# Finding Interaction Faults Adaptively using Distance-Based Strategies

Renée C. Bryce
Computer Science
Utah State University
Logan, UT 84322-4019
Renee.Bryce@usu.edu

Charles J. Colbourn
Computer Science
Arizona State University
Tempe, Arizona 85287-8809
colbourn@asu.edu

D. Richard Kuhn
National Institute of
Standards and Technology
Gaithersburg, MD 20899
kuhn@nist.gov

***Abstract*—Software systems are typically large and exhaustive testing of all possible input parameters is usually not feasible. Testers select tests that they anticipate may catch faults, yet many unanticipated faults may be overlooked. This work complements current testing methodologies by adaptively dispensing one-test-at-a-time, where each test is as "distant" as possible from previous tests. Two types of distance measures are explored: (1) distance defined in relation to combinations of parameter-values not previously tested together and (2) distance computed as the maximum minimal Hamming distance from previous tests. Experiments compare the effectiveness of these two types of distance-based tests and random tests. Experiments include simulations, as well as examination of instrumented data from an actual system, the Traffic Collision Avoidance System (TCAS). Results demonstrate that the two instantiations of distance-based tests often find more faults sooner and in fewer tests than randomly generated tests.**

***Keywords*-Combinatorial testing, distance-based testing, Hamming distance, software testing**

## I. Introduction

NIST last estimated the annual cost of software defects as approximately $59 billion [1]. They also suggest that approximately $22 billion can be saved through more effective testing. Testers need to be more thorough in testing, yet they need to perform testing within a prescribed budget. Systematic approaches of combination testing have been suggested to complement current testing methods in order to improve rates of fault detection (see [2] and therein).

Category partitioning is a base of systematic approaches as finite values (options) for parameters are identified for testing [3]. Each of the finite parameter-values may be tested at least once, in specified combination together, or in exhaustive combination. The simplest and least-thorough combination testing approach is to test all values at least once. The most thorough is to exhaustively test all parameter-value combinations. However, exhaustive testing of all possible combinations is too expensive for most systems. Testers may place constraints to limit tests from category partitioning, however, this can be an unsatisfactory solution when constraints are arbitrarily selected to limit the number of tests [2]. Combination strategies may be a better solution to limiting tests as they systematically test combinations of parameter-values across a system.

Combination testing has been applied with ad-hoc methods, stochastic models, and combinatorial designs. Ad-hoc methods include tests developed by hand in which testers attempt to create representative tests to catch problems that they anticipate. Anti-random testing attempts to provide tests that minimize overlap using cartesian products or Hamming distance [4]. An example of a stochastic model includes Markov chains to simulate usage [5]. Combinatorial designs have been applied to test $t$-way interactions of parameter-values [6], [7], [8], [9], [10], [11], [12], [13]. Indeed there are a number of combination strategies (see [2] for a survey).

Combination strategies that are based on user profiles (such as those using Markov chains) emphasize a bias towards trying to predict user behaviors and locating the most frequently encountered faults first. On the other hand, many combination strategies based on combinatorial designs systematically examine systems without attempting to cover the most frequent usage scenarios. Their goal is to find all of the faults, not just those most frequently encountered by users. We identify these two differences because the combination strategies clearly focus on different goals. We focus on the second case of a less biased testing strategy. (We later compare our combination strategies that are not biased by user profiling to random testing that is also unbiased towards usage modeling but does not systematically cover a system.)

The success of combination strategies hinge upon correct identification of parameters and their suitable values for testing. Indeed, if parameters are missing, or category partitioning does not select suitable values for parameters, then any combination strategy may fail. We identify this threat early on since our work is an extension of combination testing work. In the remainder of this paper, we use the assumption that parameters and values have been correctly identified for testing. Further, we assume that our new distance-based testing methodology is applied to systems in which faults arise from interactions of few parameter-value combinations [10], [11]. We make no assumptions about the distribution of the faults.

The work that we present here does not propose a static combination strategy in which a tester necessarily runs an

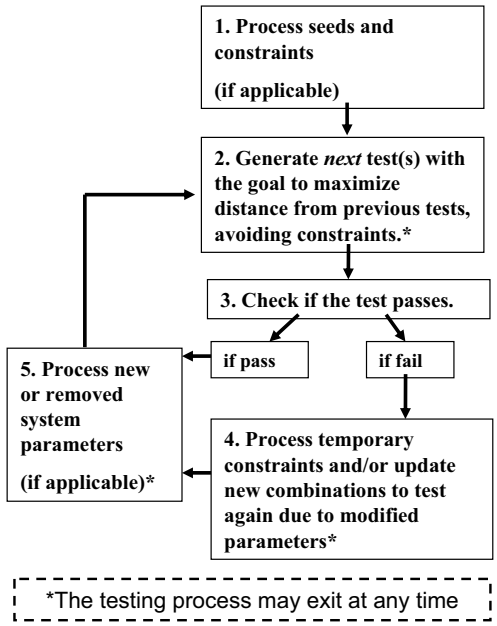



Figure 1. An adaptive distance-based testing process.

entire test suite. Instead, we adaptively suggest a "next test" that maximizes the distance of parameter-value combinations from previous tests while maintaining flexibility for changing systems. In §2, we describe the testing methodology at a high level and introduce specific instantiations of distance. In §3, we describe an algorithm that generates distance-based tests. In §4, an empirical study compares rates of fault detection using distance-based tests in simulations and on instrumented data from an actual system.

## II. Testing methodology

The input to the distance-based testing process is a list of system parameters and their associated values (options) for testing. Previously run tests can also be specified when available so that future tests are built around such tests. Figure 1 shows five steps of the testing methodology. The testing process may halt at any step. Indeed, this is likely as testers can not typically run an exhaustive set of tests that include all possible parameter-value combinations.

**1. Process seeds and constraints.** Tests that have already been run are processed as *seeds* so that the next tests are generated to maximize distance from previously tested parameter-values. Seeds are important because tests should be as distant as possible from previously run tests. *Constraints*, combinations of parameter-values that are not valid to test together, can also be specified when applicable.

**2. Generate next test(s) that maximize distance.** The distance-based testing strategy systematically generate tests in a dynamic environment with tests that are as distant as possible from previous tests. There are multiple ways to define distance (of how different tests are). Two possible distance measures are explored in this paper.

First, distance is defined as the number of new $t$-way parameter-value combinations in a test that have not previously been tested. This is similar to previous work to generate software interaction test suites [14], [15], [16], [17], [18], [19], [20], [21]. A major difference here is that tests are adaptive. While all $n$-way combinations can ultimately be covered, the goal is not to minimize the size of a static interaction test suite. Instead, the focus is to generate a "next" test that covers as many new combinations of parameter-values that have yet to be covered. If the tester stops testing at any time, tests dispensed will have been as broad in variety as possible (in relation to combinations not previously tested). In addition, since we generate tests one-test-at-a-time, we can generate a next test adaptively when system components are added, removed, modified, or temporarily unavailable. This is opposed to throwing away a complete test suite and taking time to regenerate a new one when systems change.

Second, we define distance as the maximum minimal Hamming distance from previous tests. This is similar to previous work on anti-random testing that generates static test suites [4]. However, here we *adaptively* generate one-test-at-a-time and adapt as parameter-values are added, modified, removed, or temporarily unavailable.

**3. Evaluate pass/fail status of test.** If a test passes, the process continues to Step 5. If the test fails, it is taken off-line to be evaluated in Step 4.

**4. Process temporary constraints.** While a test is taken off-line, a tester may specify temporary constraints if they do not want a next test to include the parameter-values in the test that they have taken off-line for closer evaluation. Once they identify the problem(s), constraints may change again. If parameters are modified in an effort to fix a problem, a tester may specify that combinations involving the modified parameter-values be retested. For instance, Table II shows the parameters and values for our TCAS example. If the code related to "High_confidence" is temporarily removed, there may be a constraint that our next test cases will not include this parameter. Alternatively, if we add a new parameter called "Confidence_level" that has values H, M, or L, our next tests would include these.

**5. Process new or removed system parameters** During the testing process, systems may change. New features may be added, removed, or modified at any time. Subsequent tests adapt to these scenarios.

This process differs from previous methods which use $t$-way coverage and maximum distance as surrogates for the ability to find faulty interactions. One significant difference with our approach is that the metric for quality of the tests is not their ability to cover $t$-way interactions; rather it is their ability to isolate faulty interactions quickly. Therefore, experiments in Section §4 focus on the rate of fault detection.

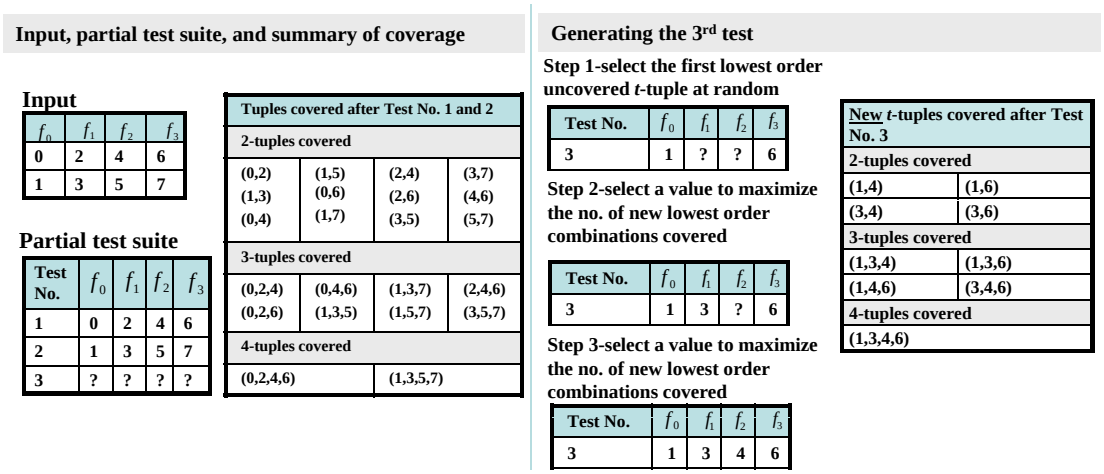



Figure 2. Uncovered combinations distance example

```
start with empty test suite
while possible tests remain do
   process constraints and seeds
   for CandidateCount candidate rows
      initialize new test with all parameters not fixed
      begin by selecting and assigning a
         parameter-value t-tuple of [maximum distance]
      while a parameter remains whose value is not fixed
         select such a parameter f at random,
         select a value ℓ for f with [maximum distance]
            using a value tie-breaking rule
         fix parameter f to level ℓ
   end for
   dispense the candidate row that covers the largest
      distance
end while
```

Figure 3. Pseudocode to generate distance-based tests.

## III. Algorithms

A test consists of assignments to $k$ parameters in a system where each parameter can take on one of $v$ values. The algorithms that we implement generate distance-based tests one-test-at-a-time with a greedy approach (shown at a high-level in Figure 3). First, the algorithm processes seeds and constraints. For each test, the algorithm generates a number of $CandidateCount$ candidate tests and dispenses the candidate that is most *distant*; here we use one candidate. For each candidate test under construction, each parameter needs to be assigned a value. The algorithm begins by selecting a $t$-tuple of parameter-values with maximum distance. For remaining parameters, they may be assigned values in any order at all; here we assign values to parameters in random order. To assign a value to each parameter, the algorithm selects the value that optimizes the *distance* from previous tests. We describe two possible measures of distance next.

### A. Uncovered combinations distance

Uncovered combinations distance is computed as the number of $t$-way parameter-value combinations in a test that have not been included in previous tests. They may also include combinations that a tester specifies to be covered again, perhaps due to modifications to the system under test. In this instantiation, we give preference to lowest order remaining combinations. For instance, we give preference to selections that cover the largest number of untested pairs of parameter-value combinations, followed by preference for selections that cover the largest number of 3-way combinations, up to n-way parameter-value combinations. Consider the example in Figure 2, with input $2^4$ (read as 4 parameters have 2 possible values each) and two tests that have already been generated. The two tests cover twelve 2-way, eight 3-way, and two 4-way combinations.

Assume that the first two tests already exist and we are generating the $3^{rd}$ test shown in Figure 2. To construct a next test, we begin by randomly selecting a 2-tuple that has not previously been included in a test. The pair (1,6) is selected (and labeled as "Step 1" in Figure 2). Random parameter ordering is used for the remaining parameters with order: $f_1$, followed by $f_2$. Values are assigned to parameters by selecting one that covers the largest number of remaining lowest order combinations (2-tuples in this example). Secondary preference is given to covering the

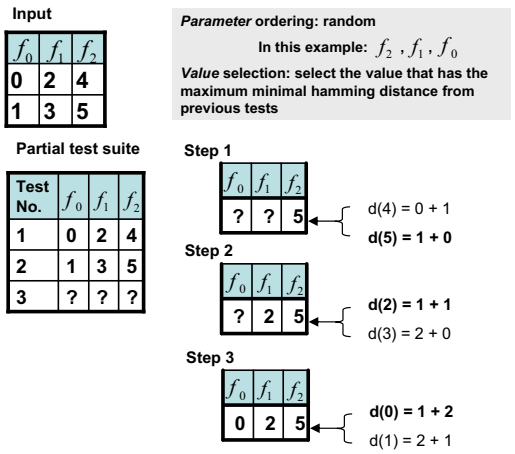



Figure 4. Maximum minimal Hamming distance example.

next lowest order combinations (3-tuples in this example). In the case of ties beyond covering combinations, random tie-breaking is used. For $f_1$, values 2 or 3 may be selected. Selecting either value results in a tie of one new 2-tuple being covered. A second tie is encountered that one new 3-tuple will be covered when either value is selected. The tie is broken at random and we select 3. Finally, a value for $f_2$ is selected. Selecting the value 4 covers two new pairs, while the value 5 covers only one new pair, so the value 4 is selected.

### B. Maximum minimal Hamming distance

Maximum minimal Hamming distance is calculated on a test-by-test basis as the number of different values between each of the previous tests and the test under construction. More specifically, *Hamming distance* between tests is the number of parameters in the tests for which corresponding values are different [22]. *Minimal Hamming distance* is the minimal number of parameters for which corresponding values are different among a set of tests. The *maximum minimal Hamming distance* is then the maximum of the minimal Hamming distances. Figure 4 provides an example.

To construct a test, each parameter is assigned a value based on the maximum minimal Hamming difference among the tests. In the case of a tie, the total Hamming distance is used, followed secondarily by random tie-breaking. We only permit selection of values if doing so results in a partial tuple of a valid remaining test.

Consider the example for an input $2^3$ (3 parameters have 2 values each) shown in Figure 4. We randomly select each parameter one-at-a-time to assign values. In this example, we use the random ordering: $f_2$, $f_1$, $f_0$. We review the assignment of values to each of these parameters and label them as Steps 1, 2, and 3 in Figure 4. Parameter $f_2$ has two values to select from: 4 and 5. Value 4 occurs in the first test and therefore is a distance of 0 from the first test; but does not appear in the second test, so the distance is one. Value 5 has no value in common with the first test, but has one in common with the second. Since both have the same maximum minimum Hamming distance we attempt to break the tie with the total Hamming distance of each. However, there is a second tie as both have a total Hamming distance of 1. Therefore, we use random tie breaking and select 5. Next, a value for $f_1$ is selected by calculating the distance between the partially generated test and the two previous tests. There are two values for $f_1$: 2 and 3. If 2 is selected (while $f_2$=5 from the previous step), there is a difference of one between the partially generated test and test one; and also a difference of one between the partial test and the second test. The selection of the value 3 results in two different values than the first test, but no different values than the second test. Value 2 is selected as it results in a minimal distance of 1 which is larger than value 2's minimal distance of 0. Finally, a value of either 0 or 1 is selected for parameter $f_0$. Selecting value 0 results in a distance of 1 from the first test and a distance of 2 from the second test. Selecting value 1 results in a distance of 2 from the first test and a distance of 1 from the second test. There is a tie for the maximum minimal Hamming distance. A primary tie break of selecting the value with the total maximum Hamming distance results in a second tie. The tie is then broken randomly and value 0 is selected.

### C. Random

Random selection simply chooses a test at random that contains combinations that have not yet been tested. The random testing does not make any assumptions about the system under test. Tests are generated uniformly at random. We do not execute a test configuration more than once. We include random testing here to evaluate whether random testing may be as effective as our systematic tests.

## IV. Empirical studies

In many realistic testing environments, testers may not have time to run entire test suites that cover all possible parameter-value combinations, or even all lower strength $t$-way parameter-value combinations. They may also be in an environment in which partial test suites are run and then testing needs to adapt. Therefore, the process here changes the efficacy of "testing interactions" and measures fault detection directly by assessing the times at which faults are detected rather than waiting until the end of testing to analyze all results.

We compare the rate of fault detection for the two distance-based testing implementations and random testing. While a large family of empirical studies is indeed necessary to assess the value of the approaches on different types of systems, we provide a preliminary measurement of whether they can be useful and examine whether distance-based testing warrants future study. Two experiments simulate systems

with randomly generated faults. Each of the simulations is run five times to report the average of the results. A third experiment examines distance-based testing applied to 41 versions of a Traffic Collision Avoidance System (TCAS) module developed at Siemens Corporation that has been used in previous testing studies. We introduce each of these studies next.

### A. Proof-of-concept Study 1: Simulations

In the following set of experiments, we model two systems with different numbers of parameters, values, and faults. A number of 2-way through 4-way faults are generated at random for each simulation. We compare the two different distance-based testing instantiations and the random testing by rates of fault detection. We also measure the overlap in the number of 2-way, 3-way, and 4-way combinations covered in the respective tests.

We break the studies into two experiments with two goals in mind. First, combinatorial testing has been criticized as an ineffective testing method that offers little benefit over random tests [23], [24]. However, this criticism [23], [24] is supported only in small study and contradicts other existing literature that reports on the success of combinatorial testing (see [17] and therein). Another study [25], [26] reported better performance for combinatorial testing and suggested that its advantage over random testing increases with the number of values per variable and interaction strength. The high proportion of binary variables in [23] may thus explain its finding no advantage for combinatorial methods. Nonetheless, in the first experiment, we examine the points of criticism, discuss scalability of these points, and show an example that supports that combination testing can indeed be useful, especially when higher strength combinations are inclusive of testing.

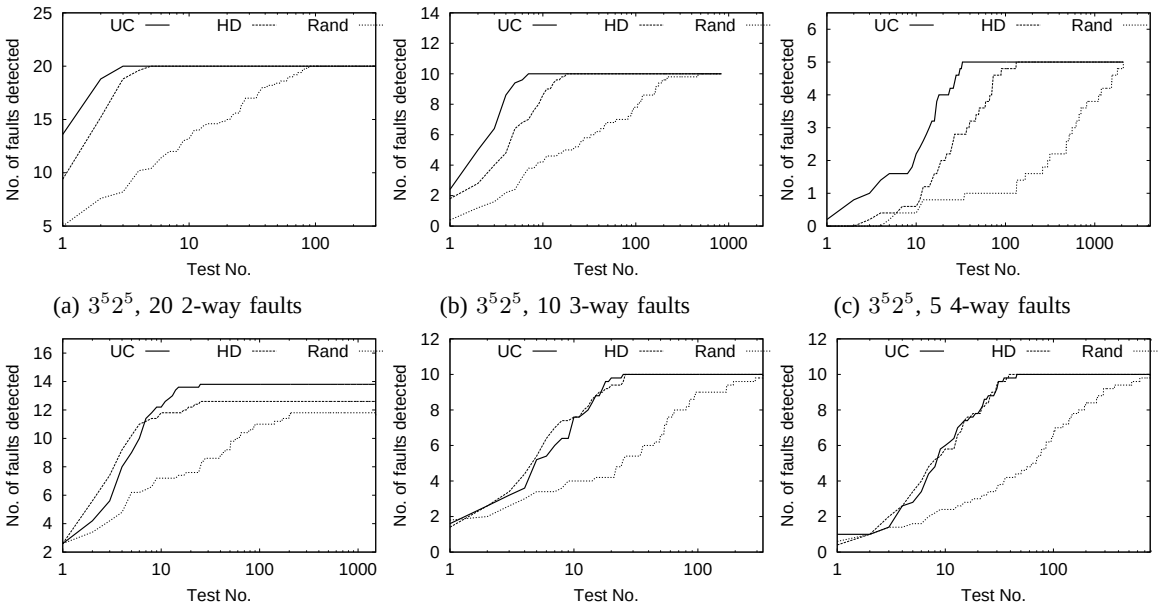
A first simulation uses the testing method statically and only one batch of tests are run. This allows us to more closely compare our results to previous literature. According to criticism, we would anticipate that the distance-based tests and random tests will both cover a large number of $t$-tuples and have comparable rates of fault detection. For instance, in [23], two static combinatorial test suites are compared to random tests. The authors report that between 94.5 to 99% of 2-way combinations are covered in random tests; 97.66 to 99.99% of 3-way combinations; and 99.07 to 99.99% of 4-way combinations. Additional work reports an 88% overlap of pairwise combinations for random tests generated for a subset of MSWord [24]. However, when compared to a broader set of data, it is reasonable to anticipate that distance-based tests will not overlap so substantially with random tests and will indeed perform better than random tests. For instance, Dalal et. al. report a smaller overlap of 68.4 to 99.6% of 2-way combinations in random tests; 41.4 to 94.7% for 3-way combinations; and 10.6 to 88.4% for 4-way combinations [27]. While criticism reports greater overlap of combinations covered in random tests, they also report close overlap in fault detection achieved by combination testing and random testing, suggesting that random testing can be as competitive as combination testing [23], [24]. Due to this controversy, we examine both the overlap of combinations in the distance-based tests and random tests, as well as, the rate of fault detection for each.

A second simulation moves away from previous literature by using the testing method adaptively with our distance-based testing process described earlier in Section §2. More specifically, after each fault found, it is assumed that the test is taken off-line to identify the the parameter-value(s) causing the fault and that the parameter-value(s) are modified to fix the problem. Combinations involving the modified parameter-value(s) may be tested again. In the simulation, after each fault is found with a test, a parameter-value is randomly selected as the "cause" of the fault, it is "modified", and all combinations involving that parameter-value are added back to the list of combinations to cover in future tests. In addition, it is possible in practice that modified parameter-values actually result in new and different faults. In the simulation, one new fault is injected with random probability after each fault that is "fixed".

In both experiments, the rate of fault detection is graphed for the three testing instantiations. This includes (1) distance based on combinations not previously covered; (2) distance based on maximum minimal Hamming distance; and (3) random tests that include untested combination(s).

*1) Simulation 1: A batch of tests with input $3^5 2^5$:* Here we simulate a system with $3^5 2^5$ parameters and values (read as five parameters have three values and five parameters have two values each). The distribution in the number of faults includes a larger number of lower strength interaction faults since reports in literature suggest that this is typically the case [10], [11]. For instance, in [11], several systems are studied and of the interaction faults reported, 6% to 47% are 2-way faults, 2% to 19% are 3-way faults, 1% to 7% are 4-way faults, and even fewer are faults beyond 4-way interactions. The distribution that we use in this experiment is: twenty 2-way, ten 3-way, and five 4-way interaction faults. Since the interaction faults are randomly generated and all three algorithms that generate tests involve some randomization, we run the simulation five times for each experiment and report the average of the runs.

Figures 5-(a-c) show that the rate of fault detection is better with the distance-based test suites. Uncovered combinations distance tests have the quickest rate of fault detection for 2-way, 3-way, and 4-way interaction faults, followed by tests generated with maximum minimal Hamming distance. Random tests are much less effective. For instance, uncovered combinations distance testing finds all 2-way interaction faults in 15 tests, maximum minimal Hamming distance tests in 25 tests, and random tests in 441 tests. The 3-way interaction faults are found in 35 tests with uncovered




(a) $3^5 2^5$, 20 2-way faults (b) $3^5 2^5$, 10 3-way faults (c) $3^5 2^5$, 5 4-way faults

(d) $2^{10}$, 15 2-way faults (e) $2^{10}$, 10 3-way faults (c) $2^{10}$, 10 4-way faults

Figure 5. Rate of interaction fault detection for two simulations.

combinations distance, 90 tests with maximum minimal Hamming distance, and in 2,340 tests with random tests. The difference between the distance-based tests and random tests further grows when 4-way interactions are considered. Uncovered combinations distance testing uses 66 tests to find all 4-way interaction faults, maximum minimal Hamming distances uses 263, and random testing uses 4,977 tests to find these same faults.

The overlap of parameter-value combinations in distance-based tests and random tests is strongest in the earliest tests, but quickly tapers off as more tests are run. For instance, we examine one sample test suite for each of the three types of test suites used. Table I shows the percentage of $t$-tuples covered in each type of test suite after 10, 50, 100, 250, 500, 750, and 1,500 tests. The random tests typically cover many 2-way, 3-way, and 4-way combinations in the earliest tests, but the coverage slows and ultimately, more than ten times as many tests are needed to cover all combinations with random tests as compared to distance-based tests. In addition, the last row of Table I shows the number of tests needed to cover each of the $t$=$\{2,3,4\}$ sized tuples of parameter-value combinations for each of the three testing strategies. Not surprisingly, the rate of fault detection is similar to this overlap - distance-based tests locate faults significantly sooner than random tests. This discrepancy scales significantly as the strength of parameter-value combinations increases. The results are similar to previous literature of combination testing in (see [17], [27], [11] and therein). We continue with an experiment that examines the adaptive use of distance-based testing.

*2) Simulation 2: Adaptive testing with input* $2^{10}$*:* In this second simulation, we model a system with ten parameters that can take on one of two possible values each. Thirty-five interaction faults are initially generated at random: fifteen 2-way faults, ten 3-way faults, and ten 4-way faults. Each of the three adaptive testing approaches are run. When any of these tests find an interaction fault, we randomly select a parameter-value as the "cause" of the fault and assume that it would be fixed by developers. We inject a new 2-way fault involving this parameter-value with 10% probability since modifications sometimes introduce new faults in practice. We then require that all combinations involving this parameter-value be tested again. Since this simulation involves randomization, we run each of the three algorithms five times and report the average of the results.

Figure 5-(d-f) shows the rate of fault detection for each of the three algorithms. Distance based on uncovered combinations and maximum minimal Hamming distance produce fairly similar results, sometimes one instantiation working slightly better than the other. Random testing is less competitive, especially towards the end as a considerable number of tests are needed to find the last few faults. The difference between the two distance-based tests and random tests further magnifies when higher strength faults are considered. For instance, uncovered combinations distance tests find all 3-way interaction faults in 25 tests, maximum minimal Hamming distance tests find all in 27 tests; however random tests take 336 tests to find the ten 3-way faults. The trend continues as uncovered combinations distance tests uncover ten 4-way faults in 46 tests, maximum minimal Hamming

| No. of tests | UC t=2 | HD t=2 | Random t=2 | UC t=3 | HD t=3 | Random t=3 | UC t=4 | HD t=4 | Random t=4 |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 93% | 93% | 39% | 55% | 56% | 21% | 25% | 25% | 11% |
| 50 | 100% | 100% | 64% | 100% | 98% | 42% | 80% | 79% | 25% |
| 100 | 100% | 100% | 76% | 100% | 100% | 53% | 98% | 94% | 34% |
| 250 | 100% | 100% | 90% | 100% | 100% | 71% | 100% | 100% | 51% |
| 500 | 100% | 100% | 98% | 100% | 100% | 84% | 100% | 100% | 66% |
| 750 | 100% | 100% | 99% | 100% | 100% | 90% | 100% | 100% | 75% |
| 1500 | 100% | 100% | 100% | 100% | 100% | 97% | 100% | 100% | 88% |
| No. tests for *t*-way coverage | 15 | 22 | 122 | 52 | 108 | 3814 | 144 | 554 | 7305 |

Table I
THE PERCENTAGE OF T-TUPLES COVERED IN TESTS GENERATED BY UNCOVERED COMBINATIONS DISTANCE, (UC), MAXIMUM MINIMAL HAMMING DISTANCE TESTS (HD), AND RANDOM TESTS.

distance in 39 tests, and random in 782 tests. Indeed, distance-based testing has a more effective rate of fault detection than the random tests in both of our experiments thus far.

### *B. Study 2: Traffic Collision Avoidance System*

The Traffic Collision Avoidance System (TCAS) module is a C program with 9 functions, 138 lines of code, and takes a dozen parameters, each of which can take on numerous values, shown in Table II. This system has been used in independent work that studies dataflow and control flow-based test adequacy criteria [28] and also regression test selection [29]. We use it as an example here because it is an actual system with 41 versions that are seeded with faults. These 41 faulty versions were seeded by researchers at Siemens Corporation with faults that they felt were realistic. For instance, most fault seeding was done independently by 10 individuals; the faults reflect their experience with real software applications.

We use equivalence classes defined in previous work to partition the parameter-values into discrete options shown in Table II [30]. Using these parameters and values, there are 230,400 possible combinations that can be run as tests for each of the 41 versions of code, for a total of 9,466,400 possible tests. If all of these tests can not be run, distance-based testing is a candidate for selecting a limited number of tests to run.

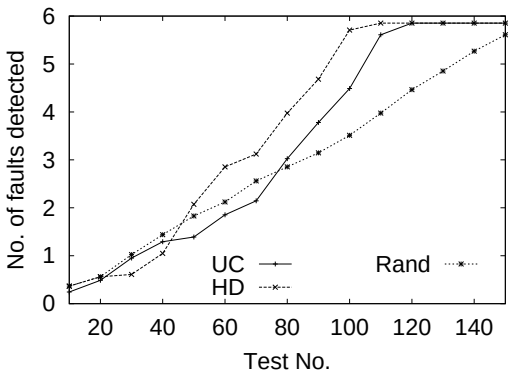
We compare tests generated with distance measures of uncovered combinations and maximum minimal Hamming distance to randomly selected tests. Again, since randomization is used to some degree in each of the algorithms that generate test suites here, we run each five times and report the averages of the five runs.

To determine whether each tests passes or fails, we compare the results to an oracle. If a test fails, we examine the root cause of the fault(s) by hand and also identify the parameter-value combination(s) that triggered the fault(s). For each test that we generate from the input parameters, we compute the expected output with a model checker [31], [30]. A model checker visits all reachable states in a formal specification and verifies that requirements expressed as temporal logic expressions are satisfied. If an expression is not satisfied, the model checker attempts to generate a counterexample - as states and variable values - that prove the existence of the error. Tests can be generated by negating specifications of result states, causing the model checker to produce a counterexample. The counterexample, which demonstrates how the result can be produced, is then post-processed into executable test code. This procedure introduces a certain degree of test case optimization. For example, the model checker automatically filters invalid combinations of parameter values as a result of constraints embedded in the formal specification.

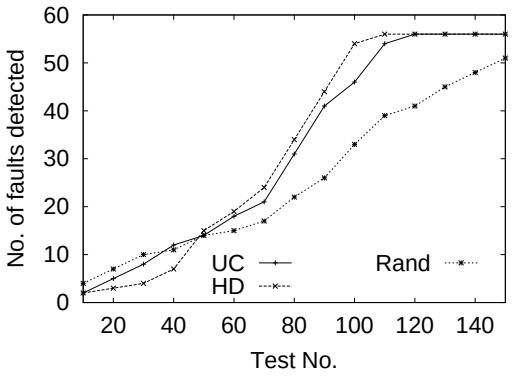
| **Parameter** | **Values** |
|---|---|
| Cur_vertical_sep | 299, 300, 601 |
| High_confidence | true, false |
| Two_of_three_reports_valid | true, false |
| Own_tracked_alt | 1, 2 |
| Other_tracked_alt | 1, 2 |
| Own_tracked_alt_rate | 600, 601 |
| Alt_layer_value | 0, 1, 2, 3 |
| Up_separation | 0, 399, 400, 499, 500, 639, 640, 739, 740, 840 |
| Down_separation | 0, 399, 400, 499, 500, 639, 640, 739, 740, 840 |
| Other_RAC | no intent, do not climb, do not descend |
| Other_capability | TCAS TA, other |
| Climb_inhibit | true, false |

Table II
TCAS VARIABLES.

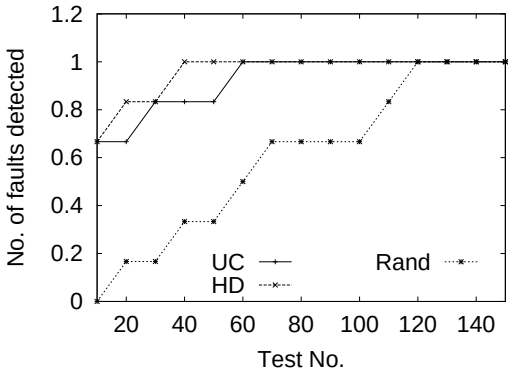
We run the TCAS code and find an average of 5.85 faults triggered by 2-way combinations of parameter-values in each version. However, there is quite a bit of variation in this number as 22 versions do not have any faults that are triggered by 2-way interactions of parameters (all faults in these versions are more complex and involve more than 2-way interactions to trigger faults). In addition, six versions have only one 2-way interaction fault and one version has 56 2-way interaction faults. Therefore, we examine not only the average case, but also some of the extreme cases as well




(a) 41 versions of TCAS




(b) Version 14 of TCAS




(c) Six versions of TCAS that have only one fault triggered by 2-way

Figure 6. Average rate of fault detection for the respective TCAS versions.

in regards to number of faults per version.

Figure 6-(a) shows the average rate of fault detection for the 41 versions of TCAS. When testing actual software here, random testing is quite competitive in the earliest tests but is not competitive after approximately 45 to 80 tests are run. Hamming distance appears to be more effective than uncovered combinations distance than we have seen in our previous simulations.

Version 14 of the TCAS code has 56 faults that are triggered by 2-way interactions. We choose to highlight this example because it has the largest number of 2-way interaction faults among the different versions of the TCAS code. Figure 6-(b) shows the rate of 2-way fault detection. In the initial tests, random testing has a higher rate of fault detection, however, it is still not competitive with the other two distance-based tests when larger numbers of tests are run.

Six versions of the TCAS module have only one fault that is triggered by 2-way interactions of parameters. In Figure 6-(c), we graph the average rate of fault detection among these six versions. Uncovered combinations distance and Hamming distance have fairly comparable rates of fault detection, but that random test case selection is less effective.

While the distance-based testing works well in this example to identify 2-way interaction faults, our initial experiments with $t = \{3, 4, 5\}$ do not exhibit any clear pattern. We find that uncovered combinations and Hamming distance metrics sometimes appear to be more successful at finding the first fault, however, maximizing distance with either of these approaches is not particularly effective. We attribute this to the characteristics of the faults - the numerous faults injected into the TCAS system cluster around similar parameter-values. In cases when faults cluster, these notions of distance may not be adequate. We are currently studying alternate notions of distance that do not penalize next tests based on their proximity to a fault. One can expect such an alternate notion of distance to serve well in locating clustered faults. Indeed if faults are known *a priori* to cluster, the appropriate notion of distance must take this into account. Our current investigations are exploring such alternate notions of distance.

## V. Threats to Validity

The main contribution of this paper is a new testing approach of *adaptive distance-based testing* that can localize interaction faults. Previous work on testing for interaction faults has focused on constructing "small" test suites that cover all $t$-way combinations of parameter-values [14], [15], [16], [17], [32], [33], [19], [20], [21]. Empirical studies then report the number of interaction faults found with these test suites on a variety of systems [23], [6], [7], [8], [9], [10], [11], [12], [24], [13]. The results available in these studies are invalidated if only part of a test suite is run. No studies have indicated whether the size of the test suite has an impact on fault detection, nor what the rate of fault detection is. Our method does not suffer from these two problems, since it dispenses one-test-at-a-time, and the metric is not to find all faults, but rather to find as many as possible in the budget permitted (which is not known to us in advance). Indeed the threats to validity that we face are shared by all of the previous approaches, and we have mitigated some of them by (1) not fixing a test suite size, and (2) changing the metric to find faults as quickly as possible.

Three major threats remain in our work. A major threat to internal validity is that only two proof-of-concept instantiations of distance are used - *uncovered combinations distance* and *maximum minimal Hamming distance*. Distance may be defined in a number of other ways. For instance, distance may be based on complexity (ie: select a 'next' test that covers more complex parts of a system) or based on cost (ie: select a 'next' test that incurs the lowest cost). This threat is particularly raised in the TCAS example where we observe sensitivity in our distance metric when faults cluster around a small number of parameter-values. Therefore, our future work clearly needs to address distance metrics for cases in which interaction faults are typically clustered around a small distribution of parameter-values. A major threat to external validity is that a large family of empirical studies is needed to further validate adaptive distance-based testing. Our simulation injects faults with random probability after each fault is fixed and may not be representative of other systems. The TCAS example is also only one example. Indeed, we conduct only a small initial set of experiments here and a larger family of empirical studies would help us to better understand the relationship of characteristics of applications and distance metrics. A larger study would also allow us to study the scalability of the distance approaches, especially in comparison to random testing and with systems that have larger input spaces. Further, if parameters are not correctly identified for a system, or category partitioning does not select appropriate values for parameters, then the testing may not be effective. Based on the findings of these studies, we believe that it is worthwhile to expand the scope and study of adaptive distance-based testing to a broader variety of actual systems and distance measures.

## VI. Conclusions

Distance-based testing is a systematic testing technique that may be used to augment current testing practices. The methodology is an abstraction of static combination strategies that have been proposed in the past. Instead of generating test suites that are run as a whole, an adaptive one-test-at-a-time process is more flexible. Tests are adaptively generated as systems can undergo modifications. System components may be added, removed, modified or temporarily unavailable and tests will adapt. The effectiveness of the strategy is examined for an actual system and in simulation by measuring the rate of fault detection of dispensed tests.

As distance-based testing can be instantiated using a number of different combination strategies, we considered recent controversy on combination strategies when conducting our experiments. For instance, a specific example of distance-based testing, implemented with "uncovered combinations" has been reported with mixed reviews. The majority of empirical studies report that it is a useful approach, while other work suggests that it offers little benefit over random testing. In addition, previous work only reports the number of faults found with test suites of specific sizes that cover all $t$-way interactions; they do not report how fast the test suites localize faults, nor what happens if a tester can not run an entire test suite. Therefore, the work here on distance-based testing closely examines the overlap of combinations covered by distance-based tests and random tests and also compares their *rates of fault detection*. Two instantiations of distance are studied as a proof-of-concept - "uncovered combinations distance" and "maximum minimal Hamming distance". Our simulation results indicate that random testing is effective only when one runs far too many tests, and hence a comparison of random and structured schemes can be misleading when one fixes a large number of tests in advance to be run. For instance, our simulations indicate that distance-based tests can be more effective in locating faults sooner and in fewer tests than random tests, especially when faults are more complex (ie: more parameter-values interact to cause faults) and faults do not cluster around only few parameter-values. In our experiments, the overlap of combinations covered in the distance-based tests and random tests is quite high in the earliest tests, but this finding does not scale; more than 10 times as many random tests are needed to cover all 2-way, 3-way, and 4-way combinations in our experiments. In the TCAS experiment, distance-based tests work well in finding 2-way interaction faults, but mixed results are observed for 3-way, 4-way, 5-way testing. Indeed, a closer look at the tests and data observe that the 3-way, 4-way, and 5-way interaction faults involve many of the same parameter-values. This suggests that further exploration of distance metrics are needed for systems in which interaction faults cluster around a smaller number of parameter-values.

## VII. Acknowledgements

This research is supported in part by by ARO grant DAAD 19-1-01-0406.

## References

[1] NIST, "The economic impacts of inadequate infrastructure for software testing," March 2003.

[2] M. Grindal, J. Offutt, and S. Andler, "Combination testing strategies: a survey," *Software Testing, Verification, and Reliability*, vol. 15, no. 3, pp. 167–199, March 2005.

[3] T. J. Ostrand and M. J. Balcer, "The category-partition method for specifying and generating functional tests," *Communications of the ACM*, vol. 31, no. 6, pp. 676–686, June 1988.

[4] Y. K. Malaiya, "Antirandom testing: getting the most out of black-box testing," in *Proceeding of the International Symposium on Software Reliability Engineering*, October 1995, pp. 86–95.

[5] J. A. Whittaker and M. G. Thomason, "A markov chain model for statistical softare testing," *IEEE Transactions on Software Engineering*, vol. 20, no. 10, pp. 812–824, 1994.

[6] R. C. Bryce and C. J. Colbourn, "Prioritized interaction testing for pairwise coverage with seeding and avoids," *Information and Software Technology Journal (IST, Elsevier)*, vol. 48, no. 10, pp. 960–970, October 2006.

[7] K. Burr and W. Young, "Combinatorial test techniques: Table-based automation, test generation, and code coverage," in *Proceedings of the International Conference on Software Testing Analysis and Review*, October 1998, pp. 503–513.

[8] S. R. Dalal, A. Karunanithi, J. Leaton, G. Patton, and B. M. Horowitz, "Model-based testing in practice," in *Proceedings of the International Conference on Software Engineering*, May 1999, pp. 285–294.

[9] S. Dunietz, W. K. Ehrlich, B. D. Szablak, C. L. Mallows, and A. Iannino, "Applying design of experiments to software testing," in *Proceedings of the International Conference on Software Engineering*, October 1997, pp. 205–215.

[10] D. Kuhn and M. Reilly, "An investigation of the applicability of design of experiments to software testing," in *Proceedings of the* $27^{th}$ *Annual NASA Goddard/IEEE Software Engineering Workshop*, October 2002, pp. 91–95.

[11] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering*, vol. 30, no. 6, pp. 418–421, October 2004.

[12] R. Mandl, "Orthogonal latin squares an application of experiment design to compiler testing," *Communications of the ACM*, vol. 28, no. 10, pp. 1054–1058, October 1985.

[13] C. Yilmaz, M. B. Cohen, and A. Porter, "Covering arrays for efficient fault characterization in complex configuration spaces," *IEEE Transactions on Software Engineering*, vol. 31, no. 1, pp. 20–34, January 2006.

[14] R. C. Bryce, C. J. Colbourn, and M. B. Cohen, "A framework of greedy methods for constructing interaction tests," in *Proceedings of the 27th International Conference on Software Engineering*, May 2005, pp. 146–155.

[15] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton, "The combinatorial design approach to automatic test generation," *IEEE Software*, vol. 13, no. 5, pp. 82–88, October 1996.

[16] M. B. Cohen, C. J. Colbourn, P. B. Gibbons, and W. B. Mugridge, "Constructing test suites for interaction testing," in *Proceedings of the International Conference on Software Engineering*, May 2003, pp. 28–48.

[17] C. J. Colbourn, "Combinatorial aspects of covering arrays," *Le Matematiche (Catania)*, vol. 58, pp. 121–167, 2004.

[18] R. C. Bryce and C. J. Colbourn, "A density-based greedy algorithm for higher strength covering arrays," *Software Testing, Verification, and Reliability*, vol. 19, no. 1, pp. 37–53, 2009.

[19] K. Tai and L.Yu, "A test generation strategy for pairwise testing," *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 109–111, January 2002.

[20] Y. Tung and W. Aldiwan, "Automating test case generation for the new generation mission software system," in *IEEE Aerospace Conference*, March 2000, pp. 431–37.

[21] A. W. Williams and R. L. Probert, "A practical strategy for testing pair-wise coverage of network interfaces," in *Seventh International Symposium on Software Reliability Engineering*, October 1996, pp. 246–254.

[22] R. W. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, April 1950.

[23] J. Bach and P. Schroeder, "Pairwise testing: A best practice that isn't," in *Proceedings of the Pacific Northwest Quality Conference 2004*, October 2004, pp. 175–191.

[24] P. Schroeder, P. Bolaki, and V. Gopu, "Comparing the fault detection effectiveness of n-way and random test suites," in *Proceedings of the International Symposium on Empirical Software Engineering*, August 2004, pp. 49–59.

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

[26] D. R. Kuhn, R. Kacker, and Y. Lei, "Combinatorial and random testing effectiveness for a grid computer simulator," in *NIST Tech. Report 24, available online: http://csrc.nist.gov/groups/SNS/acts/documents/kuhn-lei-kacker-random-combinatorial.pdf*, October 2008.

[27] S. R. Dalal and C. L. Mallows, "Factor-covering designs for testing software," *Technometrics*, vol. 50, no. 3, pp. 234–243, August 1998.

[28] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow and controlflow-based test adequacy criteria," in *Proceedings of the* $16^{th}$ *International Conference on Software Engineering*, May 1994, pp. 191–200.

[29] G. Rothermel and M. J. Harrold, "Empirical studies of a safe regression test selection technique," *IEEE Transactions on Software Engineering*, vol. 24, no. 6, pp. 401–419, June 1998.

[30] D. R. Kuhn and V. Okun, "Pseudo-exhaustive testing for software," in *Proceedings of the* $30^{th}$ *NASA/IEEE Software Engineering Workshop*, April 2006.

[31] P. E. Ammann, P. E. Black, and W. Majurski, "Using model checking to generate tests from specifications," in *Proceedings of the* $2^{nd}$ *IEEE International Conference on Formal Engineering Methods*, December 1998, pp. 46–54.

[32] R. C. Bryce and C. J. Colbourn, "The density algorithm for pairwise interaction testing," *Software Testing, Verification, and Reliability*, vol. 17, no. 3, pp. 159–182, August 2007.

[33] A. Hartman, "Software and hardware testing using combinatorial covering suites," in *Graph Theory, Combinatorics and Algorithms: Interdisciplinary Applications*, M. C. Golumbic and I. B.-A. Hartman, Eds. Springer, 2005, pp. 237–266.