing campaign toward completion? Do different programs achieve the same path coverage, say, 6 or 48 hours into the fuzzing campaign?

Table 2. Subjects: Four Security-Critical Open-Source C Projects of Different Program Sizes

Program	Size	Test Driver	Description
json [110]	44 kLOC	`parse_msgpack`	JSON parser
libjpeg-turbo [112]	91 kLOC	`libjpeg_turbo_fuzzer`	JPEG image library
openssl [117]	472 kLOC	`server`	cryptography and SSL/TLS library
libxml2 [113]	500 kLOC	`xmlint -d`	XML parser
ffmpeg [106]	1071 kLOC	`AV_CODEC_ID_MPEG4_fuzzer`	audio and video streaming library
wireshark [123]	3522 kLOC	`fuzzshark_media_type-json`	network protocol analyzer

RQ.2 Can discovery probability $\hat{U}(n)$ be used to effectively estimate the residual risk of leaving detectable vulnerabilities undetected? Is the estimate representative for different fuzzing campaigns of similar length?

RQ.3 How *biased* is the *Chao1*-estimator $\hat{S}$ of the total number of paths? Is $\hat{S}$ systematically positively or negatively biased? What is the bias's magnitude and how can it be corrected?

RQ.4 How *precise* is the *Chao1*-estimator $\hat{S}$ of the total number of paths? How can the precision of $\hat{S}$ be increased?

RQ.5 How *biased* is the extrapolation of the number of discovered paths $\hat{S}(n + m^*)$ if m^* more inputs were generated, where m^* is the number of test inputs that we can expect to generate in 30 minutes, 1 hour, 2 hours, or 4 hours? Is $\hat{S}(n + m^*)$ systematically positively or negatively biased? What is the magnitude of the bias and how can it be corrected? Does the rule of thumb [34] to keep the extrapolation within twice the sampling effort apply to automated software testing and analysis?

RQ.6 How *precise* is the extrapolation of the number of discovered paths $\hat{S}(n + m^*)$ if m^* more inputs were generated, where m^* is the number of test inputs that we can expect to generate in 30 minutes, 1 hour, 2 hours, or 4 hours? How can the precision of $\hat{S}(n + m^*)$ be increased?

We present a *summary* of our results w.r.t. our main objectives in Section 6.5.

6.2 Setup and Infrastructure

Implementation. We implemented the pertinent estimators and extrapolators into American Fuzzy Lop (AFL) [100]; we call our tool Pythia. Pythia uses lightweight instrumentation to determine, with negligible performance overhead, a unique identifier for the path that is exercised by an input. New inputs are generated by mutating a seed input using bit flips, boundary values, and block deletion and insertion strategies. If the new input exercises a new branch or exercises a previously exercised branch exponentially more (or less often), it is added to the fuzzer's queue. Pythia stores for each seed in the queue the path id and the number of generated test inputs that yield the same path id. About every 5 seconds, Pythia writes to a file the pertinent fuzzer data, including the current unix time, the number of generated test inputs n, the number of discovered paths $S(n)$, and the number of singletons f_1 and doubletons f_2 (i.e., #paths exercised once or twice).

Subjects. We chose six subject programs from Google's OSSFuzz fuzzing infrastructure [118]. The infrastructure fully automates the fuzzing of the 50+ integrated open-source C projects. OSS-Fuzz automatically downloads the most recent version of the subject, builds the subject, compiles the test drivers, and provides the initial seed corpus. We integrated Pythia as fuzzer into the fuzzing infrastructure. The list of subject programs used for our experiments is shown in Table 2. We chose these subjects because they are all security critical, well fuzzed, and of different sizes.

Setup. For each subject we ran 10 fuzzing campaigns for 100 hours. The 10-fold repetition of the fuzzing campaign allows us to discuss bias and precision of the estimators. The fuzzer was started with the same set of seeds and targeted the same test driver (col. 3 in Table 2). In total, we spent a cumulative 6,000 hours, ≈8.2 months, fuzzing these six subjects.

Estimator performance. Bias and precision are standard performance measures for estimators and extrapolators in biostatistics and ecology [12, 96]. *Bias* quantifies the difference between the estimate and the true value of the estimation target. A systematically positively or negatively biased estimator consistently over- or underestimates, respectively, the true value of the estimation target. *Precision* quantifies the statistical variance of the estimator (i.e., how close repeated estimates of the same quantity are to each other). An estimator with a low variance has a high precision, and vice versa. Unlike bias, the magnitude of precision is only dependent on the estimated values and is hence completely independent of the true value. Bias and precision are *scaled* w.r.t. the true value of the estimation target. For instance, an estimator with a bias of 0 provides exactly the true value of the estimation target, an estimator with a bias of -1 provides 0 as estimate, and an estimator with a bias of 1 provides exactly twice the true value.

6.3 Estimator Evaluation

We compute the *mean bias* of the estimator $\hat{S}$ of the total number of paths S as the average bias over $N = 10$ runs and the *imprecision* of $\hat{S}$ as the standard deviation of the bias [12]:

$$mean\ bias = \sum_{i=1}^{N} \frac{\hat{S}_i - S_i}{NS} \tag{19}$$

$$imprecision = \sqrt{\frac{\sum_{i=1}^{N} \left(\frac{\hat{S}_i - S_i}{S_i} - \frac{\left[\sum_{j=1}^{N} \hat{S}_j - S_j\right]}{NS} \right)^2}{N-1}}, \tag{20}$$

where S_i is the estimated total number of paths for the ith run *at 100 hours of fuzzing*. Even at 100 hours, the empirical value may still substantially underestimate the true species richness. A low imprecision means that all estimates are similarly (un)biased.

RQ.1 Completeness $\hat{G}(n)$. Path coverage provides a useful indicator of the progress of a fuzzing campaign toward completion. Some programs require more time than others to achieve the same path coverage. Table 3 shows the estimated path coverage $\hat{G}(n)$ at 6, 48, and 100 hours into the fuzzing campaign as an average over 10 runs. Six hours into the fuzzing campaign, we see $\hat{G} = 99\%$ for json, meaning Pythia has discovered almost all paths that it could potentially explore. In fact, after spending seven times more time (i.e., after 48 hours), Pythia has discovered only 45 more paths. For all practical purposes the average fuzzing campaign for both json and wireshark might be considered completed shortly after the 6-hour mark.

Clearly, openssl, libxml2, and ffmpeg do not appear to be completed at the 6-hour mark with a path coverage well below 90%. In fact, more than 300, 1,600, and 100 paths are still being discovered, respectively, until the 48-hour mark. At the 48-hour mark, the average fuzzing campaign for ffmpeg might be considered completed. In fact, only 12 more paths are found until the 100-hour mark. However, the average fuzzing campaign for libxml2, and openssl remains incomplete even at the 48-hour mark. Indeed, about 1k, and 100 more paths are being discovered, respectively, until the 100-hour mark. For libxml2, about 3,200 new paths are found until the *800*-hour mark, on average (33 days; 10,846 avg. #paths). We explain the decrease in coverage from the 48- to the 100-hour mark for libxml2 and openssl with the lack of a discernible horizontal asymptote (second row in

Table 3. Average Number of Discovered Paths S_{obs} and Estimated Path Coverage $\hat{G}$ Over 10 Runs at 6, 48, and 100 Hours into the Fuzzing Campaign, Respectively

Subject	S_{obs} ($\hat{G}$) @ 6hrs	S_{obs} ($\hat{G}$) @ 48hrs	S_{obs} ($\hat{G}$) @ 100hrs
json	2,612 (98.7%)	2,657 (99.9%)	2,665 (99.9%)
libjpeg-turbo	2,224 (95.2%)	2,547 (99.4%)	2,623 (99.6%)
openssl	1,041 (86.3%)	1,356 (93.3%)	1,444 (88.6%)
libxml2	5,071 (57.3%)	6,672 (67.3%)	7,656 (66.0%)
ffmpeg	2,420 (71.8%)	2,554 (99.3%)	2,568 (99.6%)
wireshark	427 (98.0%)	454 (99.1%)	456 (99.3%)

< 95%	The campaign is considered incomplete.
< 98%	Decide based on other factors.
≥ 98%	The campaign is considered nearly complete.

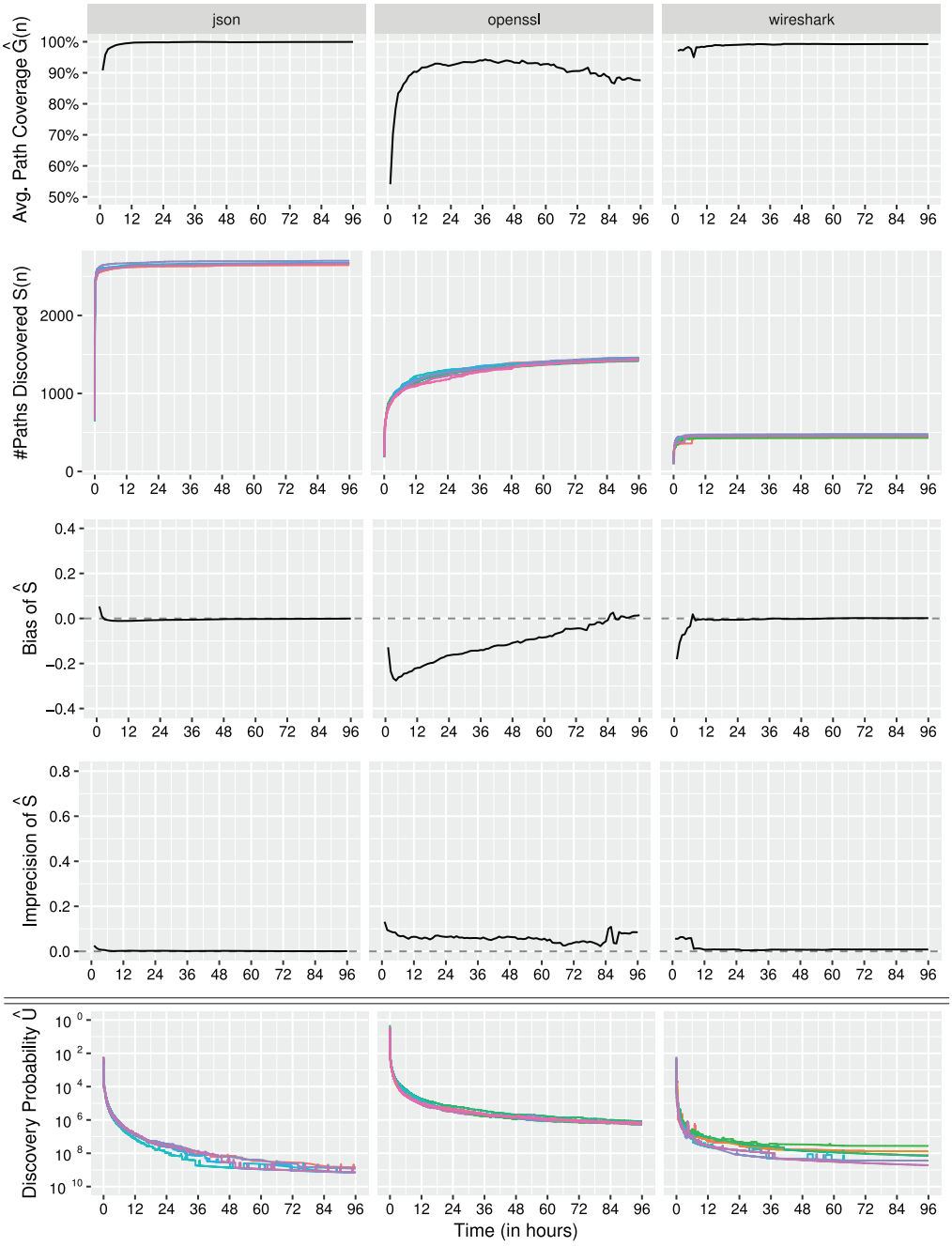
Tables 4 and 5; RQ.3). We also note that AFL's existing stopping rule[13] would have (incorrectly) aborted openssl already at the 6-hour mark, while ffmpeg would (incorrectly) continue even after the 48-hour mark.

RQ.2 Residual Risk $\hat{U}(n)$. The current discovery probability provides a useful indicator of the current residual risk that a discoverable vulnerability exists but remains undiscovered in the ongoing fuzzing campaign. The discovery probability measured for one fuzzing campaign is fairly representative for other fuzzing campaigns of similar length, particularly later in the fuzzing campaign. The bottom row in Tables 4 and 5 shows the discovery probability estimate $\hat{U}(n)$ over time. Notice the log-scale on the y-axis. The first observation that we can make is that fuzzing campaigns of the same length yield different degrees of residual risk for different subjects. For instance, at the 96-hour mark, the discovery probability estimate across subjects ranges over four orders of magnitude (e.g., libxml2 vs. json). The second observation that we can make is that for the same subject, the variance of the estimate across fuzzing campaigns is rather small, indicating a certain degree of representativeness. The third observation is a general deceleration where each discovery probability seems to almost approach a horizontal asymptote: as the campaign continues, it takes more and more test inputs to achieve the same decrease in discovery probability (and hence the same decrease of residual risk). Our fourth observation is that fuzzing campaigns with a relatively high discovery probability (libxml2, openssl) also achieve a relatively low path coverage estimate (first row) with a relatively high estimator bias (third row). This provides opportunities to devise suitable adaptive bias correction strategies based on the discovery probability.

RQ.3 Bias of $\hat{S}$. For four of six subjects, the bias reduces to within ±10% of the true species richness S within the first 12 hours. For the other two (libxml2 and openssl), it takes 48 hours. In some cases the mean bias is systematic and negative; in other cases it is systematic and positive. The magnitude of the positive bias can be substantial in the first few hours (>5.0 for ffmpeg). Notice that a substantial overestimation of species richness S results in a (conservative) underestimation of the path coverage $G(n)$. However, in all cases, the mean bias tends to 0 as the number of generated test inputs increases over time. There are several sources of bias in estimating S.

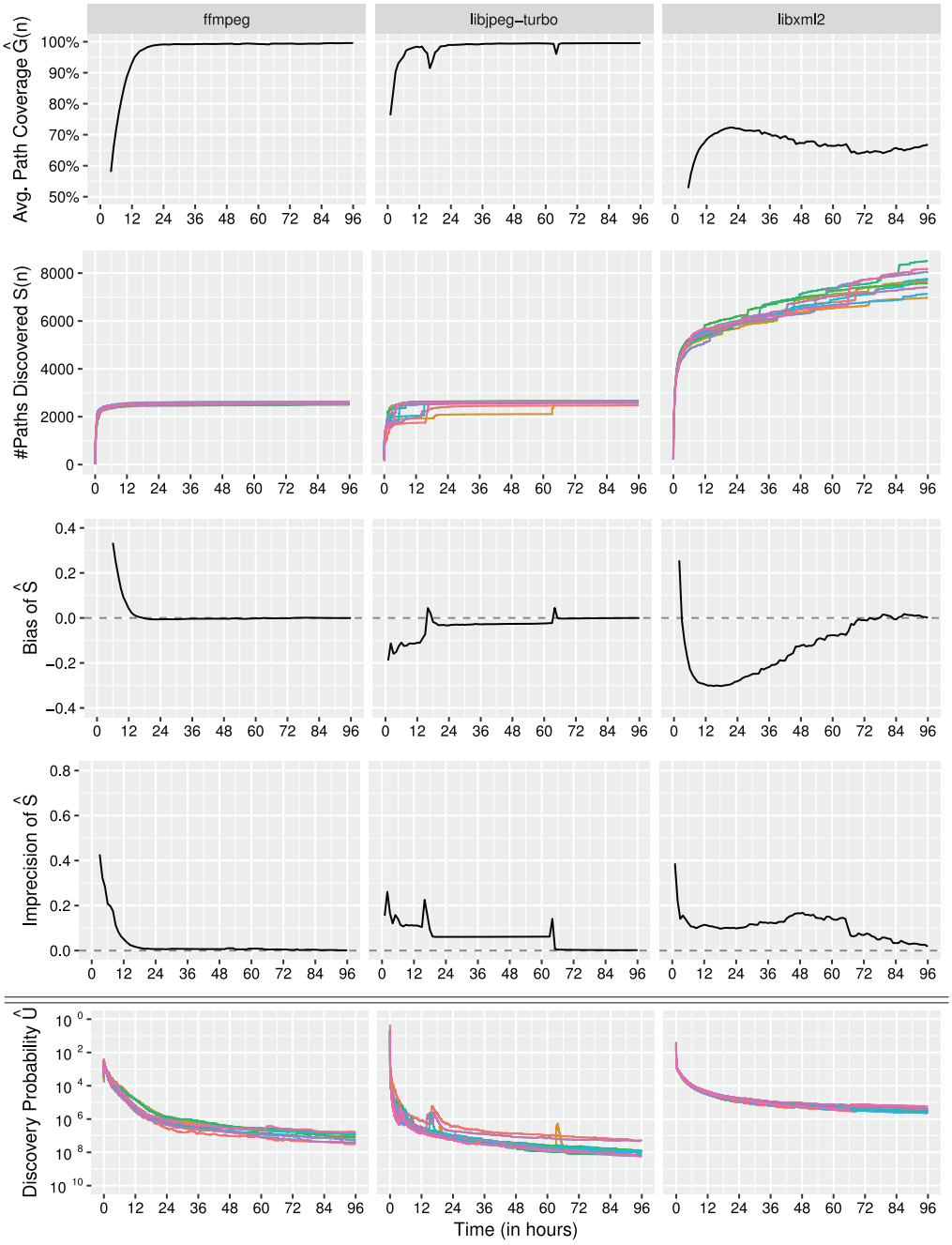
[13]If the environment variable AFL_EXIT_WHEN_DONE is set, AFL automatically aborts the current fuzzing campaign at the end of a cycle in which *no* new path was discovered, starting from the 100th cycle.

Table 4. Average of the Path Coverage Estimates $\hat{G}(n) = S(n)/\hat{S}$, the Number of Discovered Paths $S(n)$, the Bias and Precision of $\hat{S}$ over Time, and the Discovery Probability Estimate $\hat{U}(n)$ within the First 96 Hours (4 Days)



The colored curves in the second and last row represent one fuzzing campaign each.

Table 5. Average of the Path Coverage Estimates $\hat{G}(n) = S(n)/\hat{S}$, the Number of Discovered Paths $S(n)$, the Bias and Precision of $\hat{S}$ over Time, and the Discovery Probability Estimate $\hat{U}(n)$ within the First 96 Hours (4 Days)



The colored curves in the second and last row represent one fuzzing campaign each.

Campaign Ramp-up. In the beginning of a fuzzing campaign, there is often substantial bias, both positive and negative. At *1 minute*, the total number of paths S for json, openssl, and libjpeg-turbo is *substantially overestimated* with a positive mean bias of 6.2, 1.6, and 1.4, respectively. At *1 minute*, the total number of paths S for ffmpeg, libxml2, and wireshark is *underestimated* with a considerable negative mean bias of −0.9, −0.7, and −0.2, respectively. First, there is simply not sufficient data to extrapolate well [20]; i.e., there is no discernible asymptote that can be used for a good estimate of S. Second, when Pythia goes through the circular queue (i.e., the seed corpus) for the first time, the quality of the seeds varies. This means that the number of paths discovered by fuzzing one seed generally does not represent the number of paths discovered by fuzzing another seed. So the estimate $\hat{S}$ is biased. Seed quality has less impact as more queue cycles are completed. Third, Pythia is a coverage-based fuzzer and thus adaptively biased. Test inputs that increase coverage are added to the seed corpus. So the fuzzer's capability to discover paths effectively improves over time [10]. Hence, early estimates of S, particularly until the 12-hour mark, are often substantially biased. Note that an overestimate of S yields an underestimate of path coverage G.

Adaptive bias. Pythia (AFL) is a coverage-based fuzzer, which means that the relative species abundance changes during the fuzzing campaign (see Section 9.5). New seeds may be added to the corpus that enable the discovery of a path that might otherwise be difficult to discover [10]. Sudden increases in the number of discovered paths can cause the current asymptote $\hat{S}$ to systematically *underestimate* S. This happens, for instance, when an interesting test input was generated that contains the correct value for a "magic number" [101]. As we can see for libjpeg-turbo and libxml2 in the second row of Table 5, the magnitude of the increase can be quite substantial. The estimator $\hat{S}$ is negatively biased because of a "false" asymptote for the first 18 hours (libjpeg-turbo, third row). This results in sudden drops in path coverage $\hat{G}$ when many new paths are discovered in short intervals (libjpeg-turbo, first row).

Lower bound. The *Chao1*-estimator $\hat{S}$ of species richness is designed (and proved) to provide a practical lower bound rather than an imprecise point estimate [16]. In fact, Chao1 *is* an unbiased point estimator *if* rare species (undetected and singleton species) have approximately equal abundance [20]. If very rare species are unevenly distributed, the available data simply does not contain sufficient information. So the estimator $\hat{S}$ might be negatively biased (i.e., systematically underestimate the true number of species S). We can see this negative bias clearly for libxml2 and openssl (third row in Tables 4 and 5). For openssl, this results in a path coverage that remains apparently constant between 85% and 95% (first row) despite more paths being discovered (second row). The estimate for openssl is negatively biased because there is no discernible horizontal asymptote that could function as a less biased estimate $\hat{S}$ (second row). For libxml2, there is no discernible asymptote either. We continued the libxml2 experiments past the 96-hour for a total of 800 hours (i.e., 33 days) to see whether the low path coverage value is warranted. Indeed, the number of paths discovered increased from an average 7.6k to an average 10.8k paths. It is interesting to note that the discovery probability (bottom row) for libxml2 and openssl is up to *four order of magnitude higher* than for the other four subjects. Hence, we attribute the negative bias for these two subjects to the large number of rare species that are unequally distributed. We expect the negative mean bias in $\hat{S}$ to be smaller if the improved estimator *iChao1* [32] is used.

However, independent of the source, the bias always tends to 0 as the number of generated test inputs n increases. In the case of libjpeg-turbo, the mean bias seems to remain negative from 18 hours onward (row 3); the mean bias actually goes to about 0 after about 65 hours due to a sudden increase from 2k to 2.5k discovered paths for one fuzzing campaign (yellow line for libjpeg-turbo in second row). The tendency of the mean bias toward 0 as the number of generated test inputs n increases is empirical evidence of the *statistical consistency* of the estimator $\hat{S}$ despite the adaptive

sampling bias of Pythia and despite *Chao1* being a biased estimator. We expect the positive and negative bias in $\hat{S}$ to be smaller if a coverage-based estimator such as *ACE* [25] is used.

RQ.4 Precision of $\hat{S}$. For all subjects, the imprecision reduces to at most 10% of S within the first 12 hours and to at most 1% of S within the first 100 hours. The imprecision is high particularly in the beginning while the number of discovered paths increases significantly within a relatively small time interval. However, in all cases, the imprecision tends to 0 as the number of generated test inputs increases over time. The precision of an estimator quantifies the variance of the provided estimates. A high precision means that the estimates are very similar across different fuzzing campaigns. The fourth row in Tables 4 and 5 shows the *im*precision of the estimators for our subjects. When the estimator's imprecision is high, its *precision* is low, and vice versa.

As we can see for all subjects, the imprecision is high when the slope of the number of paths discovered over time is steep (second row). This is the case in the first few hours of the fuzzing campaign when most paths are discovered (e.g., ffmpeg), and later when there are sudden increases (e.g., libjpeg-turbo). The single outlier run for libjpeg-turbo (second row, yellow line) illustrates an important challenge when computing species coverage for coverage-based graybox fuzzers, such as Pythia. The *Chao1*-estimator essentially estimates the y-intercept of the horizontal asymptote of the curve describing the number of paths discovered over time (second row). The number of paths discovered might seem to approach a clear (but "false") asymptote when actually there is a sudden increase several hours later. As we can see, path coverage is bias-corrected once the sudden increase has happened (first to fourth rows in libjpeg-turbo between 12 and 18 hours). Until then, the security researcher might *incorrectly* presume that the fuzzing campaign has exceeded a path coverage threshold that is required to mark the campaign as completed. However, we can also see in the second row that some programs are more prone to such sudden increases than others, and those are mostly constrained within the first few hours.

In all cases, the imprecision tends toward 0 as the number of generated test inputs n increases. After 100 hours, the imprecision is generally less than 0.01, indicating small variance and high precision. Again, in the case of libjpeg-turbo, the imprecision remaining just below 0.1 from 18 hours onward (row 3) can be explained with the outlier campaign that converges only after 65 hours. The tendency of the imprecision towards 0 as the number of generated test inputs n increases is empirical evidence of the *representativeness* of the estimation computed for one fuzzing campaign for other fuzzing campaigns of the same length.

6.4 Extrapolator Evaluation

Let t be the time that the fuzzer has spent generating n test inputs within the fuzzing campaign. Since the time to generate a test input is fairly constant across all our fuzzing campaigns, we estimate the number m^* of test inputs generated in the interval from t to $t + t^*$, where $t^* \in \{30min, 1hr, 2hrs, 4hrs\}$ as $m^* = nt^*/t$. At time t, we compute the *mean bias* for the estimate $\hat{S}(n + m^*)$ of the number of discovered paths $S(n + m^*)$ if m^* *more* test inputs were generated as the average bias over $N = 10$ runs. At time t, we compute the *imprecision* of $\hat{S}(n + m^*)$ as the standard deviation of the bias:

$$mean\ bias = \sum_{i=1}^{N} \frac{\hat{S}_i(n + m^*) - S_i(n + m^*)}{NS_i(n + m^*)} \tag{21}$$

$$imprecision = \sqrt{\frac{\sum_{i=1}^{N} \left(\frac{\hat{S}_i(n + m^*) - S_i(n + m^*)}{S_i(n + m^*)} - \sum_{j=1}^{N} \frac{\hat{S}_j(n + m^*) - S_j(n + m^*)}{NS_j(n + m^*)} \right)^2}{N - 1}}, \tag{22}$$

where $S(n + m^*)$ is *empirically* determined at time $t + t^*$ and thus provides the true value of the estimation target.

RQ.5 Bias $\hat{S}(n + m^*)$. The number of paths discovered and thus path coverage can be effectively extrapolated with low bias. The magnitude of the bias increases with the extrapolation interval and decreases as more test inputs n are generated. Table 6 shows the mean bias within the first 48 hours (top) and the first 12 hours (bottom) of the fuzzing campaign. We chose these two intervals because of the difference in the magnitude of the bias in the first few hours. In fact, the bottom four rows feature a large range of the bias between −20% and 50% of the true, empirical $S(n + m^*)$, while the top four rows feature a much smaller range between −8% and 8% of $S(n + m^*)$.

As we can see in Table 6, $\hat{S}(n + m^*)$ might substantially overestimate the number of paths discovered in the first few hours. We explain this strong positive bias of $\hat{S}(n + m^*)$ with the strong positive bias of the estimate $\hat{S}$ of the total number of paths, which forms an important component in the extrapolation methodology proposed by Shen et al. [92] (see Table 4, third row). After the initial overestimation, $\hat{S}(n + m^*)$ is generally slightly (but systematically) underestimated. Effectively, the estimator begins to provide a *conservative estimate* of the increase in path coverage. We explain this small negative bias with Pythia being a coverage-based graybox fuzzer. Indeed, we expect a blackbox fuzzer (without adaptive sampling bias) to detect fewer paths per unit time. Another source of negative bias is a sudden increase in the number of paths discovered that would be difficult to anticipate (e.g., for libjpeg-turbo compare Table 6, top, and Table 5, second row). We also notice that the magnitude of the bias increases with the extrapolation interval. The reason is fairly obvious: the quality of the extrapolation will be worse the further we want to look into the future. However, the rule of thumb in ecology [34] to limit the extrapolation to within twice the current sampling effort does *not* find very strong empirical support for our six subjects in the domain of automated software testing and analysis.

For all six subjects and all four extrapolation intervals, the bias remains within ±2% of the empirical value $S(n + m^*)$ from 18 hours onward. The tendency of the bias toward 0 as the number n of generated test inputs increases might be explained with an expected deceleration of the number of paths $S(n)$ discovered over time approaching an asymptotic total number of paths S (see second row in Tables 4 and 5).

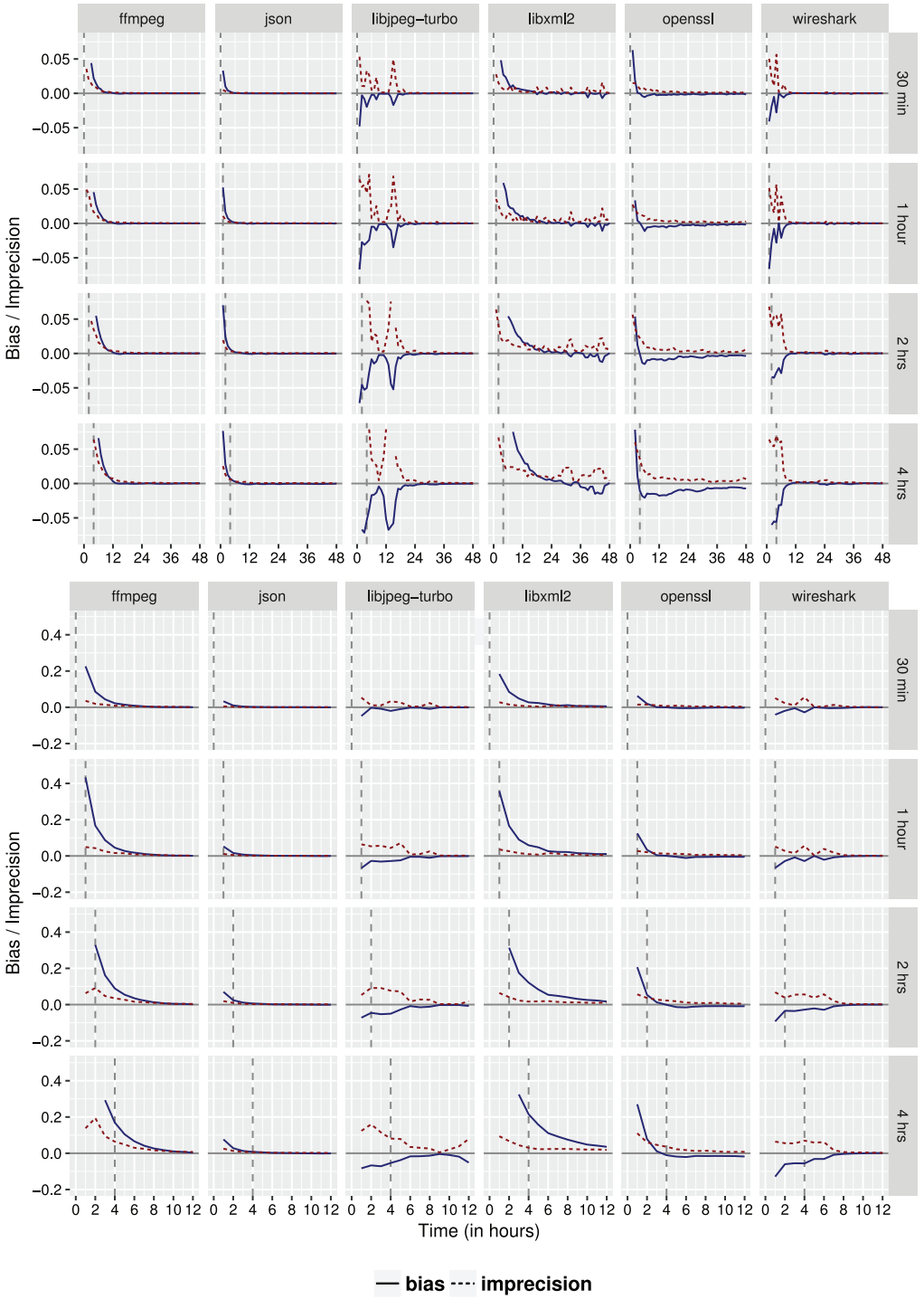
RQ.6 Precision $\hat{S}(n + m^*)$. The number of paths discovered and thus path coverage can be effectively extrapolated with high precision. The magnitude of the im*precision increases with the extrapolation interval and decreases as more test inputs n are generated.* The imprecision does not seem to be affected as substantially as the bias by the initial surge of path discoveries (see ffmpeg and openssl, bottom rows). However, the magnitude of the imprecision generally mirrors that of the bias, suggesting that bias and imprecision have the same sources. Like the bias, the imprecision tends toward 0 as the number n of generated test inputs increases.

6.5 Result Summary

The objectives of this empirical evaluation were (1) to test my main hypothesis within the STADS framework, (2) to evaluate several methodologies from ecology within the STADS framework, and (3) to investigate the impact of the adaptive sampling bias.

6.5.1 Main Hypothesis. I hypothesize that within the STADS framework, rare, discovered species contain almost all information about the detectable species that remain undiscovered. Rare species are those to which only a small number of generated test inputs belong. In our experiments, all methodologies used to extrapolate from the discovered to undiscovered species (i.e., from tested to untested program behaviors) are based on the number of singletons or doubleton species. The

Table 6. Bias and Imprecision for 6 Subjects at 4 Extrapolation Intervals for ≤ 48hrs (top) and ≤ 12hrs (Bottom)



experimental results show good estimator performance for these methodologies and thus *support my main hypothesis*.

6.5.2 Estimator Evaluation. We find that *discovery probability* provides a useful indicator of the residual risk that a discoverable vulnerability exists but remains undiscovered in the ongoing fuzzing campaign. The discovery probability estimate measured for one fuzzing campaign is fairly representative for other fuzzing campaigns of similar length.[14] While the estimates for 10 runs range over three orders of magnitude across subjects, they all range within the same order of magnitude for the same subject. The similarity between estimates of different runs for the same subject (i.e., the precision) increases as the number n of generated inputs increases.

We find that *path coverage* provides a useful indicator of the progress of a fuzzing campaign toward completion that can be used to decide effectively whether to abort or continue a fuzzing campaign. The path coverage estimate can be positively and negatively *biased*. The bias is most substantial during campaign ramp-up, within the first 12 hours, when many paths are discovered. Another source of substantial bias is the existence of many rare species (see discussion in Section 9.1) and the sudden discovery of many paths at once (see discussion in Section 9.5). However, for all subjects the magnitude of the bias reduces as the number n of generated test inputs increases. Similarly, the *precision* is low in the first few hours, while the number of discovered paths increases significantly within a relatively small time interval. However, in all cases, precision increases as n increases.

We find that path coverage can be *extrapolated* with low bias and high precision. Like the path coverage estimate, the extrapolation can be positively and negatively *biased*. Sudden surges in the number of discovered paths are not anticipated, resulting in negative bias and some *im*precision. A substantial overestimation of the total number of paths in the first few hours might result in a substantial positive bias and some imprecision. The magnitude of bias and imprecision increases with the extrapolation interval and decreases as more test inputs n are generated. We do not find very strong evidence that the rule of thumb in ecology [34] to keep the extrapolation within twice the sampling effort applies to automated software testing and analysis. Other sources of bias and imprecision seem to have a stronger impact.

6.5.3 Impact of Adaptive Sampling Bias. An underlying assumption of most methodologies in the STADS framework is that the probability p_i to generate a test input that belongs to species $\mathcal{D}_i$ does *not* change substantially during the fuzzing campaign. However, it does for feedback-directed fuzzers, such as Pythia (which is based on AFL). A *feedback-directed fuzzer* continuously adapts the strategy to generate new test input based on feedback for previous test inputs. For instance, Pythia augments the existing seed corpus with generated test inputs that increased branch coverage. New test inputs are generated by random mutations of the seed inputs that are continuously selected from a circular queue that represents the (extended) seed corpus. As the seed corpus grows, the relative species abundance $\{p_i\}_{i=1}^{S}$ changes as well. This is called an *adaptive sampling bias*, because the sampling strategy changes adaptively during the sampling itself.

We find that *in the beginning* of a fuzzing campaign, the adaptive sampling bias has a large impact on estimator performance. In the first few hours, the total number of species $\hat{S}$ is often substantially underestimated, which is explained by the improving capability of Pythia to discover new species as new seeds are added to the seed corpus. In the beginning, a large number of new seeds are added as new species are discovered.

[14] More specifically, discovery probability is fairly representative for other fuzzing campaigns where the same program is fuzzed for the same time using the same fuzzer and seed corpus (if any).

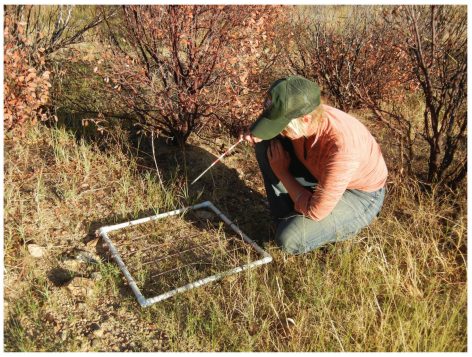


Fig. 6. Quadrats positioned at random locations in the assemblage. Present species are recorded for each quadrat. Image credit: NPS Sonoran Desert Network (Licence: CC-BY-2.0).

We find that *as more test inputs* are generated, the impact reduces. For PYTHIA, the species discovery curve is strictly correlated with the adaptive sampling bias. For every new species that is discovered, a seed is added to the seed corpus. Hence, as species discovery decelerates over time, the adaptive bias reduces just as well. For some subjects, the number of discovered species seems to approach a *false* asymptote, leading to a negatively biased estimate of species richness $\hat{S}$ (and thus a positively biased estimate of species coverage $\hat{G}$) when suddenly many more species are discovered, e.g., because of a magic number discovered [101]. This is also explained by the adaptive sampling bias, and its impact reduces over time. A more general discussion on the impact of adaptive sampling bias follows in Section 9.5.

7 BERNOULLI PRODUCT MODEL: ONE INPUT, MULTIPLE SPECIES

So far, we have discussed the multinomial model[15] in the STADS framework, where *each input belongs to exactly one species*; e.g., an input can exercise only exactly one path. However, there are many other concrete testing objectives where *each input belongs to one or more species*. For instance, a single input can exercise multiple coverage goals, such as program statements, branches, or methods; a single input can kill multiple mutants [66], witness multiple information flows [73], violate multiple assertions, expose multiple bugs, and traverse multiple program states. In ecology, it is a *sampling unit* that can contain multiple species. A sampling unit is usually a physical trap, net, quadrat, or plot. These sampling units are distributed in the assemblage and studied exhaustively—in lieu of the assemblage itself. When only the presence (or absence) of species can be determined in a sampling unit, ecologists call the data as *incidence data* and utilize the *Bernoulli product model* [21]. In the Bernoulli product model, within STADS, a generated test input is considered as a *sampling unit*.

In the following, we extend our STADS statistical framework to account for testing objectives that yield multiple species identified for a single input. Let n be the number of inputs that have been generated throughout the current fuzzing campaign, $S(n)$ be the number of species that have been discovered, and S be the total number of species. Define $\{W_{ij} \mid i = 1, 2, \ldots, S \wedge j = 1, 2, \ldots, n\}$ as the *incidence matrix* where $W_{ij} = 1$ if the jth generated test input belongs to species $\mathcal{D}_i$, and $W_{ij} = 0$ otherwise. Let Y_i be the number of generated test inputs that belong to species $\mathcal{D}_i$ for $i : 1 \leq i \leq S$, and then $Y_i = \sum_{j=1}^{n} W_{ij}$. For species that exist but remain undiscovered in the current fuzzing

[15]The multinomial model is introduced in Section 3.5. Specific estimators and extrapolators for the multinomial model are discussed in Sections 4 and 5, respectively. Several of those are evaluated in Section 6.

campaign, we have $Y = 0$. Define the *incidence frequency count* Q_k for $k : 0 \leq k \leq n$ as the number of species to which exactly k test inputs belong that have been generated throughout the current fuzzing campaign. More formally, $Q_k = \sum_{i=1}^{S} I(Y_i = k)$. Hence, $n \leq \sum_{k=1}^{n} kQ_k = \sum_{i=1}^{S} Y_i$ and $S(n) = \sum_{k=1}^{n} Q_k$. The incidence frequency count Q_k is analogous to the abundance frequency count f_k for the multinomial model. The unobservable zero frequency count Q_0 denotes the number of species that remain undiscovered in the current fuzzing campaign. We call Q_1 the number of *singleton species* and Q_2 the number of *doubleton species*.

The probability that the fuzzer generates a test input that belongs to species $\mathcal{D}_i$ is p_i for $i : 1 \leq i \leq S$. Note that $\sum_{i=1}^{S} p_i \geq 1$. We assume that each W_{ij} is a *Bernoulli random variable* with probability p_i that $W_{ij} = 1$ (and analogously with probability $1 - p_i$ that $W_{ij} = 0$). Thus, the probability distribution for the incidence matrix is

$$P(W_{ij} = w_{ij}; i = 1, 2, \ldots, S; j = 1, 2, \ldots, n) = \prod_{j=1}^{n} \prod_{i=1}^{S} p_i^{w_{ij}} (1 - p_i)^{1 - w_{ij}} \tag{23}$$

$$= \prod_{i=1}^{S} p_i^{y_i} (1 - p_i)^{n - y_i}, \qquad \text{where } y_i = \sum_{j=1}^{n} w_{ij}. \tag{24}$$

From Equation (24), we can see that the row sums $(Y_1, Y_2, \ldots, Y_S)$ are *sufficient statistics*. This renders the incidence frequency counts Q_k suitable components for the estimators of species richness. The number of generated test inputs Y_i that belong to species $\mathcal{D}_i$ follows a binomial distribution:

$$P(Y_i = y_i) = \binom{n}{y_i} p_i^{y_i} (1 - p_i)^{n - y_i}, \qquad \text{where } i = 1, 2, \ldots, S. \tag{25}$$

Chao and Colwell [21] provide more details about the Bernoulli product model and its utility in the ecologic context.

7.1 Estimation in the Bernoulli Product Model

Estimating *S*. In the Bernoulli product model of the STADS framework, the estimation of species richness S (i.e., the asymptotic total number of species) can be done using the *Chao2* and *iChao2*-estimators, which were derived by Chao [17] and Chui et al. [32]:

$$\hat{S}_{\text{Chao2}} \approx \begin{cases} S(n) + Q_1^2/(2Q_2) & \text{if } Q_2 > 0 \\ S(n) + Q_1(Q_1 - 1)/2 & \text{if } Q_2 = 0 \end{cases} \tag{26}$$

$$\hat{S}_{\text{iChao2}} \approx \hat{S}_{\text{Chao2}} + \frac{Q_3}{4Q_4} \times \max\left(Q_1 - \frac{Q_2 Q_3}{2Q_4}, 0\right). \tag{27}$$

Chao [17] showed that the Chao2 estimator $\hat{S}_{\text{Chao2}}$ provides a nonparametric lower bound on the total number of species S in an assemblage. Very recently, Chao and colleagues showed that the Chao2 estimator is an *unbiased point estimator* as long as very rare species (specifically, undetected and singleton species) have approximately equal detection probability [20].

Alternative estimators of species richness S in the Bernoulli product model include jacknife estimators [14, 83, 97] and coverage-based estimators, such as ICE [67] and ICE-1 [52], that are particularly suitable when species diversity is high (i.e., when species evenness J is low).

Estimating *U*. For inputs that can belong to one or more species, the estimate $\hat{U}(n)$ of the discovery probability requires information not only about singleton and doubleton species but also about *all discovered* species [21]. In the Bernoulli product model of the STADS framework, the

discovery probability is estimated as

$$\hat{U}(n) = \frac{Q_1}{V}\left[\frac{n\hat{Q}_0}{n\hat{Q}_0 + Q_1}\right] \tag{28}$$

$$\approx \frac{Q_1}{V}, \tag{29}$$

where in the current fuzzing campaign $V = \sum_{k=1}^{n} kQ_k = \sum_{i=1}^{S}\sum_{j=1}^{n} W_{ij}$ denotes the sum of all entries in the incidence matrix W_{ij}, n is the number of test inputs generated, the number of undetected species can be estimated as $\hat{Q}_0 = \hat{S} - S(n)$, and Q_1 and Q_2 are the incidence frequency counts of singletons and doubletons, respectively. Note that the sum of all entries V in the incidence matrix does *not* require storing the complete incidence matrix W_{ij}. Instead, V can be aggregated during the fuzzing campaign. Also notice the similarity of the approximation to the Good-Turing estimator for the multinomial model [49].

7.2 Extrapolation in the Bernoulli Product Model

Extrapolating $S(n)$. In the Bernoulli product model of the STADS framework, to estimate the expected number of discovered species $\hat{S}(n + m^*)$ when n test inputs have already been generated and if m^* more test inputs were to be generated, we have

$$\hat{S}(n + m^*) = S(n) + \hat{Q}_0\left[1 - \left(1 - \frac{Q_1}{n\hat{Q}_0 + Q_1}\right)^{m^*}\right], \tag{30}$$

where for the current fuzzing campaign n is the number of generated test inputs, $S(n)$ is the number of discovered species, $\hat{Q}_0 = \hat{S} - S(n)$ is the expected number of undiscovered species, and Q_1 is the number of singletons.

Extrapolating $U(n)$. In the Bernoulli product model of the STADS framework, the estimate of the expected discovery probability $\hat{U}(n + m^*)$ if m^* more test inputs were generated is computed as

$$\hat{U}(n + m^*) = \frac{Q_1}{V}\left[\frac{n\hat{Q}_0}{n\hat{Q}_0 + Q_1}\right]^{m^*+1}, \tag{31}$$

where for the current fuzzing campaign, $V = \sum_{k=1}^{n} kQ_k = \sum_{i=1}^{S}\sum_{j=1}^{n} W_{ij}$ denotes the sum of all entries in the incidence matrix W_{ij}, n is the number of generated test inputs, $S(n)$ is the number of discovered species, $\hat{Q}_0 = \hat{S} - S(n)$ is the expected number of undiscovered species, and Q_1 is the number of singletons.

Estimating m_{G^*} when $G^* = S(n + m_{G^*})/\hat{S}$ is given. To reach a fraction G^* of estimated total number of species $\hat{S}$ where $\hat{G}(n) < G^* \leq 1$, in the Bernoulli product model of the STADS framework the required number m_{G^*} of further test inputs is estimated as [22]

$$m_{G^*} \approx \frac{\log\left[1 - \frac{n}{(n-1)}\frac{2Q_2}{Q_1^2}(G^*\hat{S} - S(n))\right]}{\log\left[1 - \frac{2Q_2}{(n-1)Q_1 + 2Q_2}\right]}, \tag{32}$$

where in the current fuzzing campaign n is the number of generated test inputs, $S(n)$ is the number of discovered species, and Q_1 and Q_2 are the incidence frequency counts for singleton and doubleton species, respectively. The *expected fuzzing time* can be computed by multiplying the expected number of test inputs with the average time the fuzzer takes to generate a test input.

8 RELATED WORK

To the best of our knowledge, in the domain of software testing and analysis, there exists *no previous work on estimating the asymptotic total number of species, or on extrapolating the number of species discovered over time* for any definition of species. In fact, Whalen [99] says about the future of verification and validation that one of the biggest problems today is that there is *no sound basis to extrapolate from tested to untested cases.* We strongly believe that the STADS framework provides a statistically well-grounded basis to extrapolate from tested to untested program behaviors.

8.1 Residual Risk Assessment

Finding no vulnerabilities in a (long-running) fuzzing campaign does not mean that none exists. In the STADS framework, the *discovery probability* $U(n)$ measures the probability to discover a new species with the $n+1$'th generated test input where n is the number of test inputs that have been generated throughout the fuzzing campaign. If the dynamic analysis is able to identify vulnerabilities, then an accurate estimate of the discovery probability provides a *statistical guarantee* that no detectable vulnerability exists if none has been discovered. In other words, security researchers can use the STADS statistical framework for *residual risk assessment.*

There exist several *systematic approaches* to quantify the reliability of a program. However, Filieri et al. [41] recently noted that most existing approaches work on the design and architectural level rather than on the program itself. The authors present a program-level reliability estimation technique that uses *probabilistic symbolic execution* [45] to compute the probability of satisfying any of the path conditions corresponding to non-error-exposing paths. In other words, the approach computes the proportion of inputs that exercise paths that do not expose an error. Since probabilistic symbolic execution leverages model counting to determine the proportion of inputs exercising a path, the approach works only for very small input spaces. In contrast, we propose a lightweight statistical technique to estimate the confidence that a fuzzing campaign inspires in the correctness of a program, and that scales to programs of arbitrary size.

There exist several *statistical approaches* to quantify the reliability of a program. For instance, the problem of estimating the probability $P(n)$ to discover an error with the $n+1$'th test input, given that no errors have been found after generating n test inputs, can be cast as a variant of the *sunrise problem,*[16] which is classically solved with Laplace's *rule of succession* [65]: $P(n) = 1/(n+2)$. Suppose s of n generated inputs expose an error; then the probability to generate another error-exposing input follows a *beta*-distribution $\text{Beta}(s+1, n-s+1)$, the posterior of which has the expected value $(s+1)/(n+2)$.

Miller et al. [78] recognized the utility of the *beta*-distribution to quantify the probability of failure in the absence of failures and furthermore discuss the case where the test distribution does not overlap with the operational distribution (i.e., the fuzzer might not generate "typical" inputs, but we are interested in the program's reliability for typical inputs). Littlewood and Wright [68] also utilize the *beta*-distribution but discuss how to update the previous estimate of the probability of failure *after* a bug was found and fixed. In contrast to these existing works, the STADS framework leverages information on the *problem structure* by identifying the species for an input. Hence, we can provide more accurate estimates of the residual risk that a detectable vulnerability has remained undetected and of the confidence that a fuzzing campaign inspires in the correctness of the program. Moreover, the STADS framework is *more general* and also provides methodologies to estimate the total number of species and to extrapolate the number of species discovered at some future point in time.

[16] Given that we have seen the sun rise for n consecutive days, what is the probability that the sun will rise tomorrow?

It is interesting to note that only 25 years ago, the execution of *100 million* ($n = 10^8$) test inputs was utterly *unthinkable* [57]. Hamlet and Voas conjecture that "direct reliability assessment by random testing of software is impractical. The levels we would like to achieve, on the order of 10^6–10^8 executions without failure, cannot be established in a reasonable time. Some limitations of reliability testing can be overcome, but the "ultrareliable" region above 10^8 failure-free executions is likely to remain forever untestable" [57]. Today, Google's continuous fuzzing platform OSS-Fuzz generates 10 *trillion* (10^{10}) test inputs *per day* [118].

When test inputs are generated manually, a general suggestion is to increase the code coverage. The most popular measures are *code coverage* metrics, such as statement, branch, or MC/DC coverage, and *fault coverage* metrics, such as relative mutation adequacy [66]. The hope is that the fault revelation of a set of test inputs increases as its coverage increases. In other words, maximal coverage should inspire maximal confidence. However, many recent empirical studies found that such coverage metrics are in fact *poor indicators* of test suite effectiveness *in the context of automated software test generation* [29, 44, 64]. The empirical results may be explained by early theoretical investigations of testing effectiveness [39, 56, 129]. Böhme and Paul [8], similar to Hamlet and Taylor [56], argue that a set of *successful* test inputs (i.e., no input exposes an error) that achieves 100% branch coverage, 100% MC/DC coverage, and even 100% relative mutation adequacy does not inspire *any* degree of confidence in the correctness of the tested program. Indeed, vulnerabilities may still exist. In contrast, STADS provides a statistically well-grounded framework to assess the residual risk that a detectable vulnerability exists even if none has been found.

8.2 Partition Testing

In partition testing, the program's input domain is partitioned into overlapping or nonoverlapping subdomains [129]. The task of a *tester* is to select one or more elements from each subdomain. In the STADS framework, we would say that each and only input in the same subdomain belongs to the same species. However, unlike in the STADS framework, each input subdomain in partition testing is associated with a probability θ_i that an input in this subdomain reveals an error [39]. Partition testing is a probabilistic model of software testing that allows one to investigate the tester's ability to detect faults.

Analyzing the effectiveness of random testing, Duran and Ntfos [39] used the partition testing model to show that the expected number of errors $g(n)$ discovered after n test inputs have been sampled *uniformly at random*, for the case of nonoverlapping subdomains, is given as

$$g(n) = S - \sum_{i=1}^{S} (1 - p_i\theta_i)^n,$$

where S is the total number of subdomains and p_i is the probability that the randomly sampled input lies in subdomain $\mathcal{D}_i$. Duran and Ntfos observed experimentally that a tester who samples one or more inputs from each subdomain performs only slightly better than simple random testing.

Varying several parameters, Hamlet and Taylor [56] repeated the experiments of Duran and Ntafos and confirmed that the number of errors found by random and partition testing is very similar. In fact, the authors conclude that "partition testing does not inspire confidence." Weyuker and Jeng [129] found that the effectiveness of partition testing varies depending on the fault rate θ_i for each subdomain. Subsequently, several authors discussed conditions under which partition testing is generally more effective than random testing (e.g., [30, 53]). Empirical investigations [29, 44, 64] of the effectiveness of partition testing have since confirmed Hamlet and Taylor's conclusion [56].

In our previous work [8], we leverage the partition testing model to conduct the first probabilistic analysis of the efficiency of automated software testing. We identify bounds on the time the *most effective* systematic testing technique can take per test input to remain *more efficient* than random testing. We develop a hypothetical hybrid testing technique that is more efficient than both random and systematic testing. We also suggest a primitive curve-fitting method to extrapolate the partitions discovered over time. However, in the present article, we introduce more sophisticated sampling-theoretic extrapolation methodologies.

While the partition testing model allows *probabilistic analyses*, the STADS framework allows *statistical analyses*, including estimation and extrapolation. Probabilistic and statistical analysis are inverse to each other. In a *probabilistic analysis,* we consider some underlying random process where the randomness is modeled by random variables, and we resolve what happens. In a *statistical analysis,* we observe something that has happened and try to resolve what underlying process would explain those observations. In contrast to existing work, we present practical estimation and extrapolation methodologies. The STADS framework is the first work in automated software testing that allows one to extrapolate from tested to untested program behavior with quantifiable accuracy.

9 CHALLENGES AND OPPORTUNITIES

9.1 Programs as Megadiverse Assemblages

The STADS framework exhibits some peculiar features that make the application of existing ecologic methodologies more challenging: specifically, one has to deal with extremely large populations containing a huge number of species (e.g., millions of program branches or exponentially more distinct paths). Specifically, compared to common assemblages in ecology, we expect species richness S to be *very high* and species evenness J to be *very low* in the STADS model. In other words, there are a huge number of very rare species and only a few extremely abundant species.

9.1.1 Megadiversity. In ecology, we call an assemblage with high richness and low evenness a *megadiverse assemblage*. For instance, arthropods (i.e., bugs, millipedes, spiders, etc.) in a tropical forest would be considered a megadiverse assemblage [4]. There are an estimated 6.1 million tropical arthropod species, most of which are rare [54, 55]. Such assemblages are subject to several statistical challenges during estimation and extrapolation, particularly due to the relatively small sample size [35, 71]. However, compared to species inventories common in ecology, the sample size n in the STADS framework can be *very large*, which should render our data precious for ecologic biostatisticians. For instance, it took 102 ecology researchers 66 person-years to sample 129,494 arthropods representing 6,144 species from 0.48 ha of tropical rain forest [4]. In stark contrast, a fuzzer can take a million samples in only a few minutes.

9.1.2 Scarcity. The main objective of fuzzing is to discover vulnerabilities in a program. Vulnerabilities are arguably *very rare species* in the STADS framework. Similarly, a primary objective of many ecological surveys is to identify species that are so rare that they are close to extinction. Once identified, the necessary conservation policies are proposed and implemented to counter the diminishing biodiversity.

For the STADS framework, we should identify, develop, and employ estimators that are better suitable if many rare species are present. Colwell et al. [34] suggest to employ coverage-based estimators of species richness [25, 28, 52, 67] if one expects many rare species. Mao and Collwell [71] propose a mixture model to compute, with confidence intervals, a lower bound on species richness when there are many rare species. In a *mixture model*, species abundance or occurrence distributions are modeled as a weighted mixture of statistical distributions. Ohannessian [80]

observes that the Good-Turing estimator of discovery probability performs well even in the presence of many rare species. Chao et al. [23] generalize the Good-Turing estimator to develop the Good-Turing sample coverage theory. In the future, coverage-based and mixture-model-based as well as other rare event estimators [11, 80] and their performance within the STADS framework can be studied. Other suitable estimators for megadiverse assemblages with many rare species can be developed that would benefit both fields of research tremendously.

9.1.3 Endemism. Another challenge in fuzzing is the random generation of "magic numbers," such as file identifiers [101]. Only if the magic number is correct will the generated test input exercise interesting program behaviors. Only if the magic number is correct will many new species be discovered. A similar challenge exists in ecology. *Endemism* is the ecological state of a species being unique to a defined geographic location. For instance, the rain forest of Madagascar hosts a large number of (endemic) species that can only be found in Madagascar [51]. A global survey of the biodiversity in rain forests would miss many species if the "magic island" of Madagascar remains uninvestigated. A survey of the biodiversity in the Sahara Desert would miss many species if oases remain uninvestigated [40]. Hence, it is sensible only to provide an improved lower bound of the total number of species S [16, 17].

9.1.4 Opportunities. Strategies could be established that allow one to choose the best estimator at any time during the fuzzing campaign based on estimates of species evenness J and discovery probability U (e.g., [12]). Several estimators of the same quantity may be used to derive a "best estimate" [4]. In the STADS framework, the program's source code and program binary provide an additional source of information that can be used to improve estimator performance. In the future, the dependence of estimator bias and precision on the sample completeness C can be investigated to develop better bias correction mechanisms.

9.2 Species Identification and Oracle Problem

In the STADS framework, the species for an input t is identified using a combination of dynamic analyzers that record during execution of t the observed program properties of interest. For instance, to detect bugs in C programs, the compiler can be asked to inject so-called *sanitizers* [91, 94], assertions that crash the program when a bug is detected. There are different sanitizers [103, 108], e.g.,

- to detect memory-related errors, such as overflows and use-after-free (AddressSanitizer),
- to detect race conditions and deadlocks (ThreadSanitizer),
- to detect undefined behaviors (UndefinedBehaviorSanitizer),
- to detect memory leaks (LeakSanitizer), or
- to check control-flow integrity (CFISanitizer).

Whether a bug constitutes an exploitable vulnerability can be determined with excellent precision and good recall using another dynamic analysis, e.g., the CERT Triage Tools [102].

9.2.1 Misidentification and Guarantees. The correct identification of the species for an input is an important challenge in both disciplines. For instance, in ecology, Austen et al. [1] observed that even taxonomic experts were correct in only 60% of cases when asked whether two images showed the same or different species of bumblebees. Similarly, in the STADS framework, a dynamic analysis may misidentify the species for an input. For instance, misidentification in software testing may lead to input incorrectly classified as *not* exposing a vulnerability (when it actually does). Hence, the statistical guarantees provided by the STADS model hold *modulo* the dynamic analyzer's

capability to identify the correct species for an input. This motivates further research on advanced dynamic analysis techniques that are more effective at vulnerability detection.

It is interesting to note that misidentification is also an important challenge for automated verification. The formal guarantees provided by the verifier are valid only *modulo* the provided specification, which may be incomplete. For instance, the specification may allow one to check whether a race conditions exists (a classic model checking problem)—but not whether a buffer overflow exists (the number one root cause of arbitrary code execution attacks).

9.2.2 Morphospecies and Oracle Problem. In ecology, some individuals cannot be assigned to named species. A *morphospecies* is different from previously discovered species (in its morphology) but not to a sufficient degree that it could be assigned its own species. A similar challenge is known in the software testing domain as the *oracle problem*. Weyuker [128] conjectures that, in general, there exists no mechanism that can accurately decide whether or not the program behavior for an input is correct—whether an observed behavior is a bug or a feature of the program. Barr et al. [3] provide an excellent survey of recent advances in tackling the oracle problem.

9.3 Integrating Other Models into STADS

The STADS framework provides opportunities to explore other topics, models, and related methodologies in ecology. For instance, a typical problem in ecology is the extrapolation of the number of species in an enlarged area of size $A + a^*$, given only a sample of a smaller area of size A [27]. This is modeled as the *continuous Poisson model* [33]. In the Poisson model, the reference sample is not defined by sample size n but by the area A that is sampled. The ith species occurs at a species-specific mean rate $A\lambda_i$, so that the probability distribution is

$$P(X_1 = x_1, \ldots, X_S = x_s) = \prod_{i=1}^{S} (A\lambda_i)^{x_i} \frac{\exp(-A\lambda_i)}{x_i!}. \tag{33}$$

In fuzzing, we often restrict the size of the generated test inputs because larger inputs might take longer to execute, and species appear to be distributed more densely in the space of small inputs. Within the STADS framework, we can leverage the Poisson model to estimate the total number of species for large inputs, given a fuzzing campaign that restricted the fuzzing to only smaller inputs. For instance, in Google's continuous fuzzing platform OSS-Fuzz [118], the main fuzzer LibFuzzer [111] is often configured with a maximum test input size. The Poisson model would allow one to extrapolate the confidence such "restricted" fuzzing campaigns inspire in the absence of vulnerabilities from the small generated test inputs to larger "normal-sized" inputs.

In the future, mixture models [71] can be developed that synthesize better estimates from those provided by the multinomial and the Poisson model. In order to integrate the Poisson and Bernoulli product models successfully into the STADS framework, an empirical evaluation of estimator performance is left for future work.

9.4 Nonadaptive Sampling Bias

The STADS framework fully accounts for *arbitrary fuzzer heuristics*, including the sampling from the operational distribution, as long as the fuzzer does not change the sampling strategy adaptively throughout the fuzzing campaign. For instance, if a compiler fuzzer generates more programs with loops than programs without—because historically programs with loops have always found more compiler bugs—then all statistical claims derived from the STADS framework strictly hold w.r.t. that fuzzer and for that program within the stipulated confidence bounds. The main assumption upon which the (multinomial and Bernoulli product) models of the STADS framework rely is that the relative species abundance $\{p_i\}_{i=1}^{S}$ does not change substantially during the fuzzing campaign.

The compiler fuzzer is simply more likely (greater p_i) to discover a loop-based bug $\mathcal{D}_i$ than a compiler fuzzer without that heuristic, *for all fuzzing campaigns*.

While there may be some bias in the test input generation, there is *no adaptive bias* for blackbox fuzzers. A *blackbox fuzzer* does not leverage feedback from previous test executions to adapt the test generation strategy during the fuzzing campaign. A *generational blackbox fuzzer* generates test inputs either by random sampling [77] or by instantiating elements from an input model, grammar, or protocol [119]. A *mutation-based blackbox fuzzer* generates new test inputs by random perturbations of inputs in the so-called seed corpus. The probability p_i to generate a test input that belong to species $\mathcal{D}_i$ does not change *at all* during the fuzzing campaign.

In ecology, the sampling is usually subject to certain biases as well. A light trap may lure certain species more than others [59]. An ecologist may prefer to sample certain locations in an assemblage over others [88]. An ecologist may be more likely to sample species that are larger in body size (or perhaps prefer to sample only the smaller species in the case of the arachnida class) [74].

9.5 Adaptive Sampling Bias of Feedback-Directed Fuzzers

The *main assumption* in the STADS framework upon which the multinomial, Bernoulli product, and Poisson models rely is that the relative species abundance $\{p_i\}_{i=1}^{S}$ does not change substantially during the fuzzing campaign. However, feedback-directed fuzzers are based on an *adaptive* sampling strategy. A *feedback-directed fuzzer* leverages program feedback from previous test inputs to learn and *adaptively* generate "better" test inputs. The probability p_i to sample an input that belongs to an undiscovered species $\mathcal{D}_i$ may increase.

9.5.1 Search-Based Software Testing. Fuzzers developed in the field of *search-based software testing* (SBST) are feedback directed. A fitness function evaluates how close a test input or set of test inputs is toward satisfying the concrete fuzzing objective, while a meta-heuristic steers the test generation adaptively toward new test inputs with improved fitness. For instance, a *directed graybox fuzzer* [9] evaluates the "distance" of an input to a set of target locations in the program (e.g., potential buffer overflow sites) and uses simulated annealing-based power schedules to generate new test inputs that are "closer" to those target locations. McMinn provides an excellent survey of SBST existing techniques [75] and identifies future challenges [76].

To establish the impact of the adaptive bias, we strongly suggest to evaluate estimator performance for each SBST technique. In the future, customized *bias-corrected estimators* can be developed that allow accurate estimation and extrapolation for SBST techniques.

9.5.2 Coverage-Based Graybox Fuzzing. Feedback directed are also coverage-based graybox fuzzers [10, 100, 111, 122]. A coverage-based graybox fuzzer is typically mutation based and hence starts with a seed corpus. If the fuzzer generates a test input t that belongs to a previously undiscovered species (e.g., by random perturbations of inputs in the corpus), then t is *added* to the seed corpus. Otherwise, t is discarded. At the time when t is added to the seed corpus, the probability p_i to discover any "neighboring" species $\mathcal{D}_i$ slightly increases, compared to *before* t was added. Hence, at first a coverage-based graybox fuzzer might discover more species per unit time than a mutation-based blackbox fuzzer (which is not feedback directed). However, *in the limit,* every coverage-based graybox fuzzer degenerates to a mutation-based blackbox fuzzer. Over time more and more test inputs need to be generated to discover the next species: the fuzzer cycles several times through the same set of seeds without any discoveries for hours, later for days. Hence, *in the limit,* the adaptive bias is nonexistent. Thus, if an estimator is consistent for a fuzzer that is not feedback directed, it is also consistent for a coverage-based graybox fuzzer. The accuracy of a *consistent* estimator increases as sampling effort (i.e., the number of generated test inputs) increases.

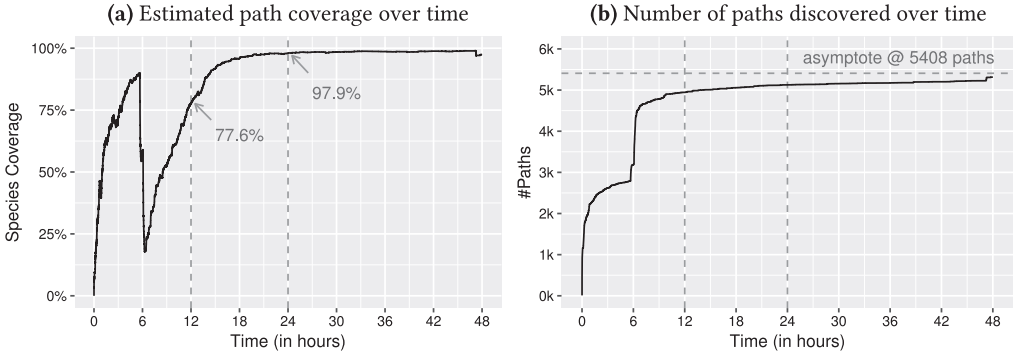


Fig. 7. Bias in the path coverage estimate for the AFL-fuzzing campaign in our motivating example.

The adaptive bias is obvious in Figure 7(a), which shows the development of the path coverage estimate over the first 48 hours of the fuzzing campaign in our motivating example (Section 2). Between 5 and 7 hours, we see a steep drop in the path coverage estimate. The reason becomes obvious in Figure 7(b), which shows the number of paths discovered for the same fuzzing campaign. Just before the 6-hour mark, the number of discovered paths seems to approach a different asymptote at about 3k paths when suddenly many more paths are discovered. This sudden increase is not very uncommon for AFL, particularly in the first few hours when still many new paths are discovered. However, such surges get more uncommon and their magnitude smaller as the sample coverage increases. This can be explained within the Markov chain model of directed graybox fuzzing [10]. The path coverage estimate quickly recovers over the next 12 hours. In Figure 7(b), we can see that 24 hours into the fuzzing campaign, a large percentage of paths has been discovered (w.r.t. the improved estimate of the asymptote). At this time, our path coverage estimate quite accurately puts the coverage at about 98%. In the future, we plan to investigate the correlation between the discovery probability estimate $\hat{U}$ of the sample and the bias/precision of the species coverage estimate $S(n)/\hat{S}$.

In our preliminary empirical study, we used the coverage-based graybox fuzzer AFL [100] to investigate the performance of the proposed estimators and extrapolators for *the* state-of-the-art vulnerability detection tool. The results are promising (see Section 6.5). For AFL, the magnitude of the estimator bias was substantial right before and during short intervals when the number of discovered path species increased suddenly and significantly. The extrapolation would not anticipate such sudden surges. However, the bias from adaptive sampling reduced over time. Close to the asymptotic species richness, the impact appeared negligible.

In future work, several correction methodologies for the adaptive sampling bias may be developed.In software testing, we can analyze the program, e.g., to quantify the likelihood of sudden increases in species coverage. Advanced program analyses may allow for *static bias correction strategies*. For coverage-based graybox fuzzers, as species discovery decelerates, the impact of the adaptive bias reduces as well. Estimates of sample completeness C or species richness S may be used as predictors of the adaptive bias that allow for *dynamic bias correction strategies*. Moreover, we anticipate *empirical studies of estimator performance* for other graybox fuzzers.

9.5.3 Symbolic Execution. Systematic (often symbolic execution-based) whitebox fuzzers are designed to discover previously undiscovered species with *every* test input that is generated. Hence, the STADS framework explicitly *does not apply* as such fuzzers violate the underlying assumptions of our statistical framework. For instance, a symbolic execution-based whitebox fuzzer

is designed to systematically enumerate every (interesting) path in the program [15]. Every generated test input exercises a different path. Once a species $\mathcal{D}_i$ has been discovered, the probability p_i of generating another input $t \in \mathcal{D}_i$ is $p_i = 0$. This is a substantial change from before the discovery.

However, we note that security researchers can use a blackbox or graybox fuzzer to establish the species evenness J for that program, and based on its value decide whether to choose that fuzzer or the symbolic execution-based whitebox fuzzer for the actual fuzzing. As symbolic execution-based fuzzers are better suited to discover rare species, there should be a certain minimal value of J below which the symbolic execution-based whitebox fuzzer performs better than the blackbox or graybox fuzzer. In the future, this value can be empirically investigated.

9.6 Adaptive Bias Correction

Unlike sampling strategies in ecology, the STADS framework allows continuous estimation and extrapolation during the fuzzing campaign itself. This provides opportunities for *continuous adaptive bias correction*. We can continuously assess the bias of our extrapolation by first predicting the value of our estimation target and later comparing it to its empirical value. The difference between predicted and empirical value describes the estimator bias. A continuous monitoring of the bias may allow one to gradually control for and correct the observed bias.

Monitoring the fuzzing campaign also enables *on-the-fly fuzzer selection*. In earlier work [8], we found that even the most effective systematic fuzzer would be less efficient than a random fuzzer if generating a test input takes relatively too long. For short fuzzing campaigns, a random fuzzer would always outperform a systematic fuzzer. However, at a certain time, it would be more efficient to switch to systematic fuzzing. Using the proposed extrapolators, we can make an informed decision when to switch, e.g,. from the "biased" random fuzzer AFL [100] to the systematic fuzzer, KLEE [15].

10 CONCLUSION

In this article, I introduced the foundations of a general, statistical framework that models software testing and analysis as discovery of species (STADS) to address a *fundamental challenge* in software testing: the statistically well-grounded *extrapolation* from program behaviors observed during testing. The STADS framework draws from over three decades of research in ecological biostatistics, where the challenge is to extrapolate from properties of the species observed in a sample to properties of the species in the complete assemblage.

Based on the STADS framework, researchers can, for the first time, formally discuss, estimate, and assess a *fuzzer*'s effectiveness and efficiency; a *campaign*'s completeness, cost-effectiveness, and residual risk; and a *program*'s fuzzability. For the first time, test engineers have gained the ability to make informed decisions about whether to abort or continue a fuzzing campaign, and to quantify what has been learned about the program at any point throughout the fuzzing campaign. Beyond this initial work, I pointed to a large number of opportunities for researchers to improve and tailor the ecologic methodologies to the automated testing and analysis process.

The first empirical evidence was provided that the main hypothesis that is underpinning the STADS framework (and thus allows the usage of existing ecological methodologies in the context of automated software testing and analysis) actually holds. The *multinomial model*, where each input belongs to exactly one species, was integrated and successfully evaluated. The evaluated estimators from ecology showed good performance, e.g., during estimation and extrapolation of path coverage. Thereupon, the STADS framework was extended with the *Bernoulli product model*, where each input can belong to one or more species. We show that the estimators can be efficiently computed even for large programs, and are guaranteed to approach the true value as the fuzzing effort increases. An overview of the pertinent estimators for both models can be found in Table 7.

Table 7. A Summary of the Pertinent Estimators and Extrapolators for the STADS Model That Were Discussed and/or Evaluated in This Article

	Multinomial Model (One Input, One Species)	**Bernoulli Product Model (One Input, Multiple Species)**
Estimating and extrapolating progress based on the total number of species $\hat{S}$ [34]		
Total #species	$\hat{S} \approx \begin{cases} S(n) + f_1^2/(2f_2) & \text{if } f_2 > 0 \\ S(n) + f_1(f_1-1)/2 & \text{if } f_2 = 0 \end{cases}$	$\hat{S} \approx \begin{cases} S(n) + Q_1^2/(2Q_2) & \text{if } Q_2 > 0 \\ S(n) + Q_1(Q_1-1)/2 & \text{if } Q_2 = 0 \end{cases}$
(*Chao1/2 estimators*)	$\hat{S} = S$ if S is known $\hat{S} = S(n) + \hat{f}_0 \rightarrow \hat{f}_0 = \hat{S} - S(n)$	$\hat{S} = S$ if S is known $\hat{S} = S(n) + \hat{Q}_0 \rightarrow \hat{Q}_0 = \hat{S} - S(n)$
How many more species are discovered with more inputs?	$\hat{S}(n+m^*) = S(n) + \hat{f}_0 \left[1 - \left(1 - \frac{f_1}{n\hat{f}_0 + f_1}\right)^{m^*}\right]$	$\hat{S}(n+m^*) = S(n) + \hat{Q}_0 \left[1 - \left(1 - \frac{Q_1}{n\hat{Q}_0 + Q_1}\right)^{m^*}\right]$
How many more inputs are needed to discover $G^* \cdot \hat{S}$ species where $G(n) < G^* < 1$?	$m_{G^*} \approx \frac{nf_1}{2f_2} \log\left[\frac{\hat{f}_0}{(1-G^*)\hat{S}}\right]$	$m_{G^*} \approx \frac{\log\left[1 - \frac{n}{(n-1)}\frac{2Q_2}{Q_1^2}(G^*\hat{S} - S(n))\right]}{\log\left[1 - \frac{2Q_2}{(n-1)Q_1 + 2Q_2}\right]}$
Estimating and extrapolating progress based on discovery probability $\hat{C}$ [24,49]		
Discovery probability	$\hat{U}(n) = \frac{f_1}{n}$	$\hat{U}(n) = \frac{Q_1}{V}\left[\frac{n\hat{Q}_0}{n\hat{Q}_0 + Q_1}\right] \approx \frac{Q_1}{V}$
How much more sample coverage is achieved with more inputs?	$\hat{U}(n+m^*) = \frac{f_1}{n}\left(\frac{n\hat{f}_0}{n\hat{f}_0 + f_1}\right)^{m^*+1}$	$\hat{U}(n+m^*) = \frac{Q_1}{V}\left[\frac{n\hat{Q}_0}{n\hat{Q}_0 + Q_1}\right]^{m^*+1}$

Notation: *In the current fuzzing campaign n is the number of generated test inputs, $S(n)$ is the number of discovered species, f_1 and Q_1 are the number of singleton species (i.e., those to which only one generated input belongs), f_2 and Q_2 are the number of doubleton species (i.e., those to which only two generated inputs belong), $\hat{f}_0$ and $\hat{Q}_0$ are estimates of the number of undiscovered species, and V denotes the sum total of the number of species that each generated test input belongs to. Note that $V > n$ if multiple species can be identified for a single input.*

The presented predictive program analysis scales to programs of arbitrary size and is general enough to work with arbitrary finite and discrete program properties. For instance, the STADS framework allows one to estimate the asymptotic total number of paths, information flows, reachable target locations, unique program crashes, or number of statements that are actually executable by the fuzzer. It allows one to extrapolate code coverage as well as mutation adequacy efficiently and with improving accuracy. Species coverage can be used effectively to judge whether the fuzzing campaign is almost completed.

Many estimators and extrapolators are readily available as statistical analyses to try out online [109, 120] or as packages in the R programming language [26, 62]. Our integration Pythia with the popular vulnerability detection tool AFL can be downloaded from Github at

https://github.com/mboehme/pythia.

The STADS framework provides a large number of opportunities for future work. For instance, software can be understood as megadiverse assemblage, which features a large number of very rare species. Novel estimators can be identified or developed that address the peculiarities of automated software testing as species discovery. Feedback-directed fuzzers introduce an adaptive bias that can result in sudden surges in species discovered. Adaptive bias correction strategies can be developed that leverage program analysis to anticipate and account for such surges to correct the adaptive bias dynamically.

ACKNOWLEDGMENTS

I would like to thank Anne Chao from the Institute of Statistics at National Tsing Hua University for her interesting comments about our model of software testing and analysis as discovery of species (i.e., the STADS model) and her suggestion to view testing objectives where each input can be assigned to multiple species as producing incidence rather than abundance data. I would also like to thank David Clark from the University College London and the attendants of the 41st CREST Open Workshop on "Software Engineering And Computer Science Using Information" for the interesting discussions about the role of entropy in automated software testing. In this article, the Shannon entropy quantifies a program's difficulty to being automatically tested by a fuzzer. Finally, I am grateful for the permission to publish a picture taken from an exhibit at the Lee Kong Chian Natural History Museum in Singapore (Figure 1).

REFERENCES

[1] Gail E. Austen, Markus Bindemann, Richard A. Griffiths, and David L. Roberts. 2016. Species identification by experts and non-experts: Comparing images from field guides. *Nature - Scientific Reports* 6 (2016), 1–7.

[2] Greg Banks, Marco Cova, Viktoria Felmetsger, Kevin Almeroth, Richard Kemmerer, and Giovanni Vigna. 2006. SNOOZE: Toward a stateful network protocol fuzZEr. In *Proceedings of the 9th International Conference on Information Security (ISC'06)*. 343–358.

[3] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. 2015. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering* 41, 5 (May 2015), 507–525.

[4] Yves Basset, Lukas Cizek, Philippe Cuénoud, Raphael K. Didham, François Guilhaumon, Olivier Missa, Vojtech Novotny, Frode Ødegaard, Tomas Roslin, Jürgen Schmidl, Alexey K. Tishechkin, Neville N. Winchester, David W. Roubik, Henri-Pierre Aberlenc, Johannes Bail, Héctor Barrios, Jon R. Bridle, Gabriela Castaño-Meneses, Bruno Corbara, Gianfranco Curletti, Wesley Duarte da Rocha, Domir De Bakker, Jacques H. C. Delabie, Alain Dejean, Laura L. Fagan, Andreas Floren, Roger L.Kitching, Enrique Medianero, Scott E. Miller, Evandro Gama de Oliveira, Jérôme Orivel, Marc Pollet, Mathieu Rapp, Sérvio P. Ribeiro, Yves Roisin, Jesper B. Schmidt, Line Sørensen, and Maurice Leponce. 2012. Arthropod diversity in a tropical forest. *Science* 338, 6113 (2012), 1481–1484.

[5] A. Bertolino. 2007. Software testing research: Achievements, challenges, dreams. In *Future of Software Engineering (FOSE'07)*. 85–103.

[6] Marcel Böhme, Bruno C. d. S. Oliveira, and Abhik Roychoudhury. 2013. Partition-based regression verification. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE'13)*. 302–311.

[7] Marcel Böhme, Bruno C. d. S. Oliveira, and Abhik Roychoudhury. 2013. Regression tests to expose change interaction errors. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'13)*. 334–344.
[8] Marcel Böhme and Soumya Paul. 2016. A probabilistic analysis of the efficiency of automated software testing. *IEEE Transactions on Software Engineering* 42, 4 (April 2016), 345–360.
[9] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. 2017. Directed greybox fuzzing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*. 2329–2344.
[10] Marcel Böhme, Van-Thuan Pham, and Abhik Roychoudhury. 2016. Coverage-based greybox fuzzing as Markov chain. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*. 1032–1043.
[11] Zdravko I. Botev and Dirk P. Kroese. 2008. An efficient algorithm for rare-event probability estimation, combinatorial optimization, and counting. *Methodology and Computing in Applied Probability* 10, 4 (2008), 471–505.
[12] Ulrich Brose, Neo D. Martinez, and Richard J. Williams. 2003. Estimating species richness: Sensitivity to sample coverage and insensitivity to spatial patterns. *Ecology* 84, 9 (2003), 2364–2377.
[13] J. Bunge and M. Fitzpatrick. 1993. Estimating the number of species: A review. *Journal of the American Statistical Association* 88, 421 (1993), 364–373.
[14] 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.
[15] Cristian Cadar, Daniel Dunbar, and Dawson Engler. 2008. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI'08)*. 209–224.
[16] Anne Chao. 1984. Nonparametric estimation of the number of classes in a population. *Scandinavian Journal of Statistics* 11, 4 (1984), 265–270.
[17] Anne Chao. 1987. Estimating the population size for capture-recapture data with unequal catchability. *Biometrics* 43, 4 (1987), 783–791.
[18] Anne Chao and Chun-Huo Chiu. 2014. *Species Richness: Estimation and Comparison*. John Wiley & Sons.
[19] Anne Chao and Chun-Huo Chiu. 2016. Nonparametric estimation and comparison of species richness. In *Encyclopedia of Life Sciences (eLS)*.
[20] Anne Chao, Chun-Huo Chiu, Robert K. Colwell, Luiz Fernando S. Magnago, Robin L. Chazdon, and Nicholas J. Gotelli. 2017. Deciphering the enigma of undetected species, phylogenetic, and functional diversity based on good-turing theory. *Ecology* 98, 11 (2017), 2914–2929.
[21] Anne Chao and Robert K. Colwell. 2017. Thirty years of progeny from Chao's inequality: Estimating and comparing richness with incidence data and incomplete sampling. *Statistics and Operations Research Transactions* 41, 1 (2017), 3–54.
[22] Anne Chao, Robert K. Colwell, Chih-Wei Lin, and Nicholas J. Gotelli. 2009. Sufficient sampling for asymptotic minimum species richness estimators. *Ecology* 90, 4 (2009), 1125–1133.
[23] Anne Chao, T. C. Hsieh, Robin L. Chazdon, Robert K. Colwell, and Nicholas J. Gotelli. 2015. Unveiling the species-rank abundance distribution by generalizing the good-turing sample coverage theory. *Ecology* 96, 5 (2015), 1189–1201.
[24] Anne Chao and Lou Jost. 2012. Coverage-based rarefaction and extrapolation: Standardizing samples by completeness rather than size. *Ecology* 93, 12 (2012), 2533–2547.
[25] Anne Chao and Shen-Ming Lee. 1992. Estimating the number of classes via sample coverage. *Journal of the American Statistical Association* 87, 417 (1992), 210–217.
[26] A. Chao, K. H. Ma, T. C. Hsieh, and C. H. Chiu. 2015. Online Program SpadeR (Species-richness Prediction And Diversity Estimation in R). Program and User's Guide. Retrieved from http://chao.stat.nthu.edu.tw/wordpress/software_download.
[27] Anne Chao and Tsung-Jen Shen. 2004. Nonparametric prediction in species sampling. *Journal of Agricultural, Biological, and Environmental Statistics* 9, 3 (2004), 253–269.
[28] R. L. Chazdon, R. K. Colwell, J. S. Denslow, and M. R. Guariguata. 1998. Statistical methods for estimating species richness of woody regeneration in primary and secondary rain forests of Northeastern Costa Rica. In *Forest Biodiversity Research, Monitoring and Modeling: Conceptual Background and Old World Case Studies*. Vol. 20. Man and the Biosphere Series. 285–309.
[29] Thierry Titcheu Chekam, Mike Papadakis, Yves Le Traon, and Mark Harman. 2017. An empirical study on mutation, statement and branch coverage fault revelation that avoids the unreliable clean program assumption. In *Proceedings of the 39th International Conference on Software Engineering (ICSE'17)*. 597–608.
[30] Tsong Yueh Chen and Yuen-Tak Yu. 1996. On the expected number of failures detected by subdomain testing and random testing. *IEEE Transactions on Software Engineering* 22, 2 (1996), 109–119.

[31] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea. 2011. S2E: A platform for in-vivo multi-path analysis of software systems. In *Proceedings of the 2011 ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11)*. 265–278.

[32] C. H. Chiu, Y. T. Wang, B. A. Walther, and A. Chao. 2014. An improved nonparametric lower bound of species richness via a modified Good-Turing frequency formula. *Biometrics* 70, 3 (2014), 671–682.

[33] Bernard D. Coleman. 1981. On random placement and species-area relations. *Mathematical Biosciences* 54, 3 (1981), 191–215.

[34] Robert K. Colwell, Anne Chao, Nicholas J. Gotelli, Shang-Yi Lin, Chang Xuan Mao, Robin L. Chazdon, and John T. Longino. 2012. Models and estimators linking individual-based and sample-based rarefaction, extrapolation and comparison of assemblages. *Journal of Plant Ecology* 5, 1 (2012), 3.

[35] Robert K. Colwell and Jonathan A. Coddington. 1994. Estimating terrestrial biodiversity through extrapolation. *Philosophical Transactions of the Royal Society of London B: Biological Sciences* 345, 1311 (1994), 101–118.

[36] Robert K. Colwell and Johanna E. Elsensohn. 2014. EstimateS turns 20: Statistical estimation of species richness and shared species from samples, with non-parametric extrapolation. *Ecography* 37, 6 (2014), 609–613.

[37] Robert K. Colwell, Chang Xuan Mao, and Jing Chang. 2004. Interpolating, extrapolating, and comparing incidence-based species accumulation curves. *Ecology* 85, 10 (2004), 2717–2727.

[38] Edsger W. Dijkstra. 1970. Notes on Structured Programming.

[39] Joe W. Duran and Simeon C. Ntafos. 1984. An evaluation of random testing. *IEEE Transactions on Software Engineering* 10, 4 (July 1984), 438–444.

[40] S. M. Durant, T. Wacher, S. Bashir, R. Woodroffe, P. De Ornellas, C. Ransom, J. Newby, T. Abáigar, M. Abdelgadir, H. El Alqamy, J. Baillie, M. Beddiaf, F. Belbachir, A. Belbachir-Bazi, A. A. Berbash, N. E. Bemadjim, R. Beudels-Jamar, L. Boitani, C. Breitenmoser, M. Cano, P. Chardonnet, B. Collen, W. A. Cornforth, F. Cuzin, P. Gerngross, B. Haddane, M. Hadjeloum, A. Jacobson, A. Jebali, F. Lamarque, D. Mallon, K. Minkowski, S. Monfort, B. Ndoassal, B. Niagate, G. Purchase, S. Samala, A. K. Samna, C. Sillero-Zubiri, A. E. Soultan, M. R. Stanley Price, and N. Pettorelli. 2014. Fiddling in biodiversity hotspots while deserts burn? Collapse of the Sahara's megafauna. *Diversity and Distributions* 20, 1 (2014), 114–122.

[41] Antonio Filieri, Corina S. Păsăreanu, and Willem Visser. 2013. Reliability analysis in symbolic pathfinder. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE'13)*. 622–631.

[42] R. A. Fisher, A. S. Corbet, and C. B. Williams. 1943. The relation between the number of species and the number of individuals in a random sample of an animal population. *Journal of Animal Ecology* 12 (1943), 42–58.

[43] William A. Gale and Geoffrey Sampson. 1995. Good-Turing smoothing without tears. *Journal of Quantitative Linguistics* 2 (1995), 217–237.

[44] G. Gay, M. Staats, M. Whalen, and M. P. E. Heimdahl. 2015. The risks of coverage-directed test case generation. *IEEE Transactions on Software Engineering* 41, 8 (Aug. 2015), 803–819.

[45] Jaco Geldenhuys, Matthew B. Dwyer, and Willem Visser. 2012. Probabilistic symbolic execution. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis (ISSTA'12)*. 166–176.

[46] Patrice Godefroid, Adam Kiezun, and Michael Y. Levin. 2008. Grammar-based whitebox fuzzing. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'08)*. 206–215.

[47] Patrice Godefroid, Nils Klarlund, and Koushik Sen. 2005. DART: Directed automated random testing. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'05)*. 213–223.

[48] I. J. Good. 2000. Turing's anticipation of empirical Bayes in connection with the cryptanalysis of the naval enigma. *Journal of Statistical Computation and Simulation* 66, 2 (2000), 101–111.

[49] Irving John Good. 1953. The population frequencies of species and the estimation of population parameters. *Biometrika* 40 (1953), 237–264.

[50] I. J. Good and G. H. Toulmin. 1956. The number of new species, and the increase in population coverage, when a sample is increased. *Biometrika* 43, 1/2 (1956), 45–63.

[51] Steven M. Goodman and Jonathan P. Benstead. 2005. Updated estimates of biotic diversity and endemism for Madagascar. *Oryx* 39, 1 (2005), 73–77.

[52] Nicholas J. Gotelli and Anne Chao. 2013. Measuring and estimating species richness, species diversity, and biotic similarity from sampling data. In *Encyclopedia of Biodiversity* (2nd ed.). Vol. 5. Academic Press.195–211.

[53] Walter J. Gutjahr. 1999. Partition testing vs. random testing: The influence of uncertainty. *IEEE Transactions on Software Engineering* 25, 5 (Sept. 1999), 661–674.

[54] Andrew J. Hamilton, Yves Basset, Kurt K. Benke, Peter S. Grimbacher, Scott E. Miller, Vojtech Novotnỳ, G. Allan Samuelson, Nigel E. Stork, George D. Weiblen, and Jian D.L. Yen. 2010. Quantifying uncertainty in estimation of tropical arthropod species richness. *American Naturalist* 176, 1 (2010), 90–95.

[55] Andrew J. Hamilton, Yves Basset, Kurt K. Benke, Peter S. Grimbacher, Scott E. Miller, Vojtech Novotnỳ, G. Allan Samuelson, Nigel E. Stork, George D. Weiblen, and Jian D. L. Yen. 2011. Correction. *American Naturalist* 177, 4 (2011), 544–545.

[56] D. Hamlet and R. Taylor. 1990. Partition testing does not inspire confidence [program testing]. *IEEE Transactions on Software Engineering* 16, 12 (Dec. 1990), 1402–1411.
[57] Dick Hamlet and Jeff Voas. 1993. Faults on its sleeve: Amplifying software reliability testing. In *Proceedings of the 1993 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'93)*. 89–98.
[58] Mary Jean Harrold. 2000. Testing: A roadmap. In *Proceedings of the Conference on the Future of Software Engineering (ICSE'00)*. 61–72.
[59] Frank J. Hernandez and David G. Lindquist. 1999. A comparison of two light-trap designs for sampling larval and presettlement juvenile fish above a reef in Onslow Bay, North Carolina. *Bulletin of Marine Science* 64, 1 (1999), 173–184.
[60] Joaquin Hortal, Paulo A. V. 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.
[61] Matthias Höschele and Andreas Zeller. 2016. Mining input grammars from dynamic taints. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE'16)*. 720–725.
[62] T. C. Hsieh, K. H. Ma, and Anne Chao. 2016. iNEXT: An R package for rarefaction and extrapolation of species diversity (Hill numbers). *Methods in Ecology and Evolution* 7, 12 (2016), 1451–1456. DOI: http://dx.doi.org/10.1111/2041-210X.12613
[63] Stuart H. Hurlbert. 1971. The nonconcept of species diversity: A critique and alternative parameters. *Ecology* 52, 4 (1971), 577–586.
[64] Laura Inozemtseva and Reid Holmes. 2014. Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*. 435–445.
[65] E. T. Jaynes. 2003. *Probability Theory: The Logic of Science*. Cambridge University Press.
[66] Y. Jia and M. Harman. 2011. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering* 37, 5 (Sept. 2011), 649–678.
[67] Shen-Ming Lee and Anne Chao. 1994. Estimating population size via sample coverage for closed capture-recapture models. *Biometrics* 50, 1 (1994), 88–97.
[68] B. Littlewood and D. Wright. 1997. Some conservative stopping rules for the operational testing of safety critical software. *IEEE Transactions on Software Engineering* 23, 11 (Nov. 1997), 673–683.
[69] John T. Longino and Robert K. Colwell. 1997. Biodiversity assessment using structured inventory: Capturing the ant fauna of a tropical rain forest. *Ecological Applications* 7, 4 (1997), 1263–1277.
[70] Anne E. Magurran and Brian J. McGill. 2011. *Biological Diversity: Frontiers in Measurement and Assessment*. Oxford University Press.
[71] Chang Xuan Mao and Robert K. Colwell. 2005. Estimation of species richness: Mixture models, the role of rare species, and inferential challenges. *Ecology* 86, 5 (2005), 1143–1153.
[72] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: Multi-objective automated testing for android applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA'16)*. 94–105.
[73] Björn Matthis, Vitalii Avdiienko, Ezekiel Soremekun, Marcel Böhme, and Andreas Zeller. 2017. Detecting information flow by mutating input data. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE'17)*. 1–11.
[74] Brian A. Maurer, James H. Brown, and Renee D. Rusler. 1992. The micro and macro in body size evolution. *Evolution* 46, 4 (1992), 939–953.
[75] Phil McMinn. 2004. Search-based software test data generation: A survey: Research articles. *Journal of Software Testing, Verification and Reliability* 14, 2 (June 2004), 105–156.
[76] P. McMinn. 2011. Search-based software testing: Past, present and future. In *Proceedings of the 4th IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW'11)*. 153–163.
[77] Barton P. Miller, Louis Fredriksen, and Bryan So. 1990. An empirical study of the reliability of UNIX utilities. *Communications of the ACM* 33, 12 (Dec. 1990), 32–44.
[78] K. W. Miller, L. J. Morell, R. E. Noonan, S. K. Park, D. M. Nicol, B. W. Murrill, and M. Voas. 1992. Estimating the probability of failure when testing reveals no failures. *IEEE Transactions on Software Engineering* 18, 1 (Jan. 1992), 33–43.
[79] Camilo Mora, Derek P. Tittensor, Sina Adl, Alastair G. B. Simpson, and Boris Worm. 2011. How many species are there on earth and in the ocean? *PLOS Biology* 9, 8 (2011), 1–8.
[80] Mesrob I. Ohannessian. 2012. *On Inference about Rare Events*. PhD dissertation. Massachusetts Institute of Technology.
[81] Alon Orlitsky and Ananda Theertha Suresh. 2015. Competitive distribution estimation: Why is Good-Turing good. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS'15)*. 2143–2151.
[82] Alon Orlitsky, Ananda Theertha Suresh, and Yihong Wu. 2016. Optimal prediction of the number of unseen species. *Proceedings of the National Academy of Sciences* 113, 47 (2016), 13283–13288.

[83] Michael W. Palmer. 1991. Estimating species richness: The second-order jackknife reconsidered. *Ecology* 72, 4 (1991), 1512–1513.
[84] Van-Thuan Pham, Marcel Böhme, and Abhik Roychoudhury. 2016. Model-based whitebox fuzzing for program binaries. In *Proceedings of the 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE'16)*. 543–553.
[85] E. C. Pielou. 1966. Species-diversity and pattern-diversity in the study of ecological succession. *Journal of Theoretical Biology* 10, 2 (1966), 370–383.
[86] F. W. Preston. 1948. The commonness, and rarity, of species. *Ecology* 29, 3 (1948), 254–283.
[87] Dawei Qi, Hoang D. T. Nguyen, and Abhik Roychoudhury. 2013. Path exploration based on symbolic output. *ACM Transactions on Software Engineering and Methodology* 22, 4 (Oct. 2013), 32:1–32:41.
[88] Sushma Reddy and Liliana M. Dávalos. 2003. Geographical sampling bias and its implications for conservation priorities in Africa. *Journal of Biogeography* 30, 11 (2003), 1719–1727.
[89] Herbert E. Robbins. 1968. Estimating the total probability of the unobserved outcomes of an experiment. *Annals of Mathematical Statistics* 39, 1 (1968), 256–257.
[90] Mark D. Robinson, Davis J. McCarthy, and Gordon K. Smyth. 2010. edgeR: A bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics* 26, 1 (2010), 139–140.
[91] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitry Vyukov. 2012. AddressSanitizer: A fast address sanity checker. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference (USENIX ATC'12)*. 28–28.
[92] Tsung-Jen Shen, Anne Chao, and Chih-Feng Lin. 2003. Predicting the number of new species in further taxonomic sampling. *Ecology* 84, 3 (2003), 798–804.
[93] Andrew R. Solow and Stephen Polasky. 1999. A quick estimator for taxonomic surveys. *Ecology* 80, 8 (1999), 2799–2803.
[94] Evgeniy Stepanov and Konstantin Serebryany. 2015. MemorySanitizer: Fast detector of uninitialized memory use in C++. In *Proceedings of the 2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO'15)*. 46–55.
[95] A. B. Wagner, P. Viswanath, and S. R. Kulkarni. 2006. Strong consistency of the good-turing estimator. In *Proceedings of the 2006 IEEE International Symposium on Information Theory*. 2526–2530.
[96] Bruno A. Walther and Joslin L. Moore. 2005. The concepts of bias, precision and accuracy, and their use in testing the performance of species richness estimators, with a literature review of estimator performance. *Ecography* 28, 6 (2005), 815–829.
[97] Bruno A. Walther and Joslin L. Moore. 2005. The concepts of bias, precision and accuracy, and their use in testing the performance of species richness estimators, with a literature review of estimator performance. *Ecography* 28, 6 (2005), 815–829.
[98] Ponemon Institute. 2017. Ponemon Cost of Cyber Crime Study. Retrieved from https://www.accenture.com/us-en/insight-cost-of-cybercrime-2017. Accessed 11-13-2017.
[99] IEEE Computer Society. 2013. Lockheed Martin Webinar Series: Michael Whalen on the Future of Verification and Validation. *R*etrieved from https://www.computer.org/cms/Computer.org/webinars/lmco/012413Slides-Whalen.pdf. Accessed: 05-13-2017.
[100] Michał Zalewski. 2017. AFL: American Fuzzy Lop Fuzzer. Retrieved from http://lcamtuf.coredump.cx/afl/technical_details.txt. Accessed: 05-13-2017.
[101] Michał Zalewski. 2017. AFL: Pulling Jpegs Out of Thin Air, Michael Zalewski. Retrieved from https://lcamtuf.blogspot.com/2014/11/pulling-jpegs-out-of-thin-air.html. Accessed: 05-13-2017.
[102] CERT Division. 2017. CERT Triage Tools. Retrieved from https://www.cert.org/vulnerability-analysis/tools/triage.cfm?. Accessed: 05-13-2017.
[103] The LLVM Compiler Infrastructure Project. 2017. Clang Compiler Documentation. Retrieved from https://clang.llvm.org/docs/index.html. Accessed: 05-13-2017.
[104] US Defense Advanced Research Projects Agency. 2017. DARPA Cyber Grand Challenge. Retrieved from http://www.darpa.mil/news-events/2016-08-04. Accessed: 05-13-2017.
[105] Facebook. 2017. Facebook: Mark Harman on Software Engineering at Facebook Scale. Retrieved from https://research.fb.com/mark-harmon-on-software-engineering-at-facebook-scale/. Accessed: 05-13-2017.
[106] FFMPEG. 2017. FFMPEG: A Complete, Cross-Platform Solution to Record, Convert and Stream Audio and Video. Retrieved from https://www.ffmpeg.org/. Accessed: 05-13-2017.
[107] GNU. 2017. GCov: Coverage Testing Tool. Retrieved from https://linux.die.net/man/1/gcov. Accessed: 11-13-2017.
[108] GNU. 2017. GNU GCC Sanitizer Options. Retrieved from https://gcc.gnu.org/onlinedocs/gcc-6.3.0/gcc/Instrumentation-Options.html#index-fsanitize_003daddress-947. Accessed: 05-13-2017.

[109] Anne Chao, K. H. Ma, and T. C. Hsieh. 2017. iNext Online: Species iNterpolation and EXTrapolation. Retrieved from https://chao.shinyapps.io/iNEXTOnline/. Accessed: 05-13-2017.
[110] Niels Lohmann. 2017. JSON for Modern C++. Retrieved from https://github.com/nlohmann/json. Accessed: 11-13-2017.
[111] The LLVM Compiler Infrastructure Project. 2017. LibFuzzer: A Library for Coverage-Guided Fuzz Testing. Retrieved from http://llvm.org/docs/LibFuzzer.html. Accessed: 05-13-2017.
[112] The Libjpeg-turbo Project. 2017. libjpeg-turbo Is a JPEG Image Codec to Accelerate Baseline JPEG Compression and Decompression. Retrieved from http://libjpeg-turbo.virtualgl.org/. Accessed: 05-13-2017.
[113] The GNOME Project. 2017. LibXML2: The XML C Parser and Toolkit of Gnome. Retrieved from http://xmlsoft.org/. Accessed: 11-13-2017.
[114] Microsoft. 2017. Microsoft: Project Springfield. Retrieved from https://www.microsoft.com/Springfield/. Accessed: 05-13-2017.
[115] Google. 2017. Monkey: Android Random Testing. Retrieved from http://developer.android.com/tools/help/monkey.html. Accessed: 05-13-2017.
[116] Mozilla. 2017. Mozilla: Fuzzing Firefox with Peach.*R*etrieved from https://wiki.mozilla.org/Security/Fuzzing/Peach. Accessed: 05-13-2017.
[117] The OpenSSL Project. 2017. OpenSSL: A Toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) Protocols. Retrieved from https://www.openssl.org/. Accessed: 05-13-2017.
[118] Google. 2017. OSS-Fuzz: Five Months Later. Retrieved from https://testing.googleblog.com/2017/05/oss-fuzz-five-months-later-and.html. Accessed: 05-13-2017.
[119] Peach Fuzzer. 2017. Peach Fuzzer Platform. Retieved from http://www.peachfuzzer.com/products/peach-platform/. Accessed: 05-13-2017.
[120] Anne Chao, K. H. Ma, T. C. Hsieh, and Chun-Huo Chiu. 2017. SpadeR Online: Species-Richness Prediction and Diversity Estimation in R. Retrieved from https://chao.shinyapps.io/SpadeR/. Accessed: 05-13-2017.
[121] Eric Archer. 2017. sprex: Calculate Species Richness and Extrapolation Metrics. Retrieved from https://cran.r-project.org/web/packages/sprex/. Accessed: 05-13-2017.
[122] Google. 2017. Syzkaller: Coverage-Guided Kernel Fuzzing. Retrieved from https://github.com/google/syzkaller. Accessed: 05-13-2017.
[123] The Wireshark team. 2017. Wireshark Is the World's Foremost and Widely-Used Network Protocol Analyzer. Retrieved from https://www.wireshark.org/. Accessed: 05-13-2017.
[124] Sean Eron Anderson. 2017. Bithacks: Implementing Logarithm Efficiently. Retrieved from https://graphics.stanford.edu/ seander/bithacks.html#IntegerLogFloat. Accessed: 11-13-2017.
[125] Cloudflare. 2017. Incident Report on Memory Leak Caused by Cloudflare Parser Bug. Retrieved from https://blog.cloudflare.com/incident-report-on-memory-leak-caused-by-cloudflare-parser-bug/. Accessed: 11-13-2017.
[126] Haseeb Qureshi. 2017. Medium: A Hacker Stole USD 31M of Ether. Retrieved from https://medium.freecodecamp.org/a-hacker-stole-31m-of-ether-how-it-happened-and-what-it-means-for-ethereum-9e5dc29e33ce. Accessed: 05-13-2017.
[127] Damien Gayle, Alexandra Topping, Ian Sample, Sarah Marsh, and Vikram Dodd. 2017. NHS Seeks to Recover from Global Cyber-Attack as Security Concerns Resurface. Retrieved from https://www.theguardian.com/society/2017/may/12/hospitals-across-england-hit-by-large-scale-cyber-attack. Accessed: 11-13-2017.
[128] Elaine J. Weyuker. 1982. On testing non-testable programs. *Computer Journal* 25, 4 (1982), 465–470.
[129] E. J. Weyuker and B. Jeng. 1991. Analyzing partition testing strategies. *IEEE Transactions on Software Engineering* 17, 7 (July 1991), 703–711.
[130] B. Yang and M. Xie. 2000. A study of operational and testing reliability in software reliability analysis. *Reliability Engineering & System Safety* 70, 3 (2000), 323–329.
[131] Xuejun Yang, Yang Chen, Eric Eide, and John Regehr. 2011. Finding and understanding bugs in C compilers. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'11)*. 283–294.
[132] Cun-Hui Zhang and Zhiyi Zhang. 2009. Asymptotic normality of a nonparametric estimator of sample coverage. *Annals of Statistics* 37, 5A (10 2009), 2582–2595.

Received August 2017; revised April 2018; accepted April 2018