Formal Analysis of the Probability of Interaction Fault Detection Using Random Testing

Andrea Arcuri, *Member, IEEE* and Lionel Briand, *Fellow, IEEE.*

Abstract—Modern systems are becoming highly configurable to satisfy the varying needs of customers and users. Software product lines are hence becoming a common trend in software development to reduce cost by enabling systematic, large scale reuse. However, high levels of configurability entail new challenges. Some faults might be revealed only if a particular combination of features is selected in the delivered products. But testing all combinations is usually not feasible in practice, due to their extremely large numbers. Combinatorial testing is a technique to generate smaller test suites for which all combinations of t features are guaranteed to be tested. In this paper, we present several theorems describing the probability of random testing to detect interaction faults and compare the results to combinatorial testing when there are no constraints among the features that can be part of a product. For example, random testing becomes even more effective as the number of features increases and converges towards equal effectiveness with combinatorial testing. Given that combinatorial testing entails significant computational overhead in the presence of hundreds or thousands of features, the results suggest that there are realistic scenarios in which random testing may outperform combinatorial testing in large systems. Furthermore, in the common situations where test budgets are constrained and unlike combinatorial testing, random testing can still provide minimum guarantees on the probability of fault detection at any interaction level. However, when constraints are present among features, then random testing can fare arbitrarily worse than combinatorial testing. As a result, in order to have a practical impact, future research should focus on better understanding the decision process to choose between random testing and combinatorial testing, and improve combinatorial testing in the presence of feature constraints.

Index Terms—Combinatorial Testing, Random Testing, Interaction Testing, Theory, Constraint, Feature Diagram, Lower Bound

1 Introduction

Many software testing activities can be automated. Testing comprises a large number of such activities across the development life-cycle. Test automation can therefore significantly reduce time to market and increase system dependability. One significant challenge is that many modern systems are highly configurable, to satisfy wide variability in customers' needs. For example, in software applications running on mobile phones, many features can be configured, such as the type of phone, operating system, and installed applications. Each configuration represents a different product and may exhibit different failure modes. In industrial systems, there are typically millions of possible configurations, where possibly only a small subset of combinations can trigger failures. The question is then how to maximize failure detection when it is not possible to test all configurations. This has been the objective of combinatorial testing.

One way to introduce configurability is through families of software systems. The most known approach to do so is software product lines, which can lead to "drastically increasing the productivity of IT-related industries" [35]. Software product lines have received particular attention from the research community, with dedicated special issues in Communications of the ACM [35] and in IEEE Software [27]. In our context, there are two main challenges: (1) how to represent the variability in an expressive and practical way and (2) how to use such variability description to automate the generation of test cases that are effective in revealing failures. Variability can be expressed with various formalisms, as for example templates in UML models, domain specific languages, and feature diagrams, and has been recently surveyed in [33], in which six types of techniques are described. In this paper we address point (2) above, with a focus on analyzing *Combinatorial Interaction Testing* (CIT) [19], [30] and comparing it with random testing [13], [2].

In CIT, the tester usually has to choose a strength t, which determines the type of interactions to be tested. To increase the probability of finding faults while keeping the cost down, CIT aims at generating a minimal test suite for which all the t-wise combinations of features are present at least once. The underlying assumption is that faults might be due to interactions when t specific features are present in a released product. Generating minimal test suites that guarantee t-wise coverage is a highly difficult problem which has been the subject of a great deal of research [19], [30].

Although there are empirical studies in the literature that show, under some specific conditions, a better performance of CIT strategies compared to random testing (e.g., [32]), CIT has some detractors, such as Bach and Schroder [3]. In their study, when they empirically compared CIT with random testing [13], [2], results showed that random test suites were highly likely to cover most of the t-wise combinations among features. Successive studies from other authors were consistent in showing a similar performance for random testing and CIT at detecting faults (e.g., [37], [14]). From a practical standpoint, if such a result is confirmed, then there may be situations where one may question the use of sophisticated and complex CIT techniques when random testing could be

• *Simula Research Laboratory, P.O. Box 134, Lysaker, Norway. E-mail: {arcuri,briand}@simula.no*

(nearly) as effective at revealing faults. This question is even more pressing given the fact that current CIT tools/techniques [19], [30] display scalability problems to handle large numbers of features, which are common in industrial software. When automated oracles can be generated, random testing could be far more effective if indeed it provides high t-wise coverage. Given the computational overhead entailed by CIT tools to generate covering arrays [19], [30], in the same amount of time, many more test cases could be run and evaluated with random testing when compared to CIT. But to the best of our knowledge, despite the practical importance of this topic, we are aware of no work in the literature investigating and comparing CIT and random testing in a real industrial setting.

Triggering failures is important to detect faults, but then characterizing the feature combinations that induce failures is required to help the debugging process. There can be several techniques to achieve this goal, as for example the classification trees used in [37]. Different techniques to generate test suites would have a different impact on the inferred models. For example, in their empirical study, Yilmaz *et al.* [37] found that test suites generated with random testing often led to unreliable classification trees. Although debugging is an important activity, it is out of the scope of this paper which focuses on the software testing phase.

In this paper, we address the effectiveness and scalability of random testing when used to detect t-wise interaction faults. In particular, we are interested in comparing the effectiveness of tests generated with a covering array tool versus tests generated randomly. We provide formal proofs that are independent from specific CIT problem instances, and hence provide more general results than what experiments would typically yield. For example, we formally prove that random testing, when compared to CIT and assuming an equal test suite size N, has *always*, even in the worst case, a probability of *at least* 63% to trigger at least one failure related to t-wise interaction faults. In other words, with probability equal or greater than 63%, there would be one or more test cases out of N that fail. For larger numbers of features, the probability increases and we formally prove that this probability converges to 100% (for an infinite number of features), thus making random testing a promising testing strategy for large systems. In addition, we show that in practice, in the common situation where one has to deal with test budget constraints, random testing can provide *minimum guarantees* about the detection of faults at any interaction level. This is something which is very difficult to obtain with CIT (we are aware of no formal results on the topic for CIT), as results would depend on the specific algorithms used to construct the covering arrays.

The results presented in this paper are valid only when there are no *constraints* among features, i.e., the choice of a feature is independent from the others, and any combination of features represents a valid product. Though this is the most common case reported in the research literature [19], [30], in many industrial systems such constraints are present and, in recent years, CIT approaches that can handle constraints have been proposed, though very few realistic studies are reported (e.g., [8], [17], [31], [4]). In the presence of feature constraints, we show that random testing can be several orders of magnitude worse than CIT tools that handle constraints, though this remains a topic of investigation in the context of real settings and constraints.

Based on the theoretical results of this paper, we conclude that, in many circumstances involving large numbers of features, CIT can be expected to be cost-effective compared to random testing *only* when constraints are present among features. This suggests that such constrained situations should be the main focus of CIT research, along with more realistic and thorough empirical studies to better support the decision process of choosing between CIT and random testing.

The paper is organized as follows. Section 2 formalizes the testing problem we address in this paper. Section 3 provides a formal analysis of CIT compared to random testing. The theory is extended in Section 4 to consider the cases of multiple faults rather than just considering the triggering of at least one failure. Section 5 discusses the practical implications of such formal analysis and provides an initial set of practical guidelines. Threats to the validity of these theoretical analyses are discussed in Section 6. Finally, Section 7 concludes the paper.

2 Problem Definition

In order to improve the probability of finding faults, CIT aims at generating test suites with high coverage of feature interactions. The "covering array" problem, which is widely studied in the mathematical community [11], can be considered to be a specific strategy to support CIT. In this paper, we use the notation provided in [11]. In particular, "A mixed level covering array MCA$(N;t,k,(v_1,v_2,...,v_k))$ is an $N \times k$ array. Let $\{i_1,\ldots,i_t\} \subseteq \{1,\ldots,k\}$, and consider the subarray of size $N \times t$ obtained by selecting columns $i_1,\ldots,i_t$ of the MCA. There are $\prod_{i=1}^{t} v_i$ distinct t-tuples that could appear as rows, and an MCA requires that each appears at least once. We use the notation CAN$(t,k,(v_1,v_2,...,v_k))$ to denote the smallest N for which such a mixed covering array exists" [11]. In other words, N is the number of test cases, k is the number of features, v_i is the number of values the ith feature can assume, and t is the strength of the array. Without loss of generality, let us consider $v_{i+1} \leq v_i$, i.e., we order the values v_i in descending order. Furthermore, we map each feature to a numerical value in $\{1,\ldots,v_i\}$. For example, if the ith feature is binary, its values will be mapped to $\{1,2\}$.

Once a strength t is chosen, then we should obtain a test suite of size N where N is as small as possible (ideally, N = CAN$(t,k,(v_1,v_2,...,v_k))$) and all the t-wise feature combinations are present. If a fault is revealed when a precise combination of up to t features is selected for a product, then CIT will guarantee to reveal such a fault.

In many cases, there can be constraints among features [8], [17], [31], [4]. For example, a particular feature can be selected only if another specific one is selected as well. There can be several ways to represent this type of constraints, using for example feature diagrams. At a high level, we can see these constraints as first order logic predicates, which determine whether a particular combination of features would represent a valid product. In this context, CIT aims to generate

minimal test suites for which all t-wise feasible combinations of features are present at least once.

3 Comparison with Random Testing

Assume that a CIT tool is used to generate a test suite of size N for a given strength t when no constraint among features is present. The motivation of doing so would be to detect feature interaction faults up to t-wise interactions and at the lowest execution cost. However, how would random testing perform in this context? In other words, what would be the probability that a randomly generated test suite of size N would detect t-wise interaction faults? In practice, because of the overhead incurred by CIT to generate covering arrays, one could even run a number of random test cases that is significantly larger than N within the same time that CIT takes to generate and execute N test cases (as long as an automated oracle is available, e.g., does the program crash?). This question has significant practical implications: what would be the point of using a complicated and computationally expensive technique such as CIT if, in some circumstances, random testing offers a similar performance at a much lower cost?

In this paper, with the term "random testing" we mean the following: sample N test cases at random, where the value of each of the features in the ith column is randomly chosen with uniform probability from $1,\ldots,v_i$. Notice that this procedure could be repeated q times and then select the best test suite, where the best test suite would be the one with highest number of covered t-wise interactions. Based on the computational time we can afford (i.e., the testing budget), we could run random testing with q as high as necessary to obtain a full t-wise coverage. In this paper, however, we only consider the case $q = 1$. Larger values of q would of course lead to higher fault detection. In a comparison with CIT, we could choose a value for q that corresponds to the CIT overhead required to generate covering arrays.

As discussed in the introduction, there exists empirical work that has shown good performance for random testing when tackling combinatorial testing problems (e.g., [3], [37], [14]). However, because the empirical nature of these works, their results are difficult to generalize to all values of t, k, v_i and N. To address this problem, in this paper, we provide general results that are valid for any choice of those parameters. We first provide a lower bound to the probability that random testing triggers at least one failure related to t-wise faults for a given test suite size N. This theorem is then used to prove (1) a high lower bound, that is independent of N, to the effectiveness of random testing compared to CIT and (2) that for a large k the two testing techniques have the same effectiveness. Then, in the next sections, we analyze the probabilities that random testing finds one or more different faults.

In this paper, we mainly deal with *lower bounds* related to some probabilities that describe the dynamics of random testing when applied to find interaction faults. Assume P to be the probability that random testing triggers at least one failure. A probability is always bounded in $[0,1]$. Random testing might or might not trigger a failure when run once. Depending on the problem instances, the probability P could significantly vary. For example, on very faulty software we could have $P \simeq 1$, whereas P could be much lower in cases where only a single feature combination triggers failures. Because before running any large empirical study it would not be possible to know P in advance, then it would be of practical interest to know a lower bound b (i.e., $P \geq b$) that is valid for *any* problem instance.

A trivial lower bound is $b = 0$, but it is practically useless. A useful lower bound should be a bound that is "high" (where what is considered "high" depends on the application context). For many probability problems, there can always exist at least one problem instance for which $P \simeq 0$. Even if for these types of problems for most of the instances P could be high, the fact that at least one instance has very low P would result in a useless lower bound b. In other words, b can be considered a guarantee on the worst case scenario (in our case, a problem instance).

Another important thing to keep in mind regarding lower bounds b is the mathematical approximations used to make the theoretical analyses tractable. For example, $P \geq b$ does not necessarily mean that there exist at least one problem instance for which $P = b$. A lower bound b might not be tight, in the sense that it could be the case that $P \geq b^*$, where $b^* > b$. This is the same concept of tight bounds as in the analysis of algorithm performance [12]. Assume for example the function $f(x) = 2x+1$ representing the performance of an algorithm as a function of some variable x. One could for example prove a lower bound $f(x) = \Omega(\log x)$, which is *correct* but not tight. In this case, a tight asymptotic lower bound would be $f(x) = \Omega(x)$, where a tight asymptotic upper bound would be $f(x) = O(x)$. To prove that a lower bound $P \geq b$ is tight, one would need to find at least one instance for which $P = b$. Unless one proves that a lower bound is tight, then it might be possible that a higher lower bound exists.

The first theorem, regarding random testing applied to CIT problems, that we prove in this paper is:

Theorem 1. *Given a strength t, and a number of features k defined by $(v_1,v_2,...,v_k)$, then a random test suite of size N would have probability P_t of triggering at least one failure related to t-wise interaction faults that is at least:*

$$P_t \geq 1 - \left(1 - \frac{1}{\prod_{i=1}^{t} v_i}\right)^N . \quad (1)$$

Proof:

To prove this theorem, we first need to calculate what is the lowest probability p_f for which a random test case can reveal a t-wise interaction fault. Then, by using the properties of the *geometric distribution* [15], we can analyze the fault detection probability for N test cases.

If there is one t-wise interaction fault, considering a uniform sampling probability, then the probability of detecting it would be lower bounded by the faulty combination that is most difficult to sample. In other words, the lower bound in Inequality 1 corresponds to the situation where only one combination of t features triggers a failure and that combination is the most unlikely to randomly sample. Given $F_f = \{f_1,\ldots,f_t\}$ the

indices of the features for which there exists a set of values that trigger a failure, then a random test case would only cover one of these $\prod_{i \in F_f} v_i$ combinations of t features. Therefore:

$$p_f \geq \frac{1}{\prod_{i \in F_f} v_i} \geq \frac{1}{\prod_{i=1}^{t} v_i} \ .$$

The first inequality above holds since the number of faulty combinations may be higher than one, and the second inequality holds since v_i values are ordered in descending order and therefore the product of the first t v_i is the highest among all possible subsets of size t. A random test case would trigger a failure with probability p_f. It would not trigger a failure with probability $1 - p_f$. N test cases would not trigger any failure with probability $(1 - p_f)^N$. The probability that in N test cases at least one triggers a failure would be equal to 1 minus the probability that none of them triggers any failure. Therefore:

$$P_t = 1 - (1 - p_f)^N \geq 1 - \left(1 - \frac{1}{\prod_{i=1}^{t} v_i}\right)^N \ .$$

□

If we use a combinatorial testing tool to generate a test suite of size N, we can use Inequality 1 from Theorem 1 to provide a lower bound to the effectiveness of random testing applied on the same problem (i.e., N random test cases). Table 1 shows the benchmark that is usually employed to evaluate unconstrained CIT techniques (used in [17]). The notation to represent the features in a CIT problem is as follows: we have a list of y^j, where y is a value v_i and j is the number of features of that type. For example, 3^4 means that there are four features, each one with $v_i = 3$ possible values. The "Best reported N" column in the table is the lowest N reported in the literature for each CIT problem, along with the reference providing it. This is in turn used to compute the "Lower Bound for P" using Inequality 1, which ranges in this benchmark from 0.632 to 0.945. Though we have, however, no guarantee that this range is close to the real range, without the need to run any experiment on that benchmark, we can use Theorem 1 to assess whether, for a specific CIT problem, random testing would have high probability of detecting at least one t-wise fault, if any is present.

It could be argued that existing CIT tools presented in the literature are not optimal, in the sense that they do not guarantee to generate test suites of minimal size, i.e., $N = \text{CAN}(t,k,(v_1,v_2,...,v_k))$. *Maybe* an optimal tool could generate much smaller test suites than the ones reported in the literature (3rd column, Table 1), which *could* lead to much worse results for random testing when compared to an optimal CIT tool. However, the following theorem proves this conjecture wrong by showing that random testing has a relatively high minimum level of effectiveness that is consistent with the results in Table 1 and is independent of N:

Theorem 2. *For* any *strength* t, any *number of features* k *defined by* $(v_1,v_2,...,v_k)$, *a random test suite of size* $N \geq$ *CAN*$(t,k,(v_1,v_2,...,v_k))$ *would have probability* P_t *of triggering at least one failure related to* t*-wise interaction faults that is at least* $P_t > 0.63$.

TABLE 1
Benchmark of CIT problems used in [17]. For each of them, Inequality 1 is used to provide a lower bound to the probability P that random testing would trigger at least one failure related to t-wise interaction faults for same number N of test cases.

t	Features	Best Reported N	Reference	Lower Bound
2	3^4	9	[26]	0.654
2	$5^1 3^8 2^2$	15	[9]	0.645
2	3^{13}	15	[26]	0.829
2	$4^1 3^{39} 2^{35}$	21	[9]	0.839
2	$5^1 4^4 3^{11} 2^5$	21	[9]	0.659
2	$4^{15} 3^{17} 2^{29}$	30	[9]	0.856
2	$6^1 5^1 4^6 3^8 2^3$	30	[9]	0.638
2	$7^1 6^1 5^1 4^5 3^8 2^3$	42	[9]	0.637
2	4^{100}	45	[9]	0.945
2	6^{16}	62	[9]	0.826
2	7^{16}	84	[10]	0.823
2	8^{16}	110	[10]	0.823
2	8^{17}	111	[10]	0.826
2	10^{20}	162	[10]	0.804
3	3^6	33	[5]	0.712
3	4^6	64	[5]	0.635
3	$5^2 4^2 3^2$	100	[9]	0.634
3	5^6	125	[5]	0.634
3	5^7	180	[10]	0.764
3	6^6	258	[10]	0.698
3	$6^6 4^2 2^2$	272	[7]	0.717
3	$10^1 6^2 4^3 3^1$	360	[9]	0.633
3	8^8	512	[5]	0.632
3	7^7	545	[7]	0.796
3	9^9	729	[5]	0.632
3	10^6	1100	[10]	0.667
3	10^{10}	1219	[5]	0.705
3	12^{12}	2190	[5]	0.719
3	14^{14}	3645	[5]	0.735

Proof: To prove this theorem, we first need consider a lower bound for CAN$(t,k,(v_1,v_2,...,v_k))$. A trivial bound is CAN$(t,k,v) \geq v^t$, which in the case of mixed level arrays it would be CAN$(t,k,(v_1,v_2,...,v_k)) \geq \prod_{i=1}^{t} v_i$ [11] (recall the first t v_i are the largest). Then, we consider the following inequality [29], [28]:

$$\left(1 + \frac{w}{x}\right)^x < e^w \ .$$

By using these two properties, from Theorem 1 it follows:

$$\begin{aligned} P_t &\geq 1 - \left(1 - \tfrac{1}{\prod_{i=1}^{t} v_i}\right)^N \\ &\geq 1 - \left(1 + \tfrac{-1}{\prod_{i=1}^{t} v_i}\right)^{\prod_{i=1}^{t} v_i} \\ &\geq 1 - e^{-1} > 0.63 \ . \end{aligned}$$

The first inequality above comes from Theorem 1. The second inequality simply results from replacing N with its lower bound.

□

Since a test suite generated with a CIT tool, if it covers all the t-wise combinations, would always satisfy $N \geq$ CAN$(t,k,(v_1,v_2,...,v_k))$, we can therefore apply Theorem 2 when comparing CIT and random testing. Furthermore, as further evidence, the general lower bound of 63% proven in Theorem 2 always holds for the bounds shown in Table 1.

Another important point to clarify is that Theorem 1 and 2 only provide *lower bounds*. The actual fault detection ability of random testing could be much higher, hence the need to perform empirical studies in realistic settings.

Given that the most important challenge in the software industry lies in the testing of large (sub)systems and components, it is therefore important for test techniques to be scalable. One motivation for random testing is its ability to easily generate large numbers of test cases and therefore exercise large systems with extremely large test suites, assuming there is an automated oracle. One important question in the context of this paper is to determine how scalable is random testing compared to CIT for a large number of features k.

Theorem 2 provides a $P > 0.63$ lower bound to the effectiveness of random testing. What will happen for large numbers of features k? Will the effectiveness of random testing stay close to 0.63 or increase? If the former would be the case, then it could be argued that CIT could be still useful, for example in the context of systems with high dependability requirements. However, virtually all techniques reported in the literature show scalability problems in handling large number of features [19], [30]. This arises from the fact that most CIT techniques rely on very complex computations, in which for example constraint solvers are employed. Interestingly, the following theorem proves that, for a large k, when we compare random testing with CIT using same size N, then P converges to 1, i.e., there is no difference among the fault detection ability of the two techniques.

Theorem 3. *For* any *strength t, for an increasing number of features k defined by $(v_1,v_2,...,v_k)$ with $v_i \leq m$ for some constant m, then random test suites of size $N \geq$ CAN$(t,k,(v_1,v_2,...,v_k))$ would have probability P_t of triggering at least one failure related to t-wise interaction faults that converges to $\lim_{k\to\infty} P_t = 1$.*

Proof:

Apart from the trivial lower bound m^t for the number of required test cases N, as Colbourn states in [11], the only other known lower bound for N is due to Stevens *et al.* [34]. For $t = 2$, they proved that the minimal size of required test cases for a covering array increases in the order of $\log(k)$. Because for lower strength t and cardinality of values v_i we need less test cases N [11], from [34] it follows that CAN$(t,k,(v_1,v_2,...,v_k)) = \Omega(\log(k))$ for $t > 2$ as well. Using Theorem 1, we can hence prove:

$$\lim_{k\to\infty} P_t \geq \lim_{k\to\infty} 1 - \left(1 - \frac{1}{m^t}\right)^{\Omega(\log(k))} = 1 \ .$$

Given Theorem 1, the inequality above holds as m is always above or equal to actual v_i values and N is replaced with its lower bound as a function of k.

□

Notice that Theorem 3 has some relations with a theorem proven in [18] which states that, under the constraint $N \geq \log(k)$ and a fixed v for the number of values each feature can have, then for $k \to \infty$ a random test suite of size N would cover all possible t-wise combinations. When we look at Table 1 with the result of Theorem 3 in mind, we can see that the highest lower bounds are obtained for large numbers of features regardless of size N. For example, the highest lower bound for P (i.e., 0.945) is for 4^{100}, in which there are 100 features, though only $N = 45$ is necessary. On the other hand, in the trivial case case 4^2, although $N = 16$ is nearly a third of 45, we have a lower bound for P equal to 0.632 because only two features are involved.

Notice that Theorem 3 only shows convergence for an infinite number of features k. Though Theorem 3 cannot be applied directly to any realistic scenario, it can be useful for scalability analyses since it provides an "expected trend" in performance, i.e., the larger the system the easier it will be for random testing to perform well. This is a rather non-intuitive result as random testing would be expected to work fine on small toy problems, but then large systems would be expected to warrant more sophisticated and cost-effective testing techniques. Theorem 3 formally proves the opposite.

4 Detection of Multiple Faults

Until this point, we have only analyzed random testing from the point of view of triggering at least one failure. Software testing can only trigger failures and not directly reveal faults. If one has a test suite in which more than one test case fails, then some or all of these failures might or might not be related to the same fault. To distinguish among faults, a software tester has to debug and fix the code, and then re-run the test suite to see if any test case is still failing. Real-world systems typically contain several faults and it would be hence important to study how well random testing fares in revealing a set of combinatorial interaction faults.

Assume that the system under test has z interaction faults, whose level of interaction is at most t. A test suite generated with a CIT tool at strength t would be guaranteed to find all of these faults (i.e., have at least one test case that fails for each one of the z faults). In this section, for same test suite size, we prove lower bounds on how many faults would be found by applying random testing. Let $\mathcal{F}$ be the random variable representing how many faults are found by random testing, then:

Theorem 4. *For* any *strength t,* any *number of features k defined by $(v_1,v_2,...,v_k)$, and z faults detectable at feature interactions not higher than t, a random test suite of size $N \geq$ CAN$(t,k,(v_1,v_2,...,v_k))$ would detect at least 63% of the z faults on average, i.e., $E[\mathcal{F}] > 0.63z$.*

Proof:

For each fault i, let us consider a random variable I_i representing whether that fault has been revealed ($I_i = 1$) or not ($I_i = 0$) when the test suite has been executed. Using the same method applied in the proof of Theorem 2, then we can prove:

$$\begin{aligned} E[I] &= 0P(I = 0) + 1P(I = 1) \\ &= P(I = 1) \\ &\geq 1 - \left(1 - \frac{1}{\prod_{i=1}^{t} v_i}\right)^N \\ &> 0.63 \ , \end{aligned}$$

from which it follows:

$$E[\mathcal{F}] = E[\sum_{i=1}^{z} I_i] = \sum_{i=1}^{z} E[I_i] > 0.63z \ .$$

□

A test practitioner could also be interested to know what would be the probability $\mathcal{A}_z$ of finding *all* z faults with a test suite of size N. For random testing, a lower bound for such a probability is given in the following theorem:

Theorem 5. *Given a strength t, a number of features k defined by $(v_1,v_2,...,v_k)$, and z faults detectable with feature interactions not higher than t, then a random test suite of size N would have probability $\mathcal{A}_z$ of revealing all the z interaction faults that is at least:*

$$\mathcal{A}_z \geq \sum_{j=0}^{z} (-1)^j \binom{z}{j} \Big(1 - \frac{j}{\prod_{i=1}^{t} v_i}\Big)^N \ . \tag{2}$$

Proof:

Given z faults, the worst case for a testing technique is when each test case can reveal at most one fault at the time. This would require to generate and run at least z test cases. On the other hand, if the faulty feature combinations are all dependent on different features, then if $z \times t \leq k$ it could be possible to reveal all the faults with a single test case.

The probability distribution of finding z distinct interaction faults with N test cases is equivalent to the *occupancy problem* [15]. In the occupancy problem, N balls would be randomly assigned to z baskets with uniform probability, i.e., a ball will end up in a specific basket with probability $1/z$. The probability distribution of the number of baskets that have at least one ball can be found in [15].

In our case, z are the faults, and N are the test cases, which might or might not trigger any failure. There are two points that need to be handled to apply the occupancy problem theory to analyze $\mathcal{A}_z$. First, because we deal with a lower bound for $\mathcal{A}_z$, we should consider the worst probability for a random test case to trigger a failure: $1/\prod_{i=1}^{t} v_i$. In this case, the lower bound for the probability of covering a particular fault would be equal for all the faults. For simplicity, let us define $w = \prod_{i=1}^{t} v_i$. A test case would hence reveal a particular fault triggered by one or more combinations with probability at least $1/w$, that is the probability for the combination that is most difficult to sample randomly. Second, since there are many test cases that do not trigger any failure, $1/w < 1/z$ and therefore $w > z$, and so we cannot directly apply the occupancy problem theory (as the sum of these z equal probabilities does not add up to 1). We made a simple extension to the theory of occupancy problem to handle the cases in which the probability of a ball ending in a particular basket is $1/w \leq 1/z$, where w is equal for all the baskets. This could represent the fact that balls might just be discarded without ending in any of the baskets. A complete analysis of this "generalized" occupancy problem is presented in Appendix A. Using this generalized theory, we can finally prove:

$$\begin{aligned}\mathcal{A}_z &\geq \tbinom{z}{0}\Big(1 - \tfrac{0}{w}\Big)^N \textstyle\sum_{j=0}^{z-0} (-1)^j \tbinom{z-0}{j} \big(1 - \tfrac{j}{w-0}\big)^N \\ &= \textstyle\sum_{j=0}^{z} (-1)^j \tbinom{z}{j} \Big(1 - \tfrac{j}{\prod_{i=1}^{t} v_i}\Big)^N \ .\end{aligned}$$

□

The lower bound in Inequality 2 is not easy to understand and interpret without computing illustrative examples. In Table 2, we consider the same case studies from Table 1, where we calculate lower bounds $\mathcal{A}_z$ for $z \in \{2,4,8\}$. In contrast to previous bounds in this paper, the lower bounds coming from Theorem 5 are not particularly useful from a practical standpoint as they are rather low, though what is an acceptable threshold depends on the application context. Recall that having a "low" lower bound does *not* necessarily mean that the actual $\mathcal{A}_z$ is low. In fact, lower bounds might not be tight. Of particular interest is the case of features 4^{100}, in which, if $z = 8$, then it is proven that there is at least a 62% probability of finding all of those eight faults. This relatively high value can be explained by the fact that it is the case with most features (100). If we look back at Table 1, recall that this case was also the one with highest lower bound for triggering at least one failure.

Similarly to Theorem 3, we can prove that for a large number k of features, the probability $\mathcal{A}_z$ of finding all the z faults converges to 1. Formally:

Theorem 6. *For* any *strength t, for increasing number of features k defined by $(v_1,v_2,...,v_k)$ with $v_i \leq m$ for some constant m, and z faults detectable with feature interactions not higher than t, then a random test suite of size $N \geq$ CAN$(t,k,(v_1,v_2,...,v_k))$ would have probability $\mathcal{A}_z$ of revealing all the z interaction faults that converges to $\lim_{k\to\infty} \mathcal{A}_z = 1$.*

Proof:

Using the lower bound in Inequality 2, using the same method as in the proof of Theorem 3, then:

$$\begin{aligned}\lim_{k\to\infty} \mathcal{A}_z &\geq \lim_{k\to\infty} \textstyle\sum_{j=0}^{z} (-1)^j \tbinom{z}{j} \Big(1 - \tfrac{j}{\prod_{i=1}^{t} v_i}\Big)^N \\ &= \lim_{k\to\infty} 1 + \textstyle\sum_{j=1}^{z} (-1)^j \tbinom{z}{j} \Big(1 - \tfrac{j}{\prod_{i=1}^{t} v_i}\Big)^N \\ &\geq 1 + \lim_{k\to\infty} \textstyle\sum_{j=1}^{z} (-1)^j \tbinom{z}{j} \Big(1 - \tfrac{j}{m^t}\Big)^{\Omega(\log k)} \\ &= 1 + \textstyle\sum_{j=1}^{z} (-1)^j \tbinom{z}{j} \cdot 0 \\ &= 1 \ .\end{aligned}$$

□

However, from a practical point of view, Theorem 6 is of less interest compared to the others. Not only it is valid only for $k \to \infty$, but it also assumes z to be constant. For larger systems, one would expect more faults than in a small system. We include Theorem 6 in this paper for sake of completeness, as it can be a starting point for further analyses on the topic (e.g., considering z in scalability analyses as a function $\mathcal{G}$ of k: $z = \mathcal{G}(k)$).

TABLE 2
Benchmark of CIT problems used in [17]. For each of them, we report lower bounds for the probability $\mathcal{A}_z$ of finding all z faults for different values of z.

t	Features	N	Reference	Lower Bound for $\mathcal{A}_2$	Lower Bound for $\mathcal{A}_4$	Lower Bound for $\mathcal{A}_8$
2	3^4	9	[26]	0.411	0.140	0.005
2	$5^1 3^8 2^2$	15	[9]	0.406	0.149	0.013
2	3^{13}	15	[26]	0.681	0.446	0.161
2	$4^1 3^{39} 2^{35}$	21	[9]	0.700	0.478	0.202
2	$5^1 4^4 3^{11} 2^5$	21	[9]	0.428	0.172	0.021
2	$4^{15} 3^{17} 2^{29}$	30	[9]	0.730	0.525	0.257
2	$6^1 5^1 4^6 3^8 2^3$	30	[9]	0.403	0.155	0.019
2	$7^1 6^1 5^1 4^5 3^8 2^3$	42	[9]	0.402	0.156	0.021
2	4^{100}	45	[9]	0.893	0.795	0.626
2	6^{16}	62	[9]	0.680	0.458	0.202
2	7^{16}	84	[10]	0.676	0.454	0.201
2	8^{16}	110	[10]	0.677	0.456	0.203
2	8^{17}	111	[10]	0.681	0.462	0.209
2	10^{20}	162	[10]	0.645	0.415	0.169
3	3^6	33	[5]	0.503	0.245	0.052
3	4^6	64	[5]	0.401	0.157	0.023
3	$5^2 4^2 3^2$	100	[9]	0.401	0.158	0.024
3	5^6	125	[5]	0.400	0.159	0.024
3	5^7	180	[10]	0.584	0.339	0.113
3	6^6	258	[10]	0.487	0.236	0.055
3	$6^6 4^2 2^2$	272	[7]	0.514	0.263	0.068
3	$10^1 6^2 4^3 3^1$	360	[9]	0.400	0.159	0.025
3	8^8	512	[5]	0.400	0.159	0.025
3	7^7	545	[7]	0.634	0.401	0.160
3	9^9	729	[5]	0.400	0.159	0.025
3	10^6	1100	[10]	0.445	0.198	0.039
3	10^{10}	1219	[5]	0.496	0.246	0.060
3	12^{12}	2190	[5]	0.516	0.266	0.071
3	14^{14}	3645	[5]	0.540	0.292	0.085

5 Practical Implications

The theorems proven in this paper can be used to analyze in a new light the applicability of CIT ([6], [36], [23], [25], [24]) in practical contexts, in particular in the presence of a large number of features. One should consider whether the expected improvement in fault detection (if any) given by CIT compensates for its much higher computational cost, due to the generation of covering arrays, when compared to random testing. As we have seen, even in the unlikely worst case, we have proven that random testing is surprisingly effective ($P_t \geq 0.63$) and is likely to be nearly as effective as CIT in detecting t-wise interaction faults for large values of k.

This analysis is particularly relevant when *automated oracles* are available, as for example in model-based testing of industrial embedded systems [1]. In this case, given the same computational resources, random testing could then run many more test cases, which could lead to higher fault detection than CIT for a fixed t when accounting for the detection of faults at higher strength levels. For some real-world industrial problems in which there can be thousands of features, then random testing *might* outperform CIT unless CIT's scalability is improved. However, this all depends on the number of features k, the cost of running a CIT tool, and the computational cost of evaluating the automated oracles and running the test cases. Based on such information, a practitioner could decide whether to use random testing rather than CIT. To the best of our knowledge, we are not aware of any empirical study in the literature in which real-world industrial systems (with real faults) with thousands or even hundreds of features have been analyzed. For example, even if there has been work on relatively large open-source systems [8], those were only in the order of hundreds of features (e.g., the largest system in [8], GCC, has 199 features). Therefore, at the current moment we do not know what is the threshold k^* after which random testing becomes more cost-effective, as it would require an empirical investigation in specific industrial contexts. Our results, however, suggest that such an investigation should be performed in any context where CIT is being considered.

When no automated oracle is available, the test case outputs need to be manually evaluated and the CIT computational time overhead becomes irrelevant when compared to the required manual labor involved. Generating random test cases for as long as CIT takes to select test cases would not be a reasonable option as it would entail to run and evaluate more test cases than CIT. Given a configuration problem, a practitioner could use a CIT tool to generate test suites for different strengths t. For example, Table 3 shows test suite sizes for a 4^{15} problem using the IPOG tool [25], using strengths from $t = 3$ to $t = 6$. On this (small) problem, IPOG is pretty efficient in terms of time (reported time values in [25] range from few milliseconds to half an hour). For example, IPOG can generate a test suite of size 181 that *guarantees* that all 3-wise interactions are covered. Using Theorem 1, we can calculate a lower bound for random testing to trigger at least one failure related to t-wise faults. For $t = 3$, we have a probability P_t that is *at*

TABLE 3
Test suite sizes for CIT problem 4^{15} using the IPOG tool as reported in [25]. For same sizes and t, the table also shows probability lower bounds of random testing to trigger failures related to t and $t+1$ interaction faults.

t	Size	Lower Bound for P_t	Lower Bound for P_{t+1}
3	181	0.942	0.508
4	924	0.973	0.595
5	4519	0.988	0.668
6	20384	0.993	0.712

least 0.94 (see Table 3). Although the true probability could be higher, it can *never* be equal to 1 because random testing is a randomized algorithm. Therefore, under these conditions one could argue that CIT is a better option to find all t-wise faults when automated oracles are missing.

However, to the best of our knowledge, since empirical analyses in the literature do not consider industrial cases in which there could be hundreds or even thousands of parameters to configure (e.g., in [25] the largest problem instance has only 20 features), the cost of CIT could be prohibitive in practice. But, when accounting for the fact that in practice test budgets are limited and often severely constrained, even in the cases in which the CIT overhead is affordable and no automated oracle is available, there are at least two related conditions under which random testing might still be preferred:

1) When information is required about fault detection for interaction faults at a higher strength than the selected t .
2) When there are strict constraints on the testing budget.

Let us go into more details regarding these two conditions. Following the previous example, a CIT suite of 181 test cases would guarantee to find all the faults up to interaction strength 3. But what if the system under test has no fault at any level up to 3, and rather it has faults at level $t = 4$? In practice, a software tester would not know which type of faults are present in the system before performing extensive testing. Furthermore, current CIT tools used at strength t do not give any guarantee on the probability of finding faults at interactions higher than t. If a CIT tool is deterministic (or if it has a bias toward some configuration patterns), then it would be pretty easy to construct problem instances for which it cannot, for example, find some $t+1$ interaction faults. On the other hand, random testing would *always* have a non-zero probability of finding any interaction fault at any strength t. But if such probability is low, then it would be of little practical interest. We used Theorem 1 to compute the lower bound of probability P_{t+1} and Table 3 report probability values ranging from 50% to 71%. Let us consider the case where $t = 6$. Using a tool such IPOG would take half an hour (using the same machine as in [25]), and would result in a test suite fully guaranteeing the finding of 6-wise interaction faults and, to the best of our knowledge, no guarantee regarding 7-wise faults. On the other hand, given the same number of test cases $N = 20384$, random testing would take only few seconds to run, and would have *at least* a 99% probability of triggering at least one failure related to 6-wise faults, and *at least* a 72% *guaranteed* probability of triggering at least one failure related to 7-wise faults.

Notice that, as discussed above, if no automated oracle is available, running and evaluating 20384 test cases would overshadow any computational cost for test data generation. Even if we consider a less demanding criterion, as for example $t = 3$, then running and manually evaluating 181 test cases would still be very expensive in terms of manual labor. However, in this case random testing would have at least a 94% probability of triggering failures related to 3-wise faults, and at least a guaranteed 50% probability of triggering failures related to 4-wise faults. We have no guarantee regarding the latter with CIT.

It could be argued that, if one wants to have a 100% guarantee of finding $t+1$ faults, then the CIT tool should be run with strength $t+1$. Unfortunately, this is not always possible in practical contexts with limited budgets. Because test cases might have to be run on actual hardware or in specialized testing labs, budget limitations can be quite inflexible in practice. This practically entails that a tester cannot, in many circumstances, run as many test cases as ideally needed. For example, this is a case we encountered when applying model-based testing on the video-conference system, for which we had to develop sophisticated techniques to choose subsets of test cases that could be run within the testing budget [20], [21]. Considering the case in Table 3, a testing budget of 400 test cases would be significantly more than what CIT yields with $t = 3$ (181) but much less than what is obtained with $t = 4$ (924). Table 4 shows that, if one uses random testing to choose those 400 test cases, then $P_3 \geq 99\%$ and $P_4 \geq 79\%$ (the table also shows the cases for test suite sizes $2{,}000$ and $10{,}000$, for which we can see similar trends). From a practical standpoint and in this particular case, it could be better to use random testing since the probability of triggering failures related to 3-wise faults is very close to 1 but the probability for 4-wise faults is still rather high, something for which we have no guarantee with CIT at strength $t = 3$. It might further be argued that one could rather extend the test suite generated at $t = 3$ by adding 219 new test cases, or reduce $t = 4$ by removing 524 test cases. However, how increasing or reducing the test suite is actually performed would have strong impact on the resulting probability of finding 4-wise faults. But there might very well be cases in which such an approach could be better than generating 400 test cases directly with random testing. However, even if empirical studies would support such a conjecture, we would still need to prove what could happen in the worst case. Unfortunately, as already mentioned above, current CIT tools do not provide any formally proven guarantee on the probability of finding interaction faults at higher levels than t. On the other hand, using random testing on that example would *guarantee* to have $P_4 \geq 79\%$. In many critical systems, a guarantee of 79% given by random testing would be far better than no guarantee at all.

In case of limited testing budget, say 400 test cases, a CIT tool could be designed to maximize the number of distinct covered configurations (as recently done in [16]). A CIT tool could guarantee a 3-wise coverage and try to maximize the

number of 4-wise combinations in the constructed 400 test case suite. In this case, each 4-wise interaction fault would be either found or not. If one repeats such an experiment n times, and counts the number of experiments a in which at least one 4-wise fault is found, then the probability that a CIT tool finds at least one 4-wise fault (triggering at least one failure related to one 4-wise fault) could be estimated with $P_4(CIT) \simeq a/n$ if n is large enough. If the employed CIT tool is based on a deterministic algorithm, then it would be either $P_4(CIT) = 0$ or $P_4(CIT) = 1$, as the covered configurations would be always the same in all n experiments. On different fault patterns f (i.e., combinations of features that result in triggering a failure), then we would have different $P_t^f(CIT)$. For example, for a fault pattern f' where only one single feature combination triggers failures, we would likely obtain a P that is lower than in the cases of fault patterns where many combinations trigger a failure.

If one considers the space of all possible fault patterns F, it could be possible in principle, based on the previous example, that such a CIT tool would have *on average* a higher P_4 than random testing (RT): $\sum_{f \in F} P_4^f(CIT)/|F| > \sum_{f \in F} P_4^f(RT)/|F|$. However, particularly in the context of critical systems, it might be important to give bounds on the worst case scenario. For example, if the CIT tool is deterministic, then it is trivial to see that there would be several fault patterns for which $P_4^f = 0$. If a CIT tool is based on complex algorithms (as it is usually the case), then it could be difficult (albeit not impossible) to provide non-trivial lower bounds to the worst case scenarios. In fact, it would be necessary to prove a probability lower bound for the combination that is least likely to be sampled.

Using the theorems presented in this paper, we can study the effectiveness of random testing to detect failures at interactions higher than t, for a fixed test suite size N. In this context, a theoretical comparison with CIT techniques is not possible, as it would be tool dependent. However, Kuhn *et al.* [22] report an empirical study that shows one case in which random testing produces better results than CIT (using IPOG). The case study was on detecting deadlocks in a grid computer network. Given the same test suite size N, random testing was better than pairwise testing, but it fared worse than 4-wise testing (if we ignore the cost of generating the test cases). The explanation was that faults were present up to interactions $t = 4$ and pairwise testing does not give any guarantee for finding $t > 2$ faults. In that particular case study, random test suites were more effective at detecting the latter type of faults. On the other hand, 4-wise testing was of course revealing *all* $t = 4$ faults, whereas random testing expectedly could not. In practice, before testing is completed, the types of faults present in the tested system are not known. Even in the ideal case in which one would predict the type of faults likely to be present in the system (e.g., faults are present only for interactions up to $t = 6$), the resulting test suites generated with a CIT tool could be too large too be evaluated (e.g., due to the absence of automated oracles), such that a lower t might need to be chosen. Therefore, it is not possible to claim with 100% certainty that a test suite generated with a CIT tool would be better at detecting faults than a test suite chosen at random. Notice that the work of Kuhn *et al.* [22] was of empirical nature and that, as a result, results might be different on other case studies. However, what is important is that this study shows how and why random testing can outperform CIT in practice.

From a practical standpoint, a test engineer would need to decide whether to employ a CIT tool or to rely on random testing. As we discussed in this paper, this choice is far from trivial, as none of these techniques can be expected to be superior to the other in all testing scenarios and case studies. Nevertheless, it is important to summarize what we have learnt so far and provide some guidelines that would be valid in most cases in order to help practitioners in choosing the most appropriate test technique in context. However, it is important to keep in mind that such guidelines might not hold for all possible cases, as exceptions likely exist, and practitioners should therefore exercise discretion in using them.

- If there are numerous features (e.g., in the order of hundreds or thousands) and the employed CIT tool does not scale (i.e, it takes hours/days before generating a test suite, or it simply runs out of memory and crash), then random testing is a viable alternative.
- If there are automated oracles, then a software tester might want to generate test cases until a first failure is triggered. In this case, especially in cases where the number of features is large and the overhead of the CIT tools is expected to be large, random testing can be a better option as that overhead could rather be spent in running more test cases. Furthermore, there would be no need to specify a specific strength t (which is always a difficult decision), which in the context of constrained resources is also impractical as it entails that test engineers should run a test suite of specific size N that may not be easily accommodated by existing resources in the presence of numerous features.
- If testing budgets are strictly constrained and only a specific number Z of test cases can be run (e.g., there is hardware-in-the-loop testing or a test lab needs to booked in advance for a specific amount of time), then random testing can be used to generate the exact number of test cases that can be run. With a CIT tool, once a strength t is chosen, the number of test cases generated can much lower or higher than Z.
- If dependability requirements are high and strict testing guarantees need to be provided (e.g., when testing safety critical software), then a CIT tool may be a better option as it guarantees to find all faults up to strength level t. However, if only a low strength (e.g., $t = 2$) can practically be used (e.g., the resulting test suite is too large to run or be manually evaluated), then random testing might become a better alternative as it provides lower bounds on the probability of finding faults at higher levels than t, whereas current CIT tools do not. For low strength levels, as demonstrated by existing case studies, random testing may turn out to be better than CIT at finding faults at higher levels than t.

TABLE 4
For CIT problem 4^{15}, three different test suite sizes are considered, with lower bounds of the probability of random testing to trigger failures related to t-wise faults. Note that the probability values 1 are only due to numerical approximations.

Size	Lower Bound for P_3	Lower Bound for P_4	Lower Bound for P_5	Lower Bound for P_6
400	0.998	0.791	0.323	0.093
2000	1.000	1.000	0.858	0.386
10000	1.000	1.000	1.000	0.913

- In all the other cases in which there are no automated oracles, the number of features is not so high as to lead CIT tools to large computational overheads, and there are no strict testing budgets, then CIT is likely to be a better option than random testing.

Our analysis relies on the assumption that there are no constraints among features. However, in the case of constraints, the performance of random testing could be arbitrarily low compared to CIT. To clarify this point, consider the following example: pairwise testing ($t = 2$), k binary features X_i and the constraint $X_1 \wedge X_2 \Rightarrow X_3 \wedge \ldots \wedge X_k$. In other words, if the first two features are selected, then all the others have to be selected. Assume also that failures are triggered only if X_1 and X_2 are both selected. CIT tools such as [8], [17], [31], [4] would not have any particular problem in generating 2-wise test suites (in fact, that constraint is pretty easy to solve with a constraint solver) and hence reveal a fault. On the other hand, a random test case would have probability $P_{X_1 \wedge X_2}$ to generate a valid test case that reveals a fault, that is only $P_{X_1 \wedge X_2} = 1/2^k$, which would be extremely low even with a small number of features k. But in the general case, whether constraint solvers can be effective for this task still remains to be empirically investigated in realistic settings, with large numbers of real constraints.

6 Threats to Validity

Threats to *internal validity* for this work come from the fact that mathematical analyses can be subject to errors. To reduce the probability of this being the case, we have also carried out simulations to verify them (whose details are not reported in this paper as they do not add any valuable information). This would be equivalent, in empirical studies, to "test" the software tools (e.g., using JUnit for software written in Java) that have been used to carry out the experiments. However, simulations cannot be used to prove that a proof is error-free though they are useful as supportive evidence.

Threats to the *external validity* of formal analyses depend on the assumptions that are made. Depending on the validity of these assumptions, the application of the theoretical results to real-world scenarios could be compromised. In our theoretical analyses, we have proved lower bounds to some probabilities regarding the application of random testing applied to CIT problems. We have not made any explicit assumptions in these theorems. Some theorems, however, cannot be directly applied in practical contexts, as they consider the theoretical case of an infinite number of features (Theorem 3 and Theorem 6). Those theorems are anyway valuable in scalability analyses, when for example varying the number of features, to understand and explain trends when comparing performance of CIT and random testing. One possible threat to external validity is that we have only formally analyzed unconstrained CIT problems. Although most CIT publications are on this subject [19], [30], most CIT problems in industry could be of the constrained type. But the research literature currently does not contain enough empirical evidence to either accept or reject such an hypothesis.

7 Conclusion

In this paper we have formally proven six theorems regarding the effectiveness of random testing when applied to unconstrained combinatorial interaction testing (CIT) problems. These theorems provide general results that cannot be obtained with empirical studies, though the latter are necessary to refine our understanding. First, we proved that for any t-wise CIT problem (independently of its properties), random testing would *always* have *at least* a 63% probability of triggering at least one failure related to t-wise interaction faults (if any is present) when compared against *any* CIT tool using the same number of test cases. Second, perhaps more importantly, this probability increases with the number of features present in the software under test and converges to 1 (for infinite number of features), that is to equal effectiveness with CIT techniques. Given that current CIT tools suffer severe scalability problems in the presence of large numbers of features, thus leading to significant execution overheads to generate covering arrays, this additional time could be easily used by random testing to run more test cases if an automated oracle is available. Our results suggest that in such situations random testing would be effective at detecting interaction faults.

When no automated oracle is available, we have identified realistic situations in which random testing might still be preferred: when there are strict constraints on the testing budget and it is necessary to provide guarantees on the probability of finding faults at interaction levels higher than the chosen t. We have shown that such guarantees, which are not provided with current CIT techniques, can actually be obtained with random testing.

Since one cannot assume that in practice CIT techniques are necessarily better than random testing, our work motivates the need for more empirical studies in realistic contexts to compare the two approaches. The main goal should be to refine the decision guidelines we provide in this paper in order to help testers select, in practical contexts, between random testing and CIT techniques.

The theoretical results presented in this paper are only valid in the absence of constraint among features. When there are constraints (as it often happens in industrial systems), we have shown that random testing might become very inefficient. Therefore, the results of this paper should also be seen as a way to motivate further research in CIT for the cases in which feature constraints are present. These types of applications are not so common in the literature [19], [30], but they have started to get more attention in recent years (e.g., [8], [17], [31], [4]).

In the future, it will be important to study whether it is possible to provide lower and upper bounds to the effectiveness of random testing applied to constrained CIT problems. Such results would help to quantify how much improvement a CIT technique could achieve compared to simple and practical testing techniques such as random testing.

ACKNOWLEDGMENTS

The work described in this paper was supported by the Norwegian Research Council. This paper was produced as part of the ITEA-2 project called VERDE.

APPENDIX

In this appendix, we provide a simple extension to the occupancy problem [15], in which we consider the case when balls end up in a specific basket with probability $1/w \leq 1/z$ rather than with probabilty $1/z$. Notice that, due to the simplicity of this extension, and the wide applications of probability analysis, it is likely that such generalized version has already been studied in the literature (although we have not found any reference). The formal proof in this appendix has hence been included in this paper to make it self-contained. Let Y be the random variable representing the number of baskets that are empty after sampling N balls. The following theorem provides the probability mass function for Y:

Theorem 7. *In the generalized occupancy problem where there are z baskets, N balls and each of them can end up in a specific basket with probability $1/w \leq 1/z$, then the mass function of the number of empty baskets Y is described by:*

$$P(Y=y) = \binom{z}{y}\left(1-\frac{y}{w}\right)^N \sum_{i=0}^{z-y}(-1)^i \binom{z-y}{i}\left(1-\frac{i}{w-y}\right)^N .$$

Proof:

The proof of this theorem follows the same structure of the proof in [15] for the standard occupancy problem. $P(Y=y)$ can be described as the probability that y baskets are empty ($A=y$) and, given that event, the remaining $z-y$ basket are occupied ($B=z-y|A=y$), i.e. $P(Y=y) = P(A=y) \times P(B=z-y|A=y)$. A set of y baskets would remain empty with probability $(1-y/w)^N$, and there are $\binom{z}{y}$ ways to choose a sub-set of y baskets from z of them. Therefore, $P(A=y) = \binom{z}{y}\left(1-\frac{y}{w}\right)^N$. Calculating $P(B=z-y|A=y)$ is lengthy, and requires quite a few extra intermediary steps.

Let us consider V_i representing whether a specific basket with index i is empty (after sampling N balls) given the fact that y other baskets are empty. If we know that y other baskets are empty, then the conditional probability p_i of sampling a ball that ends in basket i would be higher than $1/w$, as we know that it cannot end up in any of these y baskets. In particular, we would have:

$$p_i = \frac{1}{w} \times \frac{1}{1-\frac{y}{w}} = \frac{1}{w-y} .$$

Therefore:

$$P(V_i) = (1-p_i)^N = \left(1-\frac{1}{w-y}\right)^N ,$$

where the probability that m baskets are all empty at the same time would be:

$$P(V_1 \wedge \ldots \wedge V_m) = \left(1-\frac{m}{w-y}\right)^N .$$

Now, we can use the *inclusion-exclusion principle* [15] to calculate the probability that at least one of these m baskets remain empty:

$$\begin{aligned} P(V_1 \vee \ldots \vee V_m) &= \textstyle\sum_{i=1}^{m}(-1)^{i+1}\sum_{J\subseteq\{1,\ldots,m\};|J|=i} \\ &\quad P(V_{J(1)} \wedge \ldots \wedge V_{J(i)}) \\ &= \textstyle\sum_{i=1}^{m}(-1)^{i+1}\binom{m}{i}\left(1-\frac{i}{w-y}\right)^N . \end{aligned}$$

The probability that $z-y$ baskets are occupied is equal to 1 minus the probability that any of them is not empty, formally:

$$\begin{aligned} P(B=z-y|A=y) &= 1 - P(V_1 \vee \ldots \vee V_{z-y}) \\ &= 1 - \textstyle\sum_{i=1}^{z-y}(-1)^{i+1}\binom{z-y}{i} \\ &\quad \left(1-\frac{i}{w-y}\right)^N \\ &= \textstyle\sum_{i=0}^{z-y}(-1)^{i}\binom{z-y}{i}\left(1-\frac{i}{w-y}\right)^N . \end{aligned}$$

Finally, we can conclude the proof with:

$$\begin{aligned} P(Y=y) &= P(A=y) \times P(B=z-y|A=y) \\ &= \binom{z}{y}\left(1-\frac{y}{w}\right)^N \textstyle\sum_{i=0}^{z-y}(-1)^{i}\binom{z-y}{i} \\ &\quad \left(1-\frac{i}{w-y}\right)^N . \end{aligned}$$

□

REFERENCES

[1] A. Arcuri, M. Z. Iqbal, and L. Briand. Black-box system testing of real-time embedded systems using random and search-based testing. In *IFIP International Conference on Testing Software and Systems (ICTSS)*, pages 95–110, 2010.

[2] A. Arcuri, M. Z. Iqbal, and L. Briand. Formal analysis of the effectiveness and predictability of random testing. In *ACM International Symposium on Software Testing and Analysis (ISSTA)*, pages 219–229, 2010.

[3] J. Bach and P. Schroeder. Pairwise testing: A best practice that isn't. In *Proceedings of 22nd Pacific Northwest Software Quality Conference*, pages 180–196, 2004.

[4] A. Calvagna and A. Gargantini. A Formal Logic Approach to Constrained Combinatorial Testing. *Journal of Automated Reasoning*, pages 1–28, 2010.

[5] M. Chateauneuf and D. Kreher. On the state of strength-three covering arrays. *Journal of Combinatorial Designs*, 10(4):217–238, 2002.

[6] D. Cohen, S. Dalal, J. Parelius, and G. Patton. The combinatorial design approach to automatic test generation. *Software, IEEE*, 13(5):83–88, 1996.

[7] M. Cohen, C. Colbourn, and A. Ling. Augmenting simulated annealing to build interaction test suites. In *International Symposium on Software Reliability Engineering*, pages 394–405, 2003.

[8] M. Cohen, M. Dwyer, and J. Shi. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Transactions on Software Engineering (TSE)*, 34(5):633–650, 2008.

[9] M. Cohen, P. Gibbons, W. Mugridge, and C. Colbourn. Constructing test suites for interaction testing. In *ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 38–48, 2003.

[10] C. Colbourn. Covering array tables. Available at http://www.public.asu.edu/∼ccolbou/src/tabby/catable.html. Accessed 28 September 2009.

[11] C. Colbourn. Combinatorial aspects of covering arrays. *Le Matematiche*, 58:121–167, 2004.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2001.

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

[14] M. Ellims, D. Ince, and M. Petre. The Effectiveness of T-Way Test Data Generation. *Computer Safety, Reliability, and Security*, pages 16–29, 2008.

[15] W. Feller. *An Introduction to Probability Theory and Its Applications, Vol. 1*. Wiley, 3 edition, 1968.

[16] S. Fouché, M. Cohen, and A. Porter. Incremental covering array failure characterization in large configuration spaces. In *ACM International Symposium on Software Testing and Analysis (ISSTA)*, pages 177–188, 2009.

[17] B. Garvin, M. Cohen, and M. Dwyer. Evaluating improvements to a meta-heuristic search for constrained interaction testing. *Empirical Software Engineering*, pages 1–42, 2010.

[18] A. Godbole, D. Skipper, and R. Sunley. t-covering arrays: upper bounds and Poisson approximations. *Combinatorics, Probability and Computing*, 5(02):105–117, 1996.

[19] M. Grindal, J. Offutt, and S. Andler. Combination testing strategies: A survey. *Software Testing, Verification and Reliability (STVR)*, 15(3):167–199, 2005.

[20] H. Hemmati, A. Arcuri, and L. Briand. Reducing the cost of model-based testing through test case diversity. In *IFIP International Conference on Testing Software and Systems (ICTSS)*, pages 63–78, 2010.

[21] H. Hemmati, A. Arcuri, and L. Briand. Empirical investigation of the effects of test suite properties on similarity-based test case selection. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 327–336, 2011.

[22] D. Kuhn, R. Kacker, and Y. Lei. Random vs. combinatorial methods for discrete event simulation of a grid computer network. *Proceedings of the Mod Sim World, NASA CP-2010-216205, National Aeronautics and Space Administration*, pages 14–17, 2009.

[23] D. Kuhn, D. Wallace, and A. Gallo Jr. Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering (TSE)*, 30(6):418–421, 2004.

[24] R. Kuhn, Y. Lei, and R. Kacker. Practical combinatorial testing: Beyond pairwise. *IT Professional*, pages 19–23, 2008.

[25] Y. Lei, R. Kacker, D. Kuhn, V. Okun, and J. Lawrence. IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing. *Software Testing, Verification and Reliability (STVR)*, 18(3):125–148, 2008.

[26] Y. Lei and K. Tai. In-parameter-order: a test generation strategy for pairwise testing. In *High-Assurance Systems Engineering Symposium*, pages 254–261, 1998.

[27] J. McGregor, D. Muthig, K. Yoshimura, and P. Jensen. Guest Editors' Introduction: Successful Software Product Line Practices. *Software, IEEE*, 27(3):16–21, 2010.

[28] D. Mitrinović and P. Vasić. *Analytic inequalities*. Springer, 1970.

[29] M. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[30] C. Nie and H. Leung. A survey of combinatorial testing. *ACM Computing Surveys*, 43(2):1–29, 2011. Article 11.

[31] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. Le Traon. Automated and scalable t-wise test case generation strategies for software product lines. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 459–468, 2010.

[32] A. Pretschner, T. Mouelhi, and Y. Le Traon. Model-based tests for access control policies. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 338–347, 2008.

[33] M. Sinnema and S. Deelstra. Classifying variability modeling techniques. *Information and Software Technology (IST)*, 49(7):717–739, 2007.

[34] B. Stevens, L. Moura, and E. Mendelsohn. Lower bounds for transversal covers. *Designs, codes and cryptography*, 15(3):279–299, 1998.

[35] V. Sugumaran, S. Park, and K. C. Kang. Introduction. *Communications of the ACM*, 49(12):28–32, 2006.

[36] K. Tai and Y. Lei. A test generation strategy for pairwise testing. *IEEE Transactions on Software Engineering (TSE)*, 28(1):109–111, 2002.

[37] C. Yilmaz, M. Cohen, and A. Porter. Covering arrays for efficient fault characterization in complex configuration spaces. *IEEE Transactions on Software Engineering (TSE)*, pages 20–34, 2006.

Andrea Arcuri received a BSc and a MSc degree in computer science from the University of Pisa, Italy, in 2004 and 2006, respectively. He received a PhD in computer science from the University of Birmingham, England, in 2009. Since then, he has been a research scientist at Simula Research Laboratory, Norway. His research interests include search based software testing and analyses of randomized algorithms.

Lionel Briand is a professor of software engineering at the Simula Research Laboratory and University of Oslo, leading the project on software verification and validation. Before that, he was on the faculty of the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, where he was a full professor and held the Canada Research Chair in Software Quality Engineering. He has also been the Software Quality Engineering Department head at the Fraunhofer Institute for Experimental Software Engineering, Germany, and worked as a research scientist for the Software Engineering Laboratory, a consortium of the NASA Goddard Space Flight Center, CSC, and the University of Maryland. He has been on the program, steering, or organization committees of many international IEEE and ACM conferences. He is the editor-in-chief of the Empirical Software Engineering (Springer) and is a member of the editorial boards of Systems and Software Modeling (Springer) and Software Testing, Verification, and Reliability (Wiley). He was on the board of the IEEE Transactions on Software Engineering from 2000 to 2004. His research interests include: model-driven development, testing and quality assurance, and empirical software engineering. He is a fellow of the IEEE.