On the Efficiency of Automated Testing

Marcel Böhme*
Saarland University, Germany
boehme@cs.uni-saarland.de

Soumya Paul
National University of Singapore
pauls@comp.nus.edu.sg

ABSTRACT

The aim of automated program testing is to gain confidence about a program's correctness by sampling its input space. The sampling process can be either systematic or random. For every *systematic testing technique* the sampling is informed by the *analysis* of some program artefacts, like the specification, the source code (e.g., to achieve coverage), or even faulty versions of the program (e.g., mutation testing). This analysis incurs some *cost*. In contrast, *random testing* is unsystematic and does not sustain any analysis cost.

In this paper, we investigate the theoretical efficiency of systematic versus random testing. First, we mathematically model the *most effective* systematic testing technique $\mathcal{S}_0$ in which every sampled test input *strictly increases* the "degree of confidence" and is subject to the analysis cost c. Note that the *efficiency* of $\mathcal{S}_0$ depends on c. Specifically, if we increase c, we also increase the time it takes $\mathcal{S}_0$ to establish the same degree of confidence. So, there exists a maximum analysis cost beyond which $\mathcal{R}$ is generally more efficient than $\mathcal{S}_0$.

Given that we require the confidence that the program works correctly for $x\%$ of its input, we prove an upper bound on c of $\mathcal{S}_0$, beyond which $\mathcal{R}$ is more efficient on the average. We also show that this bound depends asymptotically only on x. For instance, let $\mathcal{R}$ take $10ms$ time to sample one test input; to establish that the program works correctly for 90% of its input, $\mathcal{S}_0$ must take less than $41ms$ to sample one test input. Otherwise, $\mathcal{R}$ is expected to establish the 90%-degree of confidence earlier. We prove similar bounds on the cost if the software tester is interested in revealing as many errors as possible in a given time span.

Categories and Subject Descriptors: D.2.5 [Software Engineering] Testing and Debugging

General Terms: Theory

Keywords: Partition Testing, Random Testing, Error-based Partitioning, Efficient Testing, Testing Theory

*The first author conducted this work during his PhD at the National University of Singapore

FSE '14, November 16–22, 2014, Hong Kong, China

1. INTRODUCTION

We can never be sure! Complex software errors exist even in critical, widely distributed programs for many years [3, 4]. So, developers are looking for an *efficient* and automated technique to *gain confidence* in their programs' correctness.

Inspiring confidence is the main goal of *software testing*. By analyzing the program's specification, tools can automatically generate test inputs that cover corner-cases [5]. By analyzing the program's source code, tools can generate inputs that stress potentially faulty statements, branches, or paths by increasing the coverage of the code [10, 6, 12]. By generating and analyzing deliberately faulty versions [21], tools can generate even more effective test input. Generally, the *more comprehensive such analysis*, the *more effective* can the testing technique be. But, with increasing analysis time, what about the associated *reduction of efficiency*?

We model the testing problem as an *exploration of error-based input partitions*. Suppose, for a program there exists a partitioning of its input space into homogeneous subdomains [30, 28]. For each subdomain, either all inputs reveal an error or none of the inputs reveal an error. The number and "size" of such error-based partitions can be arbitrary. Assuming that it is unknown *a-priori*[1] whether or not a partition reveals an error, the problem of software testing is to sample each partition in a systematic fashion to gain confidence in the correctness of the program.

Weyuker and Jeng [30] observe that a testing technique that samples from error-based partitions is *most effective*. However, realistic techniques can only approximate the error-based partitions depending on the extent of the analysis [16]. For instance, 100% branch coverage requires that at least one input is sampled from each "branch-based" subdomain, where a subdomain may cover many error-based partitions. So, some error-based partitions may not be sampled at all.

We model the *most effective* systematic technique $\mathcal{S}_0$ that samples exactly one input from *each* error-based partition and investigate its efficiency depending on the analysis cost. Every sampled input becomes a witness of the error-revealing property of the sampled partition and *strictly increases* the established degree of confidence. For each sampling, we assign a constant analysis cost and observe: with an increased cost, it takes more time to establish the same degree of confidence and discover the same number of errors. In other words, *efficiency decreases when the analysis cost increases*. We ask: For which analysis cost does systematic testing $\mathcal{S}_0$ become less efficient than unsystematic random testing $\mathcal{R}$?

[1]If it was known whether or not a partition reveals an error, there would be no need for testing.

In this paper, we study the maximum analysis cost c for the systematic testing technique $\mathcal{S}_0$ to remain more efficient than random testing $\mathcal{R}$. Not sustaining any analysis cost, we say that $\mathcal{R}$ *takes one unit of time* to sample one test input. Thus, we can give the analysis cost c of $\mathcal{S}_0$ as a factor of the time it takes $\mathcal{R}$ to sample one test input. We say that $\mathcal{S}_0$ *takes* c *units of time* to sample one test input. Note that giving cost as a factor allows us to account for the time spent on the concrete sampling-related tasks that are common to both techniques, $\mathcal{S}_0$ and $\mathcal{R}$. For instance, if $\mathcal{R}$ takes, on average, $5ms$ to generate, execute, and check against an oracle the outcome of a test input, then by definition $\mathcal{S}_0$ takes $(\mathsf{c} \cdot 5)ms$ which includes the same time spent on test generation, execution, and oracle checking *and the time spent on analysis*. Now, with increasing analysis cost, $\mathcal{S}_0$ becomes less efficient while $\mathcal{R}$ remains just as efficient. So, in order for $\mathcal{S}_0$ to maintain its efficiency over $\mathcal{R}$, the analysis cost c cannot exceed a certain value and is thus bounded above!

We explore two notions of testing efficiency that may be considered as the main goals of automated software testing: i) to achieve a given degree of confidence in minimal time, and ii) to expose a maximal number of errors in a given time. Furthermore, we take the analysis cost c as a constant for all programs. However, the analysis cost is likely to depend on the program size – and if analysis cost is bounded above, then program size is as well. That is, for such systematic testing techniques there exists a *maximum program size beyond which $\mathcal{R}$ is generally more efficient*.

The following are the three most important contributions of the paper.

- **Analytical Framework**. We provide a mathematical system to assess the efficiency of any automated testing technique $\mathcal{S}$ relative to that of random testing $\mathcal{R}$. It accounts for the cost c of $\mathcal{S}$ depending on which there exists a unique point in time where $\mathcal{S}$ and $\mathcal{R}$ "break even" towards reaching the testing goal. So, the relative efficiency of $\mathcal{S}$ is always bounded above and for a concrete instance can be computed similarly as discussed for $\mathcal{S}_0$ in this paper – where $\mathcal{S}_0$ generates *one* test input for each error-based partition that is chosen uniformly at random.

- **Testing to Achieve Confidence**. Given a degree of confidence x, we show that the time it takes $\mathcal{S}_0$ to sample an input cannot exceed $(ex - ex^2)^{-1}$ times the time it takes for $\mathcal{R}$ to sample an input. Otherwise, $\mathcal{R}$ is more efficient than $\mathcal{S}_0$ on the average. For instance, let $\mathcal{R}$ take $10ms$ to sample one test input randomly; to establish the confidence that *any* program works correctly for 90% of its input, $\mathcal{S}_0$ *must take less than* $41ms$ to sample one test input systematically.

- **Testing to Discover Errors**. Given a time bound $\hat{n}$, we show that the time taken by $\mathcal{S}_0$ to sample an input cannot exceed $\frac{\hat{n}}{k} \cdot (1 - (1 - q_{\min})^{\hat{n}})^{-1}$ times the time taken by $\mathcal{R}$ to sample an input, in order for $\mathcal{S}_0$ to remain more efficient than $\mathcal{R}$ – where k is the number of partitions and $q_{\min}$ the fractional size of the "smallest" error-revealing partition in the program's input space.

These are fundamental insights that hold for all programs and every systematic testing technique under the realistic assumptions stated in the following section.

2. PRELIMINARIES

2.1 Background

In this work, we focus on automated testing techniques that seek to establish a certain degree of confidence in the correctness of the program or reveal a maximal number of errors. Interestingly, this eliminates inexhaustive, automated techniques that seek to generate just one failing test input as evidence of the incorrectness of the program. First, the search for a failing test input may never terminate due to the undecidability of the infeasible path problem [14]. Secondly, the absence of a failing test input throughout the search does not inspire any degree of confidence in the absence of errors. Instead, we shall focus on partition testing techniques, such as coverage, mutation, and specification based testing.

Partition testing [16, 30] comprises of testing techniques that 1) divide the program's input domain into classes whose points share the same property in some respect and then 2) test the program for at least one input from each class. Thus, the problem of systematic testing is reduced to finding a "good" partition strategy. For example, a *specficiation-based* partition strategy might divide the input domain into subdomains, each of which invokes one of several program features or satisfies the pre-condition of some predicate [5]. *Mutation-based* partition strategies may yield subdomains, each of which strongly kills a certain mutant of the program [17, 21]. A *differential* partition strategy yields subdomains, each of which either homogeneously exposes a semantic difference or homogeneously shows semantic equivalence [2]. Symbolic execution is a *path-based* partition strategy [12]. One may also consider strategies that partition the input space such that classes of input do and others do not violate an assertion in the program.

However, questioning its *effectiveness*, Hamlet and Taylor [16] find that "partition testing does not inspire confidence". Varying several parameters, the authors repeated the experiments of Duran and Ntafos [9] who presented a surprising result: The number of errors found by random and partition testing is very similar. Hamlet and Taylor came to much the same conclusion. The results universally favoured partition testing, but not by much. Weyuker and Jeng [30] found that the effectiveness of partition testing varies depending on the fault rate for each subdomain that is systematically sampled and concluded that a partitioning strategy that yields error-based (revealing) subdomains is the most effective. Subsequently, several authors discussed conditions under which partition testing is generally more effective than random testing (e.g., [15, 8]). Only recently, Arcuri et al. [1] pointed out that "random testing is more effective and predictable than commonly thought" and that "analytical and empirical analyses have not shown so far a clear inferiority of random testing compared with other more sophisticated techniques". The authors provide a non-trivial, optimal lower bound on the number of test inputs that need to be generated to cover a given set of targets.

Arcuri et al. [1] study the *scalability* of random testing. In this work, scalability refers to the ability of exercising many "targets" in the program as the number of targets increases. Specifically, the authors show that random testing scales better than a directed testing technique that focuses on one target until it is "covered" before proceeding to the next. Intuitively, parallel search (here, random testing) scales better than sequential search (here, directed systematic testing).

We are the first to introduce a theory of *testing efficiency* assuming the goal is 1) to achieve a certain degree of confidence in minimal time or 2) to expose a maximal number of errors in a certain time. Thereby, we assume *error-based partitioning* and model a systematic testing technique $\mathcal{S}_0$ that samples exactly one test input from each error-based partition. Hence, $\mathcal{S}_0$ is among the *most effective* [30] and (disregarding the analysis cost) one of the *most efficient* testing techniques. Note that realistic techniques with a similar partition sampling scheme are both, *less effective* and *less efficient* since some error-based partitions are sampled several times and others not at all due to the approximation.

Leaving the scope of our analysis are several practical concerns that are common to all automated testing techniques. i) Firstly, there is the *oracle problem* [29] which states that a mechanism deciding for every input whether the program computes the correct output is pragmatically unattainable and only approximate. Partial solutions include the automated encoding of common [18, 7, 27] and the manual encoding of custom error conditions as assertions [23, 19, 13]. ii) Secondly, there is the *typicality problem* which states that automatically generated test cases may not represent the "typical" input a user would provide or "valid" input that satisfies some pre-condition for the program to execute normally. Technically, both techniques could sample according to the operational distribution [26] or using symbolic grammars [20]. Then, both techniques receive the same ability to sample typical, valid inputs. We make no such assumptions. iii) Finally, we want to stress explicitly that for the purpose of this paper the achieved *code coverage is only secondary*. For instance, suppose a branch somewhere in the program is exercised only if for some variable `i` we have `i == 780234`. Then this branch may (or may not) have a very low probability to be exercised randomly. Instead, the technique shall achieve confidence and expose errors. In our investigations, we also account for partitions that are relatively small, possibly containing only one input.

2.2 Definitions and Notations

Given any program $\mathcal{P}$, the number of input variables to the program determine the *dimensionality* of the program's input space. The values for an input variable determines the values of the corresponding dimension in the program's input space. For instance, a program with two input variables of type integer has a two dimensional input space that can take any integer values. Regarding the input space, we make the following *assumptions*:

- **Bounded Dimensionality**. Given any program $\mathcal{P}$, the space of inputs to $\mathcal{P}$ has a bounded dimension. This assumption is realistic since the length of $\mathcal{P}$ is bounded, it can only manipulate a bounded number of variables.
- **Bounded Input Space**. Given any program $\mathcal{P}$, every input variable $\mathcal{P}$ can take only a bounded number of values from a finite domain. This assumption is also realistic since in practice the size of the registers where the variables are stored is bounded.

Given these assumptions, we see that given a program $\mathcal{P}$, its input space can be taken to be a finite, measurable metric space $\mathcal{D} = \prod_{i=1}^{d} A_i$ where d is the dimension of the input space of $\mathcal{P}$ and A_i is a finite set for every $1 \leq i \leq d$. In what follows, we fix a program $\mathcal{P}$ which in turn fixes the dimension d and the input space $\mathcal{D}$.

Definition 1 (*Error-based Partitioning*)

The input space $\mathcal{D}$ of a program $\mathcal{P}$ can be partitioned into k disjoint non-empty subdomains $\mathcal{D}_i$ where $1 \leq i \leq k$ with the following property: Either every input $t \in \mathcal{D}_i$ reveals an error, or every input $t \in \mathcal{D}_i$ does not reveal an error. If every input of a partition $\mathcal{D}_i$ reveals an error then we call $\mathcal{D}_i$ an error-revealing *partition.*

We notice that Def. 1 requires *determinism*: All executions of the same test input yield the same output. This is satisfied also if a model that *renders* an execution deterministic, like a specific thread schedule, is constituent of the test input.

Since $\mathcal{D}$ is finite, k will be finite, too. Note that $|\mathcal{D}_i| > 0$ for all $1 \leq i \leq k$ where $|\cdot|$ denotes the size (cardinality) of a set and

$$|\mathcal{D}| = \sum_{i=1}^{k} |\mathcal{D}_i| \tag{1}$$

If we draw an input t uniformly at random from $\mathcal{D}$, for every partition $\mathcal{D}_i$ there is a probability that $t \in \mathcal{D}_i$. We denote this probability by p_i. Note that

$$p_i = \frac{|\mathcal{D}_i|}{|\mathcal{D}|}, \text{ for all } i \quad \text{and} \tag{2}$$

$$\sum_{i=1}^{k} p_i = 1 \tag{3}$$

If all partitions are of equal size, $|\mathcal{D}_1| = \cdots = |\mathcal{D}_k|$, then $p_i = 1/k$ for all $1 \leq i \leq k$.

For every $i : 1 \leq i \leq k$, let θ_i be the indicator random variable which is 1 if partition $\mathcal{D}_i$ reveals an error and 0 otherwise. The *failure rate* θ of program $\mathcal{P}$ [9] is given as

$$\theta = \sum_{i=1}^{k} p_i \theta_i \tag{4}$$

A *testing technique* samples the input space of the program-under-test and discovers error-based partitions. We assume that the information whether a partition does or does not reveal an error is unknown a-priori. This is a fair assumption because otherwise there was no need for testing. Hence, each sampled test case becomes a *witness* of whether or not the corresponding partition is error-revealing.

Definition 2 (*Discovered Partitions*)

Given a testing technique $\mathcal{F}$ that samples the input space $\mathcal{D}$, we say that $\mathcal{F}$ discovers *partition D_i in iteration $j \geq 1$ if no test case has been sampled from D_i in any previous iteration $j' < j$.*

While the goal of software verification is to show the correctness of the program for *all* inputs, the goal of software testing is to show the correctness of the program at least for *some* x% of the input. Arguably, this more modest goal may also be more practical and economical.

Definition 3 (*Achieving Confidence*)

For a testing technique $\mathcal{F}$ that samples the input space $\mathcal{D}$ and in j iterations discovers partitions $\mathscr{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_m\}$, we say that $\mathcal{F}$ achieves the degree of confidence *x in j iterations if the following holds*

$$\frac{\sum_{i=1}^{m} |\mathcal{D}_i|}{|\mathcal{D}|} \geq x$$

Now, we define two particular testing techniques, random testing $\mathcal{R}$ and the systematic testing technique $\mathcal{S}_0$. For each technique we assign a sampling cost that corresponds to the time that is required for sampling a test input. The *sampling* of a test input comprises of concrete tasks such as generating and executing the corresponding test case and checking the correctness of its outcome. The sampling cost is computed as the sum of the time it takes each sampling-related task.

Definition 4 (*Random Testing $\mathcal{R}$*)

Given a program $\mathcal{P}$, random testing R tests $\mathcal{P}$ by sampling at each iteration its input space $\mathcal{D}$ uniformly at random. The cost for each sampling is one *unit of time.*

Note that random testing $\mathcal{R}$ *samples with replacement*. The cost for each sampling of one unit of time includes the time to generate and execute the corresponding test case and verify the correctness of its output.

Definition 5 (*Systematic Testing Technique $\mathcal{S}_0$*)

Given a program $\mathcal{P}$, the systematic testing technique $\mathcal{S}_0$ tests $\mathcal{P}$ by sampling at each iteration exactly one undiscovered error-based partition uniformly at random. The sampled partition itself is also chosen uniformly at random from the remaining undiscovered error-based partitions. The cost for each sampling is c units of time.

Note that $\mathcal{S}_0$ samples exactly one input from each error-based partition. Eventually, $\mathcal{S}_0$ will have discovered all partitions and is thus *most effective*. The cost for each sampling of c unit of time includes the time to generate and execute the corresponding test case and verify the correctness of its output *and* the time it takes for the additional analysis. Hence, we call c the *analysis cost* of $\mathcal{S}_0$. Since $\mathcal{S}_0$ *samples without replacement*, it discovers all of k partitions in $\mathsf{c}k$ units of time.

We note, both techniques can sample from a *reduced* input subdomain that contains only e.g., valid, readable, or typical test cases instead of sampling the program's *complete* input space if such are concerns. We make no such assumptions.

We now delve into the technical details. In the following, we shall formalise relevant concepts of approximation and exponential decay.

Definition 6 (*Asymptotics*)

Let $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ be real functions. We say

1. *$f \sim g$ if $\frac{f(n)}{g(n)} \to 1$ as $n \to \infty$. Thus, for every $\epsilon > 0$ there exists $n_0 \in \mathbb{R}^+$ such that for every $n > n_0$, $|f(n) - g(n)| < \epsilon$.*
2. *$f \lesssim g$ if there exist constants $c, n_0 \in \mathbb{R}^+$ such that $|f(n)| < c|g(n)|$ for all $n > n_0$.*
3. *$f \gtrsim g$ if there exist constants $c, n_0 \in \mathbb{R}^+$ such that $|f(n)| > c|g(n)|$ for all $n > n_0$.*

Note, if $f \lesssim g$ then $g \gtrsim f$ and conversely.

Definition 7 (*Exponential Decay*)

A function $f : \mathbb{R} \to \mathbb{R}$ has exponential decay *if it is differentiable at every $x \in \mathbb{R}$ and $\frac{df(x)}{dx} = -\lambda f(x)$ for some constant λ. In particular note that the function $ae^{-\lambda x}$ where a is a constant has exponential decay.*

3. TESTING TO ACHIEVE CONFIDENCE

While the goal of software verification is to show correctness of a program for *all* inputs, one goal of software testing is to show correctness at least for *some* x% of the input – that is to say, to establish a certain degree of confidence x. Given a degree of confidence x, we compare the expected time it takes to achieve x by random testing $\mathcal{R}$ and by the systematic testing technique $\mathcal{S}_0$. After introducing the concepts and insights with an example, we investigate the efficiency of $\mathcal{S}_0$ and $\mathcal{R}$. For $\mathcal{S}_0$, the expected degree of confidence established grows linearly with time. In contrast, for $\mathcal{R}$ it is subject to exponential decay.

Given a degree of confidence x, we find that the analysis cost of $\mathcal{S}_0$ must be below $(ex - ex^2)^{-1}$ units of time in order to remain more efficient than $\mathcal{R}$. For example, to establish that the program works correctly for 90% of its input, sampling one test systematically must take much less than *five* times the time it takes to sample one test randomly.

3.1 Efficiency of $\mathcal{S}_0$ and $\mathcal{R}$ (Confidence)

In this work, we define the confidence that is achieved wrt. the input space that is discovered (Def. 3). So, we give the expected input space that is discovered by $\mathcal{S}_0$ after n units of time.

Lemma 1 (*Confidence – Systematic $\mathcal{S}_0$*)

For the systematic testing technique $\mathcal{S}_0$, the expected input space discovered after n time units is

$$f_s(n) = \frac{|\mathcal{D}|}{\mathsf{c}k} \cdot n$$

where c is the number of units of time taken for sampling one test input.

Proof: By Definition 5, $\mathcal{S}_0$ discovers n/c partitions in n units of time. Since the total number of partitions is k and $\mathcal{S}_0$ picks a partition uniformly at random from the set of undiscovered partitions, the expected contribution of some partition $\mathcal{D}_i$ in any given trial is $\frac{1}{k}|\mathcal{D}_i|$. Hence the expected contribution of $\mathcal{D}_i$ in n time units is $\frac{n}{\mathsf{c}k}|\mathcal{D}_i|$. By the linearity of expectation, we have, the expected input space discovered in n time units is $\frac{n}{\mathsf{c}k}\sum_{i=1}^{k}|\mathcal{D}_i| = \frac{n|\mathcal{D}|}{\mathsf{c}k}$. ■

Thus, the expected size of the input space discovered grows linearly with the number of iterations. As the cost increases, the slope with the time-axis, $|\mathcal{D}|/(\mathsf{c}k)$, of $f_s(n)$ decreases.

Now, we look at the case for random testing.

Lemma 2 (*Confidence – Random $\mathcal{R}$*)

For random testing $\mathcal{R}$, the expected size of the input space discovered after n units of time is

$$f_r(n) = |\mathcal{D}|\left[1 - \sum_{i=1}^{k} p_i(1-p_i)^n\right]$$
$$\sim |\mathcal{D}|\left[1 - \sum_{i=1}^{k} p_i e^{-np_i}\right]$$

Proof: By Definition 4, $\mathcal{R}$ samples n tests in n units of time. Let X_i be the indicator random variable denoting the event that partition $\mathcal{D}_i$ has been discovered within these n trials. The probability to discover $\mathcal{D}_i$ in any given trial is p_i. The probability that $\mathcal{D}_i$ is not discovered after n trials is $(1-p_i)^n$. Thus, the probability that it will be discovered in n trials

is $1-(1-p_i)^n$. Let the expected size of the input space discovered after n units of time be given by the function $f_r : \mathbb{N} \rightarrow \mathbb{R}$. We have

$$f_r(n) = E\left[\sum_{i=1}^{n} X_i | \mathcal{D}_i|\right] \tag{5}$$

$$= \sum_{i=1}^{k} |\mathcal{D}_i| E[X_i] \text{ [by linearity of expectation]} \tag{6}$$

$$= \sum_{i=1}^{k} |\mathcal{D}_i| [1-(1-p_i)^n] \tag{7}$$

$$= |\mathcal{D}| \sum_{i=1}^{k} p_i [1-(1-p_i)^n] \text{ [by Eqn. (2)]} \tag{8}$$

$$= |\mathcal{D}| \left[1 - \sum_{i=1}^{k} p_i (1-p_i)^n\right] \text{ [by Eqn. (3)]} \tag{9}$$

To approximate the above quantity, we cast the problem of achieving confidence into the problem of finding the *bonus sum* in the generalized coupon collectors problem [22]. Given $|\mathcal{D}|$ coupons with k different colours, there are $|\mathcal{D}_i|$ coupons of a colour i where $1 \leq i \leq k$ and each coupon has a bonus value of $|\mathcal{D}_i|$. Note that the probability to collect a coupon of colour i is $p_i = |\mathcal{D}_i|/|\mathcal{D}|$. Then the above quantity is nothing but the bonus sum of the coupons collected after a person collected n coupons when counting the bonus value of each colour only once. From the result of Rósen [22, Theorem 1] we have

$$f_r(n) \sim |\mathcal{D}| \left[1 - \sum_{i=1}^{k} p_i e^{-np_i}\right]$$

■

3.2 Example for Equal-Sized Partitions

We illustrate the main insights for the simplified case where the size of each partition is equal, $|\mathcal{D}_1| = \cdots = |\mathcal{D}_k|$ and hence $p_i = \frac{1}{k}$ for all $i : 1 \leq i \leq k$. In this setting, we demonstrate that the confidence achieved per unit of time decays exponentially for random testing $\mathcal{R}$ while it grows linearly for the systematic testing technique $\mathcal{S}_0$. Later, this result is generalized for partitions of arbitrary size.

First, we show a simple corollary of Lemma 2.

Corollary 1

For random testing $\mathcal{R}$ where $p_i = \frac{1}{k}$ for all $i : 1 \leq i \leq k$, the expected size of input space discovered after n time units is

$$\bar{f}_r(n) = |\mathcal{D}| \left[1 - \left(1 - \frac{1}{k}\right)^n\right]$$
$$= |\mathcal{D}| - |\mathcal{D}| e^{-\lambda n}$$

where $\lambda = \ln(\frac{k}{k-1})$.

Proof: The proof follows directly from Lemma 2 when setting $p_i = \frac{1}{k}$ for every $1 \leq i \leq k$ in $f_r(n)$. ■

Figure 1 shows the expected size of input space discovered per unit of time for $\mathcal{R}$ and $\mathcal{S}_0$ when $k = 100$ and $\mathsf{c} = 2$. So, it takes $\mathcal{S}_0$ twice as long to sample a test input compared to $\mathcal{R}$. On the average, after 80 units of time, $\mathcal{S}_0$ discovered partitions in 40% of the input space while $\mathcal{R}$ discovered partitions in 55% of the program's input space. On the average, after 160 units of time both techniques break even, having discovered partitions in 80% of the input space.

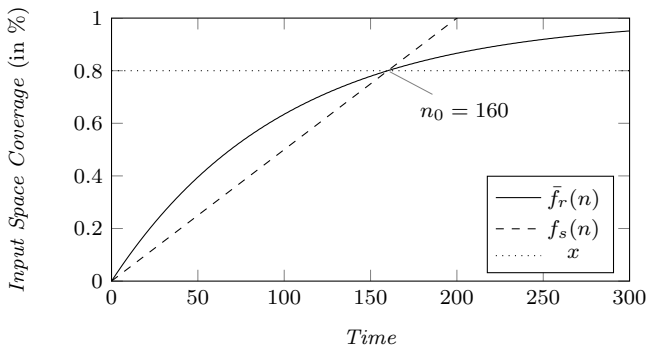

Figure 1: On the average, $\mathcal{S}_0$ and $\mathcal{R}$ break even after approximately 80% of the input space was covered and 160 random test inputs were sampled (when $\mathsf{c} = 2$, $k = 100$, $p_i = \frac{1}{k}$).

There exists a time n_0 where $\bar{f}_r(n_0) = f_s(n_0)$ and $\mathcal{S}_0$ has achieved more confidence than $\mathcal{R}$ for any $n > n_0$, on the average. To assess the *relative efficiency* of $\mathcal{S}_0$ we pose the following question: Given a degree of confidence x, what is the maximum cost c_0 for $\mathcal{S}_0$ such that $\mathcal{S}_0$ achieves x in time $n \leq n_0$? We give the answer by the following lemma

Lemma 3

Given a degree of confidence x, let n_s and n_r be the time at which $\mathcal{S}_0$ and $\mathcal{R}$ are expected to achieve x, respectively. When $p_i = \frac{1}{k}$ for every $i : 1 \leq i \leq k$, the maximum cost c_0 of $\mathcal{S}_0$, such that $n_s \leq n_r$, is given as

$$\mathsf{c}_0 = \breve{c} \cdot \left[\frac{1}{x} \ln\left(\frac{1}{1-x}\right)\right]$$

for a constant $\breve{c}$.

Proof: Setting $f_s(n) = |\mathcal{D}|x$ gives

$$n = xk\mathsf{c}_0 \tag{10}$$

Setting $\bar{f}_r(n) = |\mathcal{D}|x$ yields

$$x = 1 - \left(1 - \frac{1}{k}\right)^n \tag{11}$$

$$= 1 - \left(1 - \frac{1}{k}\right)^{xk\mathsf{c}_0} \text{ [by Eqn. (10)]} \tag{12}$$

Solving for c_0 gives

$$\mathsf{c}_0 = \frac{\ln(1-x)}{\ln\left(\left(\frac{k-1}{k}\right)^{xk}\right)} \tag{13}$$

$$= \breve{c} \cdot \left[\frac{1}{x} \ln\left(\frac{1}{1-x}\right)\right] \tag{14}$$

where

$$\breve{c} = \frac{1}{k \ln\left(\frac{k}{k-1}\right)} \tag{15}$$

■

Figure 2 shows for the segment from $x : 0.8 \leq x \leq 1$ the exact cost c_0 for $\mathcal{S}_0$ such that both techniques are expected to break even at a given degree of confidence. Giving the degree of confidence $x = 0.8$, the maximum cost is $\mathsf{c}_0 = 2$ and both techniques are expected to break even at x as shown in Figure 1. For $x = 99$, we see $\mathsf{c}_0 = 4.65$ in Fig. 2

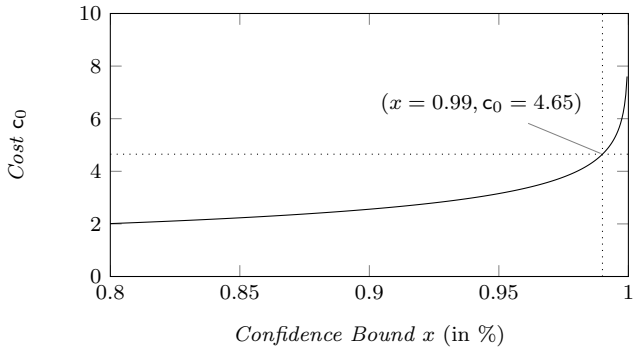

Figure 2: If the average analysis cost of $\mathcal{S}_0$ exceeds c_0 for a given degree of confidence x, then $\mathcal{R}$ is generally more efficient than $\mathcal{S}_0$ (here for $p_i = \frac{1}{k}$).

3.3 Bounds on the Expected Size of the Input Space Discovered for Random Testing

Under the simplified conditions of the example, where each partition has the same size, $|\mathcal{D}_1| = \cdots = |\mathcal{D}_k|$, we have shown that the confidence achieved per unit of time *decays exponentially* for random testing. In the following, we prove that this is the case for partitions of arbitrary sizes. Towards that, we define two quantities p_{min} and p_{max}.

$$p_{\max} = \max_{i=1}^{k}\{p_i\} \text{ and } p_{\min} = \min_{i=1}^{k}\{p_i\} \tag{16}$$

where the functions *max* and *min* compute the maximum and minimum number in a given set, respectively. Note that $p_{\max} \geq 1/k$ and $p_{\min} \leq 1/k$. We claim

Lemma 4 (*Approximate Bounds*)

$f_r(n)$ is bounded above and below approximately as

$$|\mathcal{D}|[1 - kp_{\min}e^{-np_{\min}}] \lesssim f_r(n) \lesssim |\mathcal{D}|[1 - kp_{\max}e^{-np_{\max}}]$$

Proof: Let us denote the quantity $\sum_{i=1}^{k} p_i(1-p_i)^n$ by $q(n)$. Let $I_{\max} \subseteq \{1, 2, \ldots, k\}$ be the set of indices such that $p_{\max} - p_i > 0$ iff $i \in I_{\max}$. Then, for all $i \in I_{\max}$ we have $\frac{\ln(p_{\max}) - \ln(p_i)}{p_{\max} - p_i} > 0$. Let

$$n_i \geq \frac{\ln(p_{\max}) - \ln(p)}{p_{\max} - p_i} \tag{17}$$

Note, $p_{\max} \neq p_i$ for $i \in I_{\max}$. This implies

$$\frac{e^{-n_i p_i}}{e^{-n_i p_{\max}}} \geq \frac{p_{\max}}{p_i} \tag{18}$$

whence we get

$$p_{\max}e^{-n_i p_{\max}} \leq p_i e^{-n_i p_i} \tag{19}$$

Let $n_{\max} = \max_{i \in I_{\max}}\{n_i\}$ Thus for all $n \geq n_{\max}$ we have

$$\begin{aligned}
\sum_{i=1}^{k} p_i e^{-np_i} &= \sum_{i \in I_{\max}} p_i e^{-np_i} + \sum_{i \notin I_{\max}} p_i e^{-np_i} \\
&= \sum_{i \in I_{\max}} p_i e^{-np_i} + \sum_{i \notin I_{\max}} p_{\max} e^{-np_{\max}} \\
&\qquad [\text{since } p_i = p_{\max} \text{ for } i \notin I_{\max}] \\
&\geq \sum_{i \in I_{\max}} p_{\max} e^{-np_{\max}} + \sum_{i \notin I_{\max}} p_{\max} e^{-np_{\max}} \\
&\qquad [\text{by Eqn. (19)}] \\
&= kp_{\max}e^{-np_{\max}}
\end{aligned}$$

Similarly, let $I_{\min} \subseteq \{1, 2, \ldots, k\}$ be the set of indices of the error-based partitions such that $p_i - p_{\min} > 0$ iff $i \in I_{\min}$ Let

$$n_{\min} = \max_{i \in I_{\min}} \left\{ \frac{\ln(p_i) - \ln(p_{\min})}{p_i - p_{\min}} \right\} \tag{20}$$

We can show for all $n \geq n_{\min}$ that

$$\sum_{i=1}^{k} p_i e^{-np_i} \leq kp_{\min}e^{-np_{\min}} \tag{21}$$

So, for all $n \geq \max\{n_{\min}, n_{\max}\}$, we have

$$kp_{\max}e^{-np_{\max}} \leq \sum_{i=1}^{k} p_i e^{-np_i} \leq kp_{\min}e^{-np_{\min}} \tag{22}$$

$$kp_{\max}e^{-np_{\max}} \lesssim q(n) \lesssim kp_{\min}e^{-np_{\min}} \quad [\text{by Rósen [22]}] \tag{23}$$

Hence

$$|\mathcal{D}|[1 - kp_{\min}e^{-np_{\min}}] \lesssim f_r(n) \lesssim |\mathcal{D}|[1 - kp_{\max}e^{-np_{\max}}] \tag{24}$$ ■

Thus $f_r(n)$ being bounded above and below by exponential functions also behaves like one.

3.4 Relative Efficiency of $\mathcal{S}_0$ (Confidence)

We evaluate the efficiency of the systematic testing technique $\mathcal{S}_0$ relative to that of random testing $\mathcal{R}$. Because of the additional analysis cost, sampling a test input using $\mathcal{S}_0$ takes c times longer than sampling a test input using $\mathcal{R}$. Since *in general* the achieved confidence per unit of time decays exponentially for $\mathcal{R}$ while it grows linearly for $\mathcal{S}_0$, there is a point where $\mathcal{S}_0$ and $\mathcal{R}$ are expected to break even. Its coordinates depend on the value of c.

Given a degree of confidence x, we compute the maximum cost c_0 such that the expected time it takes for $\mathcal{S}_0$ to achieve x is at most the same as the expected time it takes $\mathcal{R}$ to achieve x and $\mathcal{S}_0$ remains more efficient than $\mathcal{R}$.

Proposition 1

Given a degree of confidence $x : 1 - e^{-1} \leq x < 1$, let n_s and n_r be the time at which $\mathcal{S}_0$ and $\mathcal{R}$ are expected to achieve x, respectively. For all programs $\mathcal{P}$, the maximum cost c_0 of $\mathcal{S}_0$, such that $n_s \leq n_r$, is bounded above as

$$\mathsf{c}_0 \lesssim \frac{1}{ex - ex^2}$$

Proof: Fix a program $\mathcal{P}$ which in turn fixes the number of partitions k and also the probabilities p_i for all $i : 1 \leq i \leq k$. Let $\mathsf{c}_0^{\mathcal{P}}$ be the cost of $\mathcal{S}_0$, such that $n_s = n_r$ for $\mathcal{P}$. Now, setting $f_s(n) = |\mathcal{D}|x$ yields

$$n = xk\mathsf{c}_0^{\mathcal{P}} \tag{25}$$

Setting $f_r(n) = |\mathcal{D}|x$ gives

$$x \sim 1 - \sum_{i=1}^{k} p_i e^{-np_i} \tag{26}$$

$$\gtrsim 1 - kp_{\min}e^{-np_{\min}} \qquad [\text{by Lemma 4}] \tag{27}$$

$$\gtrsim 1 - kp_{\min}e^{-xk\mathsf{c}_0^{\mathcal{P}} p_{\min}} \qquad [\text{by Eqn. (25)}] \tag{28}$$

Solving for $\mathsf{c}_0^{\mathcal{P}}$ gives,

$$\mathsf{c}_0^{\mathcal{P}} \lesssim \frac{\ln\left(\frac{kp_{\min}}{1-x}\right)}{kxp_{\min}} \tag{29}$$

Let us denote $\frac{\ln\left(\frac{kp_{\min}}{1-x}\right)}{kxp_{\min}}$ as $h(k, p_{\min})$. From (29),

$$\mathsf{c}_0 \leq \max_{\mathcal{P}}\{\mathsf{c}_0^{\mathcal{P}}\} \lesssim \max_{\mathcal{P}}\{h(k, p_{\min})\} \tag{30}$$

where $\max_{\mathcal{P}}$ denotes the maximum of the quantity $h(k, p_{\min})$ over all programs.

To find the value $\max_{\mathcal{P}}\{h(k, p_{\min})\}$, we first relax the requirement that k takes integral values and allow k to range over the reals $\mathbb{R}$. By doing so we notice that $h(k, p_{\min})$ is a continuous function over $(\mathbb{R} \times [0, 1])$ which is differentiable everywhere. This allows us to use techniques from differential calculus to maximize $h(k, p_{\min})$ wrt $p_{\min}$ and k. [As we shall see below, $h(k, p_{\min})$ will have exactly one global extremum at some non-boundary point. Hence, the value of $\max_{\mathcal{P}}\{h(k, p_{\min})\}$, with the original requirement that k ranges over the discrete integral domain, will be attained at one of the two nearest integers.]

We first set the partial derivative of $h(k, p_{\min})$ wrt $p_{\min}$ to 0.

$$\frac{\partial}{\partial p_{\min}} \frac{\ln\left(\frac{kp_{\min}}{1-x}\right)}{kxp_{\min}} = 0 \tag{31}$$

This yields a critical point for $h(k, p_{\min})$ when

$$p_{\min} = \frac{e - ex}{k} \tag{32}$$

The second partial derivative of $h(k, p_{\min})$ wrt $p_{\min}$ is given by

$$\frac{\partial^2}{\partial p_{\min}^2} \frac{\ln\left(\frac{kp_{\min}}{1-x}\right)}{kxp_{\min}} = \frac{-3 + 2\ln\left(\frac{kp_{\min}}{1-x}\right)}{kxp_{\min}^3} \tag{33}$$

Hence for $h(k, p_{\min})$ to be maximal wrt $p_{\min}$ it must hold that

$$\frac{-3 + 2\ln\left(\frac{kp_{\min}}{1-x}\right)}{kxp_{\min}^3} < 0 \tag{34}$$

which yields

$$p_{\min} < \frac{e\sqrt{e}(1-x)}{k} \tag{35}$$

Since (32) satisfies (35) we have that $h(k, p_{\min})$ attains a maximum wrt $p_{\min}$ at $p_{\min} = \frac{e-ex}{k}$.

By a similar analysis we can demonstrate that $h(k, p_{\min})$ attains a maximum wrt k at $k = \frac{e-ex}{p_{\min}}$ which is the same as Eqn. (32), $p_{\min} = \frac{e-ex}{k}$. Plugging $p_{\min} = \frac{e-ex}{k}$ into $h(k, p_{\min})$ we get

$$\mathsf{c}_0 \lesssim \frac{1}{ex - ex^2} \tag{36}$$

Finally, to derive the bounds on the degree of confidence x for which the above inequality holds, note that it must also hold that $0 < p_{\min} \leq 1/k$ whence from Equation (32) we have

$$0 < \frac{e - ex}{k} \leq \frac{1}{k} \tag{37}$$

which gives

$$1 - e^{-1} \leq x < 1 \tag{38}$$ ■

Corollary 2

Given the degree of confidence $x \in \{0.8, 0.9, 0.95, 0.98, 0.99\}$, the maximum cost c_0^x of $\mathcal{S}_0$, such that $\mathcal{S}_0$ is expected to achieve x in at most the same time as $\mathcal{R}$ is given as

$$\mathsf{c}_0^{0.8} \lesssim 2.3, \quad \mathsf{c}_0^{0.9} \lesssim 4.1, \quad \mathsf{c}_0^{0.95} \lesssim 7.8,$$
$$\mathsf{c}_0^{0.98} \lesssim 19, \quad \mathsf{c}_0^{0.99} \lesssim 38$$

Proof: The proof follows directly from Proposition 1. ■

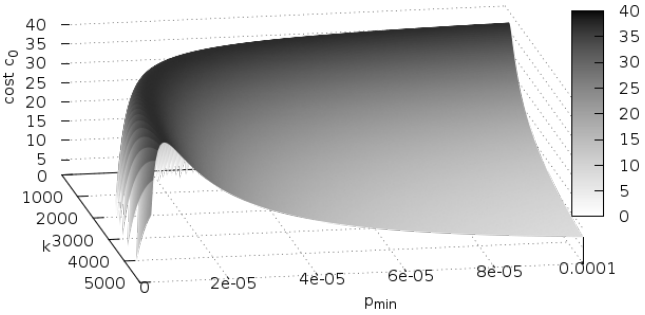

Figure 3: The maximum for the cost c_0 in terms of p_{min} and k (for the degree of confidence $x = 0.99$).

Figure 3 provides a *graphical interpretation* of the Equation (29) for the $x = 0.99$ degree of confidence. The figure shows that the maximum cost c_0 for all programs with the total number of partitions $k : 0 < k < 5000$ and the probability for the smallest partition $p_{\min} : 0 < p_{\min} < 0.0001$ reaches a maximum for some programs at below a c_0 of 40. By Proposition 1, this holds in general for all programs.

4. TESTING TO DISCOVER ERRORS

Besides achieving confidence in the program's correctness, another definition of effective software testing is to discover errors (cf. **E-measure** [1]). So, given the same time, we compare the expected number of errors found by random testing $\mathcal{R}$ with the expected number of errors found by the systematic testing technique $\mathcal{S}_0$. After illustrating our main insights by an example, we investigate the efficiency of $\mathcal{S}_0$ and $\mathcal{R}$ w.r.t. the expected number of errors discovered. The expected number of errors discovered per unit of time grows linearly for $\mathcal{S}_0$ while it decays exponentially for $\mathcal{R}$.

First, we slightly strengthen Definition 1 stating that failing inputs revealing *the same error* are grouped into one partition. This is reasonable because in practice several failing inputs may witness the same error.

Definition 8 (*Error-based Partitioning'*)

The input space $\mathcal{D}$ of a program $\mathcal{P}$ can be partitioned into k disjoint non-empty subdomains $\mathcal{D}_i$ where $1 \leq i \leq k$ with the following property: Either every input $t \in \mathcal{D}_i$ reveals the same error, or no input $t \in \mathcal{D}_i$ reveals an error.

Thus, the number of error-revealing partitions discovered corresponds to the number of errors found.

Given a time bound $\hat{n}$, we find that the expected number of errors discovered by $\mathcal{R}$ within $\hat{n}$ time units is less than or equals that of $\mathcal{S}_0$ only if the analysis cost c incurred by $\mathcal{S}_0$ is less than $\frac{\hat{n}}{k} \cdot (1 - (1 - q_{\min})^{\hat{n}})^{-1}$, where k is the number of error-based partitions, and $q_{\min}$ is the fractional size of the *"smallest" error-revealing* partition in the program's input space.

Duran and Ntafos [9] define a quantity θ_i for every partition $\mathcal{D}_i$ which gives the probability of that partition to reveal an error. In our setting, θ_i is either 0 or 1 and can be defined as

$$\theta_i = \begin{cases} 1 & \text{if } \mathcal{D}_i \text{ is error-revealing} \\ 0 & \text{otherwise} \end{cases}$$

Then the total number of errors in $\mathcal{P}$ is given by $z = \sum_{i=1}^{k} \theta_i$.

4.1 Efficiency of $\mathcal{S}_0$ and $\mathcal{R}$ (Errors Found)

First, we give the expected number of errors found per unit of time, i.e., the efficiency, for the systematic testing technique $\mathcal{S}_0$.

Lemma 5 (*Errors Found – Systematic $\mathcal{S}_0$*)

For the systematic testing technique $\mathcal{S}_0$, the expected number of errors discovered after n time units is

$$g_s(n) = \frac{z}{\mathsf{c}k} \cdot n$$

for $n : 0 \leq n \leq k$, where every trial "costs" c units of time.

Proof: By Definition 5, $\mathcal{S}_0$ performs n/c draws in n units of time. In this classical urn problem of sampling without replacement we shall call the discovery of an error(-revealing partition) a "success". The expected number of successes in n/c draws without replacement from a finite population k containing z successes is given by $\frac{z}{\mathsf{c}k} \cdot n$. ■

The expected number of errors discovered w.r.t the number of iterations grows linearly. As the cost c increases, the slope with the time-axis, $z/\mathsf{c}k$, of the line, $g_s(n)$, decreases.

Now, we look at the case for random testing.

Lemma 6 (*Errors Found – Random [9]*)

For random testing $\mathcal{R}$, the expected number of errors discovered after n time units is

$$g_r(n) = k - \sum_{i=1}^{k} (1 - p_i\theta_i)^n$$

The proof is due to Duran and Ntafos [9]. By Definition 4, every iteration occurs in one unit of time.

4.2 Example for Equal-Sized Partitions

We illustrate the main insights for the simplified case where the size of each partition is equal, $|\mathcal{D}_1| = \cdots = |\mathcal{D}_k|$ and hence $p_i = \frac{1}{k}$ for all $1 \leq i \leq k$. In this setting, we demonstrate that the efficiency decays exponentially for $\mathcal{R}$ while it grows linearly for $\mathcal{S}_0$. Later, this result is generalized for partitions of arbitrary size.

First, we derive the corollary of Lemma 6.

Corollary 3

For random testing $\mathcal{R}$ where $p_i = \frac{1}{k}$ for all $1 \leq i \leq k$, the expected number of errors found after n time units is

$$\bar{g}_r(n) = z - z\left(1 - \frac{1}{k}\right)^n$$
$$= z - ze^{-\lambda n}$$

where $\lambda = \ln\left(\left[1 - \frac{1}{k}\right]^{-1}\right)$.

Proof: There are a total of z number of error-revealing partitions. After setting $p_i = \frac{1}{k}$ in the formula of Lemma 6, we have for z number of partitions that $\theta_i = 1$ and for $k - z$ number of partitions that $\theta_i = 0$. Thus,

$$\bar{g}_r(n) = k - \sum_{i=1}^{k} \left(1 - \frac{\theta_i}{k}\right)^n \tag{39}$$
$$= k - \left((k - z) + z\left(1 - \frac{1}{k}\right)^n\right) \tag{40}$$
$$= z - z\left(1 - \frac{1}{k}\right)^n \tag{41}$$
$$= z - ze^{-\ln\left((1-\frac{1}{k})^{-1}\right)n} \tag{42}$$

■

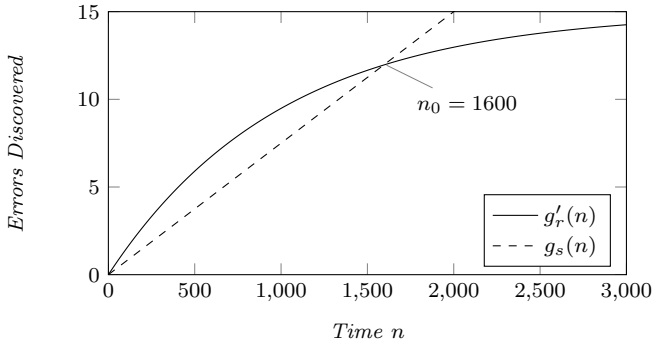

Figure 4: On the average, $\mathcal{S}_0$ and $\mathcal{R}$ break even after 12 of 15 errors were discovered and 1600 random test inputs were sampled (when $\mathsf{c} = 2$, $z = 15$, $k = 1000$, $p_i = \frac{1}{k}$).

Figure 4 depicts the expected number of discovered errors per unit of time for random testing and $\mathcal{S}_0$ in our example configuration. As the cost c is 2, it takes $\mathcal{S}_0$ twice as long to sample a test input compared to $\mathcal{R}$. After 800 units of time, $\mathcal{S}_0$ discovered 6 of $z = 15$ errors on the average, while $\mathcal{R}$ discovered 2.2 errors *more*, on the average. After 1600 units of time, both techniques discovered 12 of $z = 15$ errors, on the average. This the point of time where both testing schemes, $\mathcal{S}_0$ and $\mathcal{R}$, are expected to break even.

There exists a time n_0 where $\bar{g}_r(n_0) = g_s(n_0)$ meet and $\mathcal{S}_0$ has discovered more errors than $\mathcal{R}$ for any $n > n_0$, on the average. To assess the *relative efficiency* of $\mathcal{S}_0$ we pose the following question: Given a time bound $\hat{n}$, what is the maximum cost c_0 for $\mathcal{S}_0$ such that $n_0 \leq \hat{n}$?

Lemma 7

In the case where $p_i = \frac{1}{k}$ for every $1 \leq i \leq k$, the maximum cost c_0 of the systematic testing technique $\mathcal{S}_0$ – such that the expected number of errors discovered by $\mathcal{S}_0$ is at least the same as the expected number of errors discovered by random testing in $\hat{n}$ units of time – is given as

$$\mathsf{c}_0 = \frac{\hat{n}}{k(1 - (1 - \frac{1}{k})^{\hat{n}})}$$
$$\sim \frac{\hat{n}}{k} \qquad \text{as } \hat{n} \to \infty$$

Proof: The proof follows directly from Lemma 5 and Corr. 3 when fixing n to $\hat{n}$ and setting $\bar{g}_r(\hat{n}) = g_s(\hat{n})$. ■

Figure 5 depicts the exact cost c_0 for $\mathcal{S}_0$ such that both techniques are expected to break even at a given time $\hat{n}$.

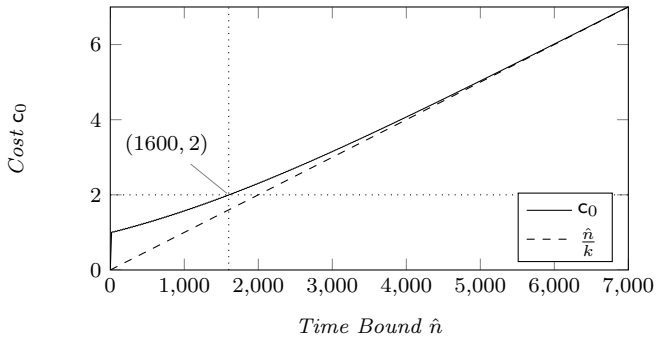

Figure 5: The maximum cost c_0 increases approximately linearly as the given time bound $\hat{n}$ increases. If the average analysis cost of $\mathcal{S}_0$ exceeds c_0 for a given time bound $\hat{n}$, then $\mathcal{R}$ is generally more efficient than $\mathcal{S}_0$ (here for $p_i = \frac{1}{k}$ and $k = 1000$).

Giving a time bound of $\hat{n} = 1600$, the maximum cost is $\mathsf{c}_0 = 2$ and both techniques are expected to break even at $\hat{n}$ as shown in Figure 4. Increasing the time-bound $\hat{n}$, increases the maximum cost c_0 approximately proportionally.

4.3 Tight Bounds on the Expected Number of Errors Discovered for Random Testing

Under the simplified conditions of the example, where each partition has the same size, $|\mathcal{D}_1| = \cdots = |\mathcal{D}_k|$, we see that the efficiency of random testing *decays exponentially*. In the following, we show that this is the case for partitions of arbitrary sizes. Intuitively, random testing discovers many (error-revealing) partitions in the beginning and much less as the number of iterations increases.

Towards that, let $Q \subseteq \{p_1, \ldots, p_k\}$ be a set of probabilities such that $p_i \in Q$ iff $\theta_i = 1$ for all indices $1 \leq i \leq k$. Thus, Q is the set of p_i's corresponding to all the error-revealing partitions $\mathcal{D}_i$.

Let $Q = \{q_1, \ldots, q_z\}$, we define two quantities $q_{\max}$ and $q_{\min}$ as

$$q_{\max} = \max\{q \mid q \in Q\} \text{ and } q_{\min} = \min\{q \mid q \in Q\} \quad (43)$$

where the functions *max* and *min* give the maximum and minimum elements in a given set, respectively. We have

Lemma 8 (*Tight bounds*)

Given a program $\mathcal{P}$, let k be the total number of partitions of the input space out of which z are error-revealing. Let

$$\lambda_{\min} = \ln\left(\frac{1}{1-q_{\min}}\right) \text{ and } \lambda_{\max} = \ln\left(\frac{1}{1-q_{\max}}\right)$$

Then we have

$$z - ze^{-\lambda_{\min} n} \leq g_r(n) \leq z - ze^{-\lambda_{\max} n}$$

Proof:

$$g_r(n) = k - \sum_{i=1}^{k}(1-\theta_i p_i)^n \quad (44)$$

$$= k - \left[\sum_{q_i \in Q}(1-q_i)^n + \sum_{q_i \notin Q} 1\right] \quad (45)$$

$$= k - \sum_{q_i \in Q}(1-q_i)^n - (k-z) \quad (46)$$

$$\leq z - \sum_{i=1}^{z}(1-q_{\max})^n \quad (47)$$

$$= z - ze^{-\lambda_{\max} n} \quad (48)$$

By similar analysis we can show that

$$g_r(n) \geq z - ze^{-\lambda_{\min} n} \quad (49)$$

■

The function $g_r(n)$ being bounded above and below by exponentially decaying functions also behaves like one. Hence $g_r(n)$ also has the nature of an exponential function.

4.4 Relative Efficiency of $\mathcal{S}_0$ (Errors Found)

We evaluate the efficiency of the systematic testing technique $\mathcal{S}_0$ relative to that of random testing $\mathcal{R}$. Because of the additional analysis cost, sampling a test input using $\mathcal{S}_0$ takes c times longer than sampling a test input using $\mathcal{R}$. Since *in general* the efficiency of $\mathcal{R}$, here w.r.t. discovering errors, decays exponentially while that of $\mathcal{S}_0$ grows linearly, there is a point in time where $\mathcal{S}_0$ and $\mathcal{R}$ are expected to break even. The coordinates of this point depend on the value of c.

Given $\hat{n}$ units of time, we compute the maximum cost c_0 such that $\mathcal{S}_0$ remains more efficient than $\mathcal{R}$. Specifically, we compute c_0 such that the expected number of errors discovered by $\mathcal{S}_0$ is at least the same as the expected number of errors discovered by $\mathcal{R}$.

Proposition 2

Given a program $\mathcal{P}$, let k be the total number of partitions of the input space out of which z are error-revealing. Given $\hat{n}$ units of time, let d_r and d_s be the expected number of error-revealing partitions discovered by the systematic testing technique $\mathcal{S}_0$ and random testing $\mathcal{R}$, respectively. Then, the maximum cost c_0 of $\mathcal{S}_0$, such that $d_r \leq d_s$, is given as

$$\mathsf{c}_0 \leq \frac{1}{k} \cdot \frac{\hat{n}}{1-(1-q_{\min})^{\hat{n}}}$$

where $q_{\min}$ is defined as in Eqn. (43).

Proof: Setting $g_s(\hat{n}) = g_r(\hat{n})$ yields

$$\frac{z\hat{n}}{k\mathsf{c}_0} = k - \sum_{i=1}^{k}(1-p_i\theta_i)^{\hat{n}} \quad (50)$$

$$\frac{z\hat{n}}{k\mathsf{c}_0} \geq z - z(1-q_{\min})^{\hat{n}} \quad \text{[By Lemma 8]} \quad (51)$$

Solving for c_0 having $\hat{n} > 0$, $k > 0$, and $z \geq 0$ gives

$$\mathsf{c}_0 \leq \frac{1}{k} \cdot \frac{\hat{n}}{1-(1-q_{\min})^{\hat{n}}} \quad (52)$$

■

5. PRACTICAL IMPLICATIONS

In this paper we present strong, elementary, theoretical results about the efficiency of automated software testing. For thirty years [9], we have struggled to understand how automated random testing and systematic testing seem to be almost *on par* [16, 30, 8, 28, 15, 25, 24]. It seems yesterday when Arcuri et al. [1] argued that "analytical and empirical analyses have not shown so far a clear inferiority of random testing compared with other more sophisticated techniques".

Today, we have formally proven limits on the efficiency of automated systematic testing beyond which random testing is certainly "superior". We first model an *ideal* systematic testing scheme which we call $\mathcal{S}_0$. By sampling one test input from each error-based partition, $\mathcal{S}_0$ is not only the *most effective* but also a very efficient testing scheme. Next we assume that $\mathcal{S}_0$ incurs a *constant analysis cost* c for each of its trials while random testing does not. Then we argue that there must be a maximum value for c beyond which $\mathcal{S}_0$ is less efficient than random testing.

Now, practical testing schemes are much *less than ideal*. In reality, our testing techniques end up sampling some error-based partitions several times and others not at all. This is because complete certainty about the "true" error-based partitioning is unattainable [29]. In fact, the quality of the approximation depends directly on the analysis cost. The more comprehensive the analysis, the more effective the testing technique. It follows that:

> *In practice, to approach the effectiveness of $\mathcal{S}_0$, we need to increase the analysis cost which in turn decreases the efficiency of the testing technique!*

Moreover, practical testing schemes may be *less efficient for bigger programs*. As opposed to $\mathcal{S}_0$, the efficiency of realistic schemes may not remain constant across all programs. To maintain effectiveness, the analysis must be more comprehensive as the number of program artifacts increases that are analyzed. Since there is an upper bound on the analysis cost which itself is a function of program size, it follows that:

> *In practice, there exists a maximum program size beyond which $\mathcal{R}$ is generally more efficient!*

Testing schemes may become *less efficient during testing*. As opposed to $\mathcal{S}_0$, the analysis cost may not remain constant but increase during testing. Take coverage-based testing for example. It requires almost no analysis to sample an initial set of inputs that cover much of the source code. However, it becomes increasingly difficult to cover the remaining few uncovered code elements [32, 31]. Besides, the order in which the error-based partitions are sampled may not be random (Def. 5). If so, the expected confidence achieved and errors discovered may reduce over time rather than grow linearly.

A practical result of Proposition 1: The "class of nines" for a given degree of confidence x is directly proportional to the magnitude of the maximum analysis cost. The class of nines for degree of confidence x is computed as $\lfloor -\log_{10}(1-x) \rfloor$, where $\lfloor . \rfloor$ is the floor function:

confidence x	**class of nines**	**bound on c**
90%	1 nine	$\mathsf{c} < 4.1 * 10^0$
99%	2 nines	$\mathsf{c} < 4 * 10^1$
99.99%	4 nines	$\mathsf{c} < 4 * 10^3$
99.9999%	6 nines	$\mathsf{c} < 4 * 10^5$

A generalization of Proposition 2: It is trivial to show how the proposition holds for disjoint input subdomains that are homogeneous w.r.t. *other* properties. As fixed in Def. 8, we investigate the efficiency w.r.t. error-based partitioning. However, there is no reason why the partitioning should not be target-based, path-based, or differential, for example. *Target-based partitioning* yields subdomains for which all inputs either do or do not reach a certain target in the source. *Differential partitions* [2] are difference- and equivalence-revealing subdomains in the context of regression testing. *Path-based partitioning* [12, 11] groups inputs that exercise a certain path. To illustrate this generalization of Prop. 2:

> **Question:** We have a program with $k = z = 10^6$ paths where the path with the least probability to be exercised is of fractional size $q_{\min} = 10^{-8}$. We have two testing tools: a symbolic execution tool $\mathcal{S}'$ that exercises each path – one at a time, chosen uniformly at random from paths not exercised – and a random testing tool $\mathcal{R}$ that takes $10ms$ to generate and execute a test case. Finally, we only have one hour ($\hat{n} = 1h$) to exercise as many paths as possible. Which technique should we choose, $\mathcal{R}$ or $\mathcal{S}'$?
>
> ***Answer:*** *We choose $\mathcal{S}'$ only if generating and executing one test case takes, on the average, less than about $1s$!*

To determine $q_{\min}$, we note that Geldenhuys et al. [11] introduced a tool that can measure the probability of a path to be exercised using model counting on the path condition.

6. CONCLUSION

In this paper, we explore two notions of *testing efficiency* that may be the main goals of automated software testing: 1) to show in minimal time the correctness of a program for a given percentage of the program's input domain (Sec. 3) and 2) to discover a maximal number of errors within a given time bound (Sec. 4).

We define a systematic testing technique $\mathcal{S}_0$ that is most effective in terms of both the above notions. Subsequently, we explore the efficiency of $\mathcal{S}_0$ again in terms of both the above notions. However, we believe that our work can also provide the formal framework to explore the efficiency of systematic testing techniques other than $\mathcal{S}_0$.

If the goal is to discover a maximal number of errors within a given time bound, we prove an upper bound on the cost of $\mathcal{S}_0$ and show that it depends on the number of error-based partitions and the fractional size of the "smallest" error-revealing partition. We discuss how this result generalizes to other homogeneous partitionings.

If the goal is to show in minimal time the correctness of a program for a given percentage of the program's input space, we prove an upper bound on the cost of $\mathcal{S}_0$ that depends asymptotically only on the given degree of confidence and holds for all programs-under-test. The existence of an upper bound has great implications on the *scalability* of systematic testing if we consider the analysis cost not as a constant but rather a function on the program size.

7. ACKNOWLEDGMENTS

We would like to thank our colleagues, Abhijeet Banerjee and Dr. Konstantin Rubinov, for the engaging discussions about the content and potential impact of this paper. This work was partially supported by Singapore's Ministry of Education research grant MOE2010-T2-2-073. The first author is funded by an ERC advanced grant 'SPECMATE'.

8. REFERENCES

[1] A. Arcuri, M. Iqbal, and L. Briand. Random testing: Theoretical results and practical implications. *IEEE Transactions on Software Engineering*, 38(2):258–277, March 2012.

[2] M. Böhme, B. C. d. S. Oliveira, and A. Roychoudhury. Partition-based regression verification. In *35th International Conference on Software Engineering*, ICSE'13, pages 302–311, 2013.

[3] M. Böhme, B. C. Oliveira, and A. Roychoudhury. Regression tests to expose change interaction errors. In *ESEC/FSE 2013*, pages 199–209, 2013.

[4] M. Böhme and A. Roychoudhury. Corebench: Studying complexity of regression errors. In *Proceedings of the 23rd ACM/SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA, pages 398–408, 2014.

[5] C. Boyapati, S. Khurshid, and D. Marinov. Korat: Automated testing based on java predicates. In *Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA '02, pages 123–133, 2002.

[6] C. Cadar, D. Dunbar, and D. R. Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *8th USENIX Symposium on Operating Systems Design and Implementation*, OSDI'08, pages 209–224, 2008.

[7] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler. Exe: Automatically generating inputs of death. *ACM Transactions on Information and System Security*, 12(2), 2008.

[8] T. Y. Chen and Y.-T. Yu. On the expected number of failures detected by subdomain testing and random testing. *IEEE Transactions on Software Engineering*, 22(2):109–119, 1996.

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

[10] G. Fraser and A. Arcuri. Evosuite: automatic test suite generation for object-oriented software. In *SIGSOFT/FSE'11*, pages 416–419, 2011.

[11] J. Geldenhuys, M. B. Dwyer, and W. Visser. Probabilistic symbolic execution. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ISSTA 2012, pages 166–176, 2012.

[12] P. Godefroid, N. Klarlund, and K. Sen. Dart: Directed automated random testing. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, pages 213–223, 2005.

[13] P. Godefroid, A. V. Nori, S. K. Rajamani, and S. D. Tetali. Compositional may-must program analysis: Unleashing the power of alternation. In *POPL '10*, pages 43–56, 2010.

[14] A. Goldberg, T. C. Wang, and D. Zimmerman. Applications of feasible path analysis to program testing. In *Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, ISSTA '94, pages 80–94, 1994.

[15] W. Gutjahr. Partition testing vs. random testing: the influence of uncertainty. *Transactions on Software Engineering*, 25(5):661–674, Sep 1999.

[16] D. Hamlet and R. Taylor. Partition testing does not inspire confidence (program testing). *Transactions on Software Engineering*, 16:1402–1411, 1990.

[17] Y. Jia and M. Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, Sept 2011.

[18] B. Korel. Automated software test data generation. *IEEE Transactions on Software Engineering*, 16(8):870–879, 1990.

[19] B. Korel and A. M. Al-Yami. Assertion-oriented automated test data generation. In *Proceedings of the 18th International Conference on Software Engineering*, ICSE '96, pages 71–80, 1996.

[20] R. Majumdar and R.-G. Xu. Directed test generation using symbolic grammars. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 134–143, 2007.

[21] L. J. Morell. A theory of fault-based testing. *IEEE Transactions on Software Engineering*, 16(8):844–857, Aug. 1990.

[22] B. Rosén. Asymptotic normality in a coupon collector's problem. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 13(3-4):256–279, 1969.

[23] D. Rosenblum. A practical approach to programming with assertions. *IEEE Transactions on Software Engineering*, 21(1):19–31, Jan 1995.

[24] R. Sharma, M. Gligoric, A. Arcuri, G. Fraser, and D. Marinov. Testing container classes: Random or systematic? In *Proceedings of the 14th International Conference on Fundamental Approaches to Software Engineering*, FASE'11, pages 262–277, 2011.

[25] M. Staats, G. Gay, M. W. Whalen, and M. P. E. Heimdahl. On the danger of coverage directed test case generation. In *15th International Conference on Fundamental Approaches to Software Engineering*, FASE'12, pages 409–424, 2012.

[26] P. Thévenod-Fosse and H. Waeselynck. An investigation of statistical software testing. *Software Testing, Verification & Reliability*, 1(2):5–25, 1991.

[27] N. Tracey, J. Clark, and K. Mander. Automated program flaw finding using simulated annealing. In *Proceedings of the 1998 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA '98, pages 73–81, 1998.

[28] E. Weyuker and T. Ostrand. Theories of program testing and the application of revealing subdomains. *IEEE Transactions on Software Engineering*, SE-6(3):236–246, May 1980.

[29] E. J. Weyuker. On Testing Non-Testable Programs. *The Computer Journal*, 25(4):465–470, Nov. 1982.

[30] E. J. Weyuker and B. Jeng. Analyzing partition testing strategies. *Transactions on Software Engineering*, 17:703–711, July 1991.

[31] T. Williams, M. Mercer, J. Mucha, and R. Kapur. Code coverage, what does it mean in terms of quality? In *Proceedings of the Reliability and Maintainability Symposium*, pages 420–424, 2001.

[32] Q. Yang, J. J. Li, and D. Weiss. A survey of coverage based testing tools. In *International Workshop on Automation of Software Test*, pages 99–103, 2006.