Experimental Designs in Software Engineering: D-Optimal Designs and Covering Arrays

Dean Hoskins
Arizona State University
Tempe, Arizona 85287
dean.hoskins@asu.edu

Renée C. Turban
Arizona State University
Tempe, Arizona 85287
renee.turban@asu.edu

Charles J. Colbourn
Arizona State University
Tempe, Arizona 85287
charles.colbourn@asu.edu

ABSTRACT

For over a century, Design of Experiment (DOE) techniques have been applied to testing in large problem domains such as agriculture, chemistry, medicine, and industrial design. Recently, the application of DOE has appeared in component-based software testing. This is a natural extension, as software testing is a complex problem that suffers from a combinatorial explosion. Exhaustive testing is not possible in most systems. In this paper, we focus on three areas: *(1) the application of DOE techniques to software testing, (2) improved algorithms for screening tests involving categorical factors, and (3) construction methods for generating covering arrays.*

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.5 [**Software Engineering**]: Testing and Debugging—*debugging aids*; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

General Terms

Experimentation

Keywords

Factorial Experiments, Covering Arrays, D-Optimal Designs

1. INTRODUCTION

Researchers, scientists, and engineers frequently run experiments to discover interesting aspects of a process or system under test (SUT). Often ad-hoc approaches, or one factor at a time methods [8], are used to set up experiments. Consequently either more experiments are run than necessary, or a sub-optimal selection of tests is run without obtaining the desired results. Design and analysis of experiments is a two-step process that begins with a structured approach to setting up experiments, and culminates with a statistical analysis. The result is that with an economy of experiments the information obtained is maximized, leading to statistically valid conclusions regarding the operation of the process or system.

WISER'04, November 5, 2004, Newport Beach, California, USA.

In large problem domains, testing is limited by cost. Every test adds to the cost, so DOE is an attractive option for testing. Specifically in the case of software testing, factorial experiments may be useful. In a factorial experiment, numerical factors are typically run at high and low levels that are near the extents of their ranges to minimize the number of runs required. A factorial experiment may involve multilevel categorical factors and may be done at a full or partial level.

In software engineering, Design of Experiments (DOE) has been explored since the mid 1980s. For instance, Berling et al. [4] applied fractional factorial design as a systematic approach to prototyping and validating a system. They successfully reduced 1,024 test cases down to 112 while obtaining knowledge needed about factor interactions in a network maintenance system [4]. Dunietz et al. also applied experimental designs to testing and reported measures of code coverage that were achieved in [13]. Kuhn et al. measured the number of defects identified at different strengths of interaction coverage in software [16] and also in systems with embedded software [17]. Several other examples include application of experimental designs to computer benchmarking [5], PC Testing [25], network testing [2, 3, 32], and compiler testing [19].

Indeed several of these studies have yielded tools for automatic construction of software test suites. For instance, the Automated Efficient Test Generator tool (AETG) has been developed by Telecordia; NASA has supported the development of the Test Case Generator tool (TCG) [31]; IBM has developed the Combinatorial Test Services tool (CTS) [1]; and TestCover.com [30] has introduced a web-based tool. The current tools construct interaction test suites based on covering arrays. This paper discusses issues with covering arrays, and further examines the use of D-Optimal designs for such testing.

2. BACKGROUND

Much work has done on interaction testing using covering arrays. Here we provide relevant background. We then introduce background on design of experiments.

2.1 Construction of Covering Arrays

Covering arrays have been proposed and empirically studied for application to software interaction testing [7, 4, 36, 25, 13, 16, 17]. Further, several algorithms for automatically constructing these objects have been designed [12, 9, 10, 11, 29, 31, 33, 34, 37, 24].

We begin with some definitions. A *covering array*, $CA_\lambda(N; t, k, v)$, is an $N \times k$ array for which every $N \times t$ subarray has the property that every t-tuple appears at least λ times. In this application,

t is the strength, k is the number of factors (*degree*), and v is the number of symbols for each factor (*order*). When λ is 1, every t-way interaction is covered at least once; this is the case of most interest, and we often omit the subscript λ when it is 1. The covering array is *optimal* if it contains the minimum possible number of rows. The size of such a covering array is the *covering array number* $CAN(t, k, v)$.

This combinatorial object is fundamental in developing interaction tests when all factors have equal numbers of levels. However, software systems are typically not composed of components (*factors*) where each has exactly the same number of options (*levels*). To avoid this restriction of covering arrays, the mixed-level covering array can be used.

A *mixed level covering array*, $MCA_\lambda(N; t, k, (v_1, v_2, \ldots, v_k))$, is an $N \times k$ array in which, for each column i, there are exactly v_i levels; again every $N \times t$ subarray has the property that each possible t-tuple occurs at least λ times. Again λ is omitted from the notation when it is 1. A mixed covering array provides the flexibility to construct test suites for systems in which components are not restricted to having the exact same number of levels.

If a covering array has strength two then any two columns in the arrray must cover all of the possible combinations of levels of each factor. Likewise, for a strength three covering array any three columns must contain all combinations of levels. A covering array of strength *t* is *minimal* when the deletion of any row causes the array not to have strength t.

As an example consider the covering arrays in Tables 1 and 2. The columns of the array are associated with factors A, B, and C, and the factors have 2 levels each, designated as high and low. The Table 1 array is of strength 3, because any three columns contain all of the level combinations of factors A, B, and C. Likewise, it is of strength 2 since and any two columns contain all of the level combinations. Table 1 is also a minimal strength 3 covering array in that deleting any row would reduce its strength to 2. Table 2 is not a strength 3 covering array because any 3 columns do not contain all the possible level combinations of Factors A, B, and C. Table 2 is a strength 2 covering array and is minimal as well since after deleting any one row it can no longer satisy the requirements of a strength 2 covering array.

Factor A	Factor B	Factor C
Low	Low	Low
Low	Low	High
Low	High	Low
Low	High	High
High	Low	Low
High	Low	High
High	High	Low
High	High	High

Table 1: Example Strength 2 and 3 Covering Array

In the practical application of software interaction testing, software systems are typically composed of numerous components. This makes a full factorial experiment too costly, and instead partial experiments are desired. For this reason, we describe methods to produce covering arrays for use in such experiments.

Factor A	Factor B	Factor C
Low	Low	Low
Low	High	High
High	Low	High
High	High	Low

Table 2: Example Strength 2 Covering Array

Computational methods to construct covering arrays fall into three classes of algorithms: algebraic, greedy, or heuristic search. Each of these methods has strengths and weaknesses in *accuracy*, *consistency*, *efficiency*, and *extensibility*. No published algorithm has yet addressed all of these concerns effectively, so software testers must determine the relative importance of these different practical concerns for their application.

We outline some methods here, not in an attempt to be exhaustive, but rather to convey the flavor of available computational methods.

2.1.1 Algebraic constructs

The construction of orthogonal arrays from algebraic objects such as finite fields has spawned many algebraic and combinatorial methods for the construction of covering arrays. For the most part, these methods are fast and accurate, but apply only to very limited sets of parameters. The promise and the challenge of these methods hinges on being able to apply them to a specific experiment. Some first efforts for strength two to apply algebraic methods generally are explored in [33, 34]. Perhaps the most substantial drawback of these methods at present is that obtaining the best accuracy often requires knowledge of the underlying mathematics used in the construction.

TConfig

The TConfig test configuration generator is a deterministic method that utilizes algebraic constructs to build covering arrays quickly [33, 34]. TConfig is relatively accurate and efficient in cases of covering arrays with equal levels and strength two. However, TConfig does not appear to be easily extensible. The current implementation is for pair-wise interactions, and the modification to mixed-level coverage remains an area of future work.

2.1.2 Greedy algorithms

Greedy algorithms have been widely studied for the construction of covering arrays primarily because of their rapid execution. However, much research demonstrates that more sophisticated heuristic search can improve upon accuracy, usually at the expense of greater execution time.

Automated Efficient Test Generator AETG

The AETG algorithm [9, 10] is a general greedy algorithm published for generating interaction tests, as well as a commercial tool that constructs software interaction tests. The algorithm uses randomization with numerous repetitions of runs to produce relatively accurate results.

AETG constructs covering arrays one row at a time. New rows are continually added until all pairs have been covered. For every row that is added to a covering array, M candidate rows are generated. After each iteration, a row is added that covers the most previously uncovered pairs. Each of the candidate rows is generated using a random factor ordering and a heuristic for choosing values based on the number of newly covered pairs.

AETG repeats the process of generating covering arrays numerous times. The randomized factor ordering and the heuristic based on newly covered pairs do not guarantee a good solution. However, in practice AETG does provide relatively accurate solutions using their recommendation of 50 candidates for each test, and 100 repetitions of constructing an entire array.

Deterministic Density Algorithm DDA
Among the greedy algorithms, DDA [12] is relatively competitive on all three criteria of accuracy, efficiency, and consistency. It establishes a theoretical bound on the size of the covering array that is logarithmic in the number of factors, yet is competitive with the other greedy algorithms.

DDA employs only one run as there is no randomization, and does not require extra resources for multiple candidates. Covering arrays are constructed one row at a time until all pairs are covered. The rows are constructed one factor at a time in an order based on a density formula. The density heuristic orders factors based on the likelihood that they cover the most new pairs. This is calculated in relation to both factors that have been fixed, and those that have yet to be fixed. Levels (values for factors), are also selected based on a density heuristic that considers the likelihood of a selection covering the most previously uncovered pairs.

In-Parameter Order - IPO
The In-Parameter-Order algorithm generates covering arrays with a strategy of horizontal followed by vertical growth [37]. The IPO strategy begins with horizontal growth and limited vertical growth such that each component (*k*) is filled in, one at a time, for the initial first v rows, where v is the number of levels for components. After this horizontal growth, if all pairs have not yet been covered, the process is repeated vertically until all pairs are covered and a test suite is generated. To date, the IPO algorithm has only been implemented for 2-way coverage.

Test Case Generator TCG
The TCG algorithm [31] is a deterministic method for constructing covering arrays. The results are generated relatively quickly and are reasonably competitive, especially for mixed-level covering arrays.

TCG adds one row at a time to a covering array and continues adding new rows until all pairs are covered. Before each row is added, up to M candidate rows are generated and a best candidate is the row that is added. M is chosen to be the maximum number of levels for a factor. Factors are fixed in order of decreasing cardinalities of levels. This is particularly beneficial for mixed-level covering arrays. Values for factors are selected to locally optimize the number of covered pairs.

2.1.3 Heuristic search

Heuristic search, in particular simulated annealing (SA) yields a notable improvement in accuracy over greedy methods (see [11], for example). Other search techniques such as hill-climbing and tabu search have also proved effective, while genetic algorithms have until this point been less explored.

Tabu Search
In Tabu search [24], a series of transformations is applied to a covering array. The cost is calculated as the number of uncovered t-tuples. Moves occur by studying the neighborhood and locating rows which allow the change of a single element. The lowest cost move is selected, with tie-breaking done randomly. Moves are only permitted if the move is not tabu, to avoid the problem of looping. A tabu move is a move that has occurred during the last T moves, where $1 \leq T \leq 10$.

Simulated annealing - SA
Simulated annealing (SA) is another heuristic search method that has produced many of the most accurate results to date for both covering arrays and mixed-level covering arrays [11]. A covering array goes through a series of transformations. After a transformation from S to S', if the cost $c(S) \leq c(S')$, then the transformation is accepted. If this inequality is not satisfied, then the transformation is accepted with probability $e^{(c(S')-c(S))/T}$, where T is the temperature. An initial temperature of approximately .2, slowly cooling at the rate of .9998 and .999999 every 2,500 iterations, is recommended in [11].

While this algorithm appears to be relatively accurate it incurs a significant amount of execution time. For instance, Cohen et al. [11] indicate that this accuracy came with a substantially larger time investment than simpler heuristics.

This rapid tour of some construction methods for covering arrays is by no means exhaustive, but serves to illustrate the variety of techniques available for the construction of covering arrays in experimental design.

2.2 Design of Experiments (DOE)

In DOE, the objective is to set up a test plan to run experiments that involve factors (*manipulated variables*) and responses (*dependent variables*). These variables can be either numerical (ratio or interval scales) or categorical (ordinal or nominal scales). Each test is an *experimental run*. In each run, factors are set to specific values within their respective ranges, and responses of one or more variables are recorded. For example we may experiment on how two factors affect the number of defects found in a software system (see Table 3). The two factors in this experiment include (1) the lines of code in a program, and (2) the years of experience for the software developer.

The values of 1 year and 5 years are used here for the experience level and a size of 100 or 10000 lines of code (LOC) for the size of the program are used. The response is measured by the number of defects found per thousand lines of code.

Run	Factor A	Factor B	Response
	Experience (years)	Size of Program (LOC)	Number of Defects
1	1	100	37
2	5	100	12
3	1	10000	27
4	5	10000	23

Table 3: Simple Experiment

In Table 3 both of the factors are numerical; an example of a categorical factor is the language in which the program is written, such as C++ or Java. This experiment is *full factorial* because all combinations of factors/levels have been tested with each other (2x2=4 runs). For larger experiments involving more factors it may be de-

sirable to reduce the number of tests to a subset of the full factorial, this is typically called a *fractional factorial*.

2.3 Analyzing Experiments

The second part of design and analysis of experiments is the statistical analysis. Depending upon the process or system of interest, the analysis may take the form of a simple ANOVA (Analysis of Variance) [22], or involve a more complex Response Surface Analysis (RSA) [20]. In ANOVA the analysis provides information regarding how much each factor (and factor interaction) contributes to the overall variance of the data, indicating the importance (as a percentage) that each factor or interaction plays in the process. These experiments are *screening experiments*, since the experiments differentiate the important factors and interactions from the unimportant ones. In addition to an ANOVA analysis in screening experiments, regression analysis can be employed to determine a relationship between the factors and response variables in the form of an equation [22].

As an example consider the analysis of the experimental data provided in Table 3. The ANOVA is given in Table 4.

Factor or Interaction	% Contribution to Variance
Years Experience	65.5
Lines of code (LOC)	0.10
SLOC x Experience	34.4

Table 4: ANOVA Results for Table 3 Data

By standard techniques [22], one can write a linear model for the response in terms of the factors. For our example, the regression equation is:

$$Defects =$$
$$43.4 - (6.2 \times yrs) - (0.02 \times LOC) + (5.3 \times yrs \times LOC) \quad (1)$$

In this analysis, the experience of the individual is much more important than the size of the code. Secondly, there is an interaction of experience and size. The regression equation indicates that the interaction of larger code and experience increases the number of defects.

To go beyond a simple analysis and optimize a process or system, Response Surface Analysis is appropriate. RSA views the problem environment in terms of two-dimensional contour plots, so that optima can be seen visually or mathematically. While these methods also begin with simple ANOVA techniques, optimizing multiple factors and responses involve more complex methods that allow second order regression and higher models to be estimated.

2.4 Designing an Experiment with DOE Techniques

Designed experiments cover a broad range of types: Full factorial experiments test all combinations of factors and their levels. In this case the number of tests required is

$$\prod_{n=1}^{x} l_n$$

where *n* is the number of factors and *l* is the number of levels associated with each factor.

Factorial experiments test numeric factors at high and low levels. The high and low levels of factors typically correspond to points at or close to boundary values (to get the most value from experimentation). The number of runs for experiments such as these is 2^n where *n* is the number of factors.

Fractional factorial experiments also test numerical factors at high and low levels. The number of runs for these experiments is 2^{n-p}, where n is the number of factors and p is the *fraction* of the factorial. When $p = 1$ it is a *half* fraction, $p = 2$ a *quarter* fraction, and so on.

Reduced factorial experiments in commercial statistical software packages use optimality criteria to reduce the number of runs for experiments involving categorical variables. Optimal designs allow experimenters to generate best designs based on the users choice of sample size, model, ranges on variables, and other constraints [22]. These designs are typically used in optimizing processes where there are irregular experimental regions, nonstandard models, or unusual sample size requirements, but in most cases optimality criteria can be applied to screening experiments as well.

Other experimental designs apply to specific types of experiments, such as: *mixture experiments* where factors are added to one another to form mixtures [18]; *nested and split plot designs* when factors can be manipulated only in groups [14, 15, 35]; *random factor designs* [22] when we are interested in a broad range of values for each factor; and *blocking designs* [15, 22] when an experiment contains uncontrollable factors that we wish to eliminate from affecting the outcome of the experiment.

2.5 Limitations of Experimental Designs

The application of DOE in other fields provides experience that can be applied to software testing. Cost benefits have been mentioned, but there are also known limitations. These limitations continue to be addressed and solved. Some areas where DOE methods are improving include:

1. Needle in a haystack processes
Certain processes may involve hundreds or even thousands of factors. Occasionally a process may be run in which exhaustive (full factorial) testing yields the identification of only a small number of important factors. Defect testing is a good example. Many applications involve millions of lines of source code, but may contain only a few defects. These defects may only show up in rare cases of factor interactions. Obviously exhaustive testing involving many millions of tests is not desired due to cost; nevertheless if the right combination of tests is not performed, we have little chance of locating the defect.

2. Categorical factors.
Exhaustive testing is common in only the smallest systems for designs involving categorical factors. For instance, consider a system with 20 factors that can each take on 4 values. Exhaustive testing requires 4^{20}=1,099,511,627,776 tests!

When categorical factors are present in large numbers, research using DOE typically involves full factorial testing [2, 3, 32]. Some statistical packages offer optimal designs as a solution to reduce the number of runs in screening experiments. However the use of optimal designs for screening is a newer concept and has not received attention in the application to software testing. It appears that cer-

tain optimal designs are currently underused as a mechanism for reducing the number of experimental runs necessary in screening experiments. We discuss this further in Section 3.

3. Screening designs in nonlinear processes
In order to reduce the number of runs in a screening experiment, certain assumptions such as approximate linearity between levels (the different levels at which factors are run) are made. Processes may be inherently nonlinear in nature, requiring the use of more runs. Estimating non-linear models requires more runs than estimating linear models [23], so an economy of runs is more difficult to achieve.

4. Efficiency of test constructions
Generating designs may require substantial amounts of time. Indeed many of the algorithms to generate designs involve the solution of NP-hard problems, and thus appear to need exponential time in the worst case. Techniques to reduce the time it takes to generate an experimental design using covering arrays were discussed in Section 2.1.

3. D-OPTIMAL DESIGNS

In large scale testing domains, every test that is executed incurs costs. Two-level factorial and fractional factorial methods can be used to reduced the number of tests. For experiments involving categorical factors, these methods cannot be used. Some statistical packages employ optimality criteria to reduce the number of runs. Optimality criteria were born out of the need to optimize processes. They are used for screening designs due to their flexibility in designing experiments given only the number of runs desired, the number of factors, and their numbers of levels. Many of the optimality criteria used in RSA have calculations involving the $X'X$ matrix. The X matrix is related to the design matrix where each experimental run is a row, and each factor is a column in the matrix. Consider the design matrix for an experiment involving three factors and one response variable shown in Table 5.

Run	Factor A	Factor B	Factor C	Response
	Network Architecture	MAC Protocol	Load (msg/sec)	Throughput (bits/sec)
1	Classic	IEEE 802.11	2	700.7
2	Swan	EDCF	4	968.6
3	Classic	EDCF	2	984.4
4	Swan	IEEE 802.11	2	1013.8
5	Swan	IEEE 802.11	4	1005.7
6	Classic	IEEE 802.11	4	853.6
7	Classic	EDCF	4	1008.5
8	Swan	EDCF	2	1023.18

Table 5: Networking Experiment

The design matrix considers only the factors and levels that need to be tested; the X matrix additionally considers the model being estimated. Consider a model in which main effects (factors A, B, and C) and two factor interactions (AB, AC, AD, BC, BD, CD) are to be estimated. The X matrix contains terms corresponding to this model, and contains additional columns for interactions (which are missing from the design matrix).

A well known optimality criterion is *D-Optimality* [21]. To understand this criterion we show the matrix form of a regression equation, commonly written as:

$$\beta = (X'X)^{-1}X'y \tag{2}$$

The β vector contains the coefficients in the regression equation relating a response vector y to the factors and interactions composing the X matrix. As an example, consider the equation (model) for two numeric variables, x_1 and x_2 and their interaction (compare to the regression equation for the data in Equation 1):

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 \tag{3}$$

D-Optimality seeks to maximize the determinant of the $X'X$ matrix, in order to minimize the variance of the β coefficients in the regression model. As in [20], from the generalized regression equation, the variance of the β coefficients is:

$$Var(\hat{\beta}) = \sigma^2_{response}(X'X)^{-1} \tag{4}$$

The σ^2 term is the variance of the response and is therefore dependent on the y vector, which is only available after the experiment is completed. The $(X'X)^{-1}$ portion of the equation is dependent only on the experimental setup and not related to the response data. Because of this a portion of the initial design can be optimized before the experiment is run. A convenient way to minimize $(X'X)^{-1}$ is to maximize $|X'X|$, the determinant of the $X'X$ matrix [6].

Some statistical packages use D-Optimality for screening designs. Recall that D-Optimality involves three inputs: number of experimental runs desired; number of factors; and number of levels for each factor. While the number of levels and the number of factors are available to the user, the user must also determine how many experimental runs are necessary; this may not be intuitive. Some statistical software packages (such as JMP© [27] and Design Expert© [28]) attempt to calculate the number of runs required from a user specification of the model to be estimated.

In industrial processes there is a principle called *Sparsity of Effects*) [22] to aid a user in determining the model to estimate. This principle states that main effects are the most important, followed by two-factor interactions, three-factor interactions, and so on. As the interactions involve more terms, the interactions become negligible. Therefore it is often reasonable to use a main effects plus two-factor interaction model. This principle, together with insight into the process or system under test, provides the prospective user of statistical software with information that can make the selection of a model more straightforward.

The use of D-Optimality for experimental screening designs involving categorical factors produces designs that are much closer to full factorial designs than can be achieved by purely selecting a random subset of runs from the full factorial. To date, for any designs involving multilevel categorical factors, D-Optimality is the preferred method by which statistical software packages design experiments. Other methods are used but only for very specific designs involving certain numbers of factors and/or levels [26]. D-Optimality can handle generic designs and it is this flexibility that

has caused its wide spread use over other design methods.

Covering designs have recently been used in tools that generate software interaction tests [1, 9, 30, 31]; however they have not been used in traditional experimental designs tools that also provide statistical analysis of results.

4. METRICS FOR COMPARING DESIGNS

We define some of the metrics used to compare covering arrays and D-optimal designs.

4.1 Metrics for Reduced Designs

We need a basis for comparison of D-Optimal designs and covering arrays. Our ultimate goal is to produce reduced designs (a subset of a full factorial) that approximate the ANOVA results from the original full factorial data well. The *percentage contribution to variance* is a desirable way to compare such designs. As an example consider the percentage contributions as analyzed from the data in Table 5 and shown in Table 6. Here we are interested in the ANOVA for main effects plus two-factor interactions (ME + 2FI). Additionally consider the experiment in Table 5 minus the fifth run, making it a $\frac{7}{8}^{th}$ fraction of the full factorial experiment. The ANOVA for both designs is shown in Table 6.

	Original Full Factorial Data	**Reduced Design Data**	**Difference**
Factor or Interaction	**contrib. to variance(%)**	**contrib. to variance(%)**	**contrib. to variance(%)**
A	30.4	26.9	3.5
B	23.8	30.5	6.7
C	1.83	0.14	1.69
AB	30.6	29.5	1.1
AC	8.09	8.07	0.2
BC	4.34	4.92	0.58

Table 6: Combinatorial Data for Numbers of Runs

We use the sum of squares error between the full factorial data and the reduced design as an indicator of how well the experimental design was constructed:

$$ErrorSquaredSum = \sum^{ME+2FI} (\rho_{ff} - \rho_{rf})^2 \qquad (5)$$

where ρ_{ff} and ρ_{rf} are the percentage contributions to variance for the full and reduced designs specific to a particluar factor or interaction. For the reduced design above the sum is 61.6. If other designs were analyzed their sum could be compared to the above and if the sum were smaller the design would be considered better at estimating the original full factorial model.

Likewise the same could be done for the above with the addition of three factor interactions (3FI). In this case difference terms for ABC, ABD, and BCD would need to be calculated (i.e. for all of the 3-factor interactions).

4.2 Covering Variance Metrics

In order to differentiate covering arrays of the same strength, a metric is required. We propose a new metric, *covering variance*, to measure how evenly the design is 2 or 3 covered with the extra runs available (i.e., the covering array is not minimal). The 2 and 3 covering variance calculation is shown in equations 6 and 7:

$$\sum^{allcolumnpairs} \sum^{allpermutationsfactors/levels} (n_i - n_d)^2 \qquad (6)$$

$$\sum^{allcolumntriples} \sum^{allpermutationsfactors/levels} (n_i - n_d)^2 \qquad (7)$$

where n_i is the number of times a particular level combination occurs in the covering array and n_d is the number of times it should occur for the most even covering.

Consider another example of an experimental design where 2 factors at 2 levels each are involved (call the factors A and B and the levels of each high and low). There are 4 possible combinations of these factors. If our experimental design requires 8 runs rather than 4 then the best (most evenly covered) design would be the 4 possible combinations doubled (replicated) to produce an 8 run design. The experimental runs are shown in Table 7 and designated Design 1. The calculation for covering variance is shown in equation 8.

$$\overbrace{(2-2)^2}^{A_{low}B_{low}} + \overbrace{(2-2)^2}^{A_{low}B_{high}} + \overbrace{(2-2)^2}^{A_{high}B_{low}} + \overbrace{(2-2)^2}^{A_{high}B_{high}} = 0 \qquad (8)$$

The variance is zero since there are 2 complete sets of factor combinations, for all the possible factor and level permutations and this is the most even covering possible. Note that there is 1 row of data shown in Equation 8, corresponding to the only 2 column pair AB (the leftmost summation in Equation 6). Consider Designs 2 and 3 in Table 7, the calculations for these are:

$$Design_2 = (4-2)^2 + (2-2)^2 + (1-2)^2 + (1-2)^2 = 6 \quad (9)$$

$$Design_3 = (5-2)^2 + (1-2)^2 + (1-2)^2 + (1-2)^2 = 12 \quad (10)$$

Design 3 with a covering variance of 12 is the worst possible design in that many runs test the 2 factors at the same levels, corresponding to one of the most unevenly 2-covered designs.

Run	Design 1 Factors		Design 2 Factors		Design 3 Factors	
	A	B	A	B	A	B
1	Low	Low	Low	Low	Low	Low
2	Low	High	Low	Low	Low	Low
3	High	Low	Low	Low	Low	Low
4	High	High	Low	Low	Low	Low
5	Low	Low	Low	High	Low	Low
6	Low	High	Low	High	Low	High
7	High	Low	High	Low	High	Low
8	High	High	High	High	High	High

Table 7: Comparative Covering Variance for 3 Designs

5. COMPARING DESIGNS

Data was taken from a 4 factor mobile ad-hoc networking experiment [32]. Information concerning these factors is summarized in Table 8.

Factor	Number of Levels	Type of Factor
Network Architecture	3	categorical
MAC Protocol	2	categorical
Routing Protocol	2	categorical
Speed	2	numeric

Table 8: Mobile Ad-hoc Experiment Factor Information

Exhaustive testing was performed to compare the following design construction methods:

- D-Optimal Designs;
- Strength 2 Covering Designs (referred to as 2 covering designs from now on); and
- Strength 3 Covering Designs (referred to as 3 covering designs from now on).

The task is to estimate a minimum model of main effects plus two-factor interactions (ME+2FI) with the possibility of extending it to include three-factor interactions (ME+2FI+3FI). In order to estimate these models, 15 runs and 22 runs from the full factorial data must be selected at a minimum. The minimum requirement of 15 and 22 runs is based on being able to estimate all main effects and interactions for the ME+2FI and ME+2FI+3FI models respectively. Any less runs would mean the inability to adequtely measure all of the main effects and interactions, any more runs be acceptable but unessessary.

Exhausive testing was performed for 15 runs through 18 runs. The number of ANOVA's necessary to run each set of exhaustive tests can be calculated based on a combination of runs out of a full factorial experiment which has 3x2x2x2 = 24 runs (3,2,2,2 correspond to the number of levels each factor has). Data is shown in Table 9.

Number of runs tested	Equation	Number of tests (exhaustive analysis)
15	${}_{24}C_{15}$	1,307,504
16	${}_{24}C_{16}$	735,471
17	${}_{24}C_{17}$	346,104
18	${}_{24}C_{18}$	134,596

Table 9: Combinatorial Data for Numbers of Runs

Results for Error Squared (Equation 5) are shown in Figures 1 to 3, and can be used collectively to compare the three design construction methods as to how well they fare against each other and for estimating full factorial results. The graphs show:

1. Similar shape of curves for all data within a particular analysis (determinant, 2 covering, 3 covering)
2. Constant improvement in error squared as the number of runs increase within a design
3. Constant improvement in error squared as $|X'X|$ gets larger

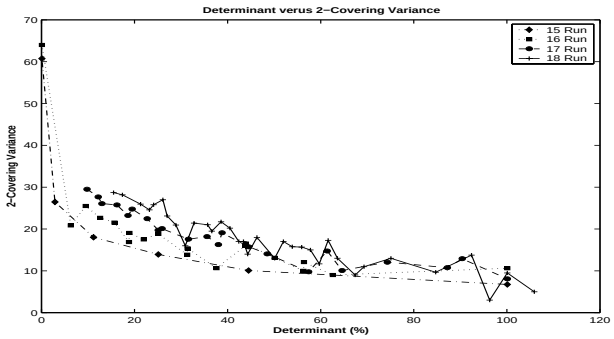

Figure 4: Determinant versus 2-Covering

4. Constant improvement in error squared as the covering improves (lower covering variance is better)

The performance of each design method in terms of increasing error squared:

$$3\,Covering \leqslant D-Optimality \leqslant 2\,Covering \leqslant Random$$

Specifically, 3 coverings were better than D-Optimal designs by between 39-74%, and D-Optimal designs were better than 2 coverings by 2-24%. The average Error Squared for random selection on 15 runs through 18 runs was 361.2, 310.2, 259.1 and 209.7 respectively (note that the data points on the graphs do not have equal numbers of samples). If the only concern is matching full factorial results as closely as possible, 3 covering is superior.

The exact relationship between D-Optimal designs based on $|X'X|$ and covering arrays is not known, primarily because the statistical analysis of covering arrays is not well understood. The graphs in Figure 4 relate $|X'X|$ and two-coverings. The graph suggests a relationship of:

$$2\,Covering\,Variance = f(\frac{1}{|X'X|^n}) \qquad (11)$$

Analysis of 3 covering data is expected to have a similar relationship; however no data was collected to determine this.

Lastly D-Optimal designs are approximately characterized by maximizing the distance between all points in a design. As an example consider an experiment involving 3 factors at 2 levels each. The full factorial is shown in Figure 5a, with each dimension representing a factor and each level of the factor representing a point on the edges of the cube. Figure 5b shows a half factorial design of 4 points that is D-Optimal, and there is no other arrangement of the points giving a greater distance between all points (although there are other designs that are equally good). Figure 5c shows a suboptimal design where the distance is lower than that of the design in Figure 5b.

Large distances among all points is also considered a good design criterion. Therefore data was collected showing distance among all points for the three types of designs under consideration (see Figures 6 to 8). Then:

- The 2 covering distance is very close to linear;
- $|X'X|$ versus distance is confined to a smaller range than covering variance data;

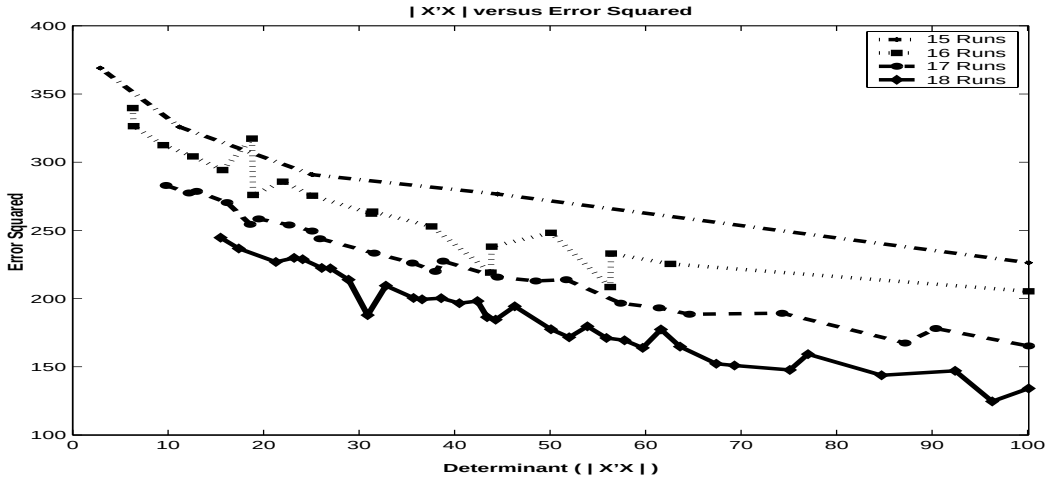

Figure 1: $|X'X|$ **versus Error**

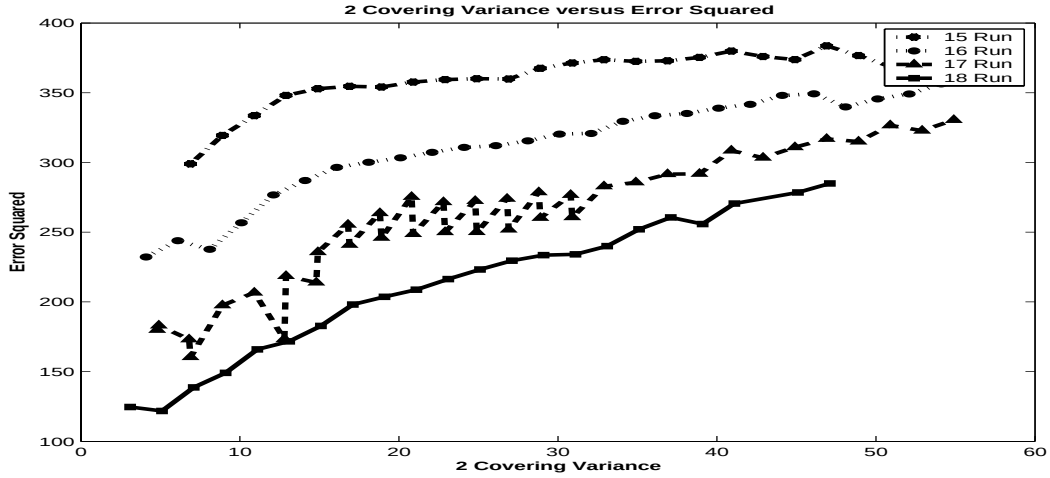

Figure 2: 2 Covering Variance versus Error

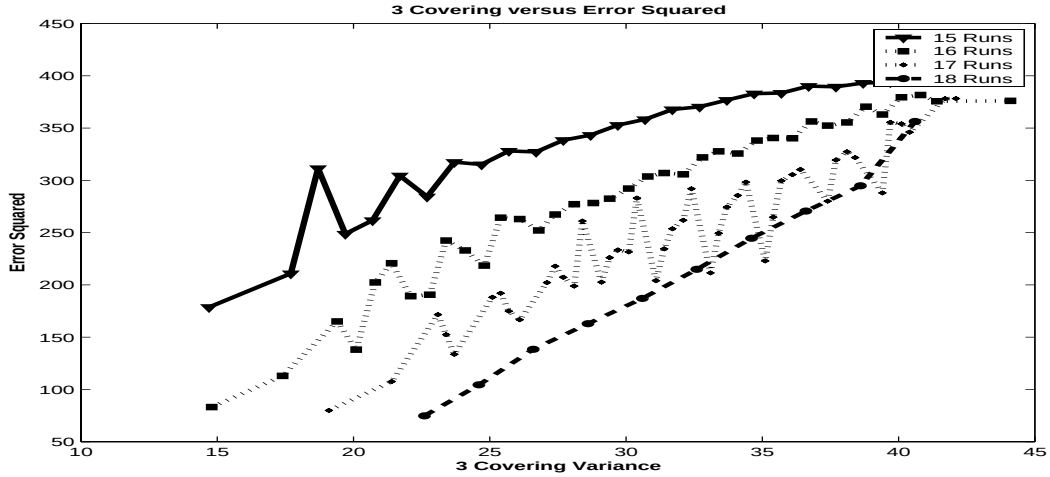

Figure 3: 3 Covering Variance versus Error

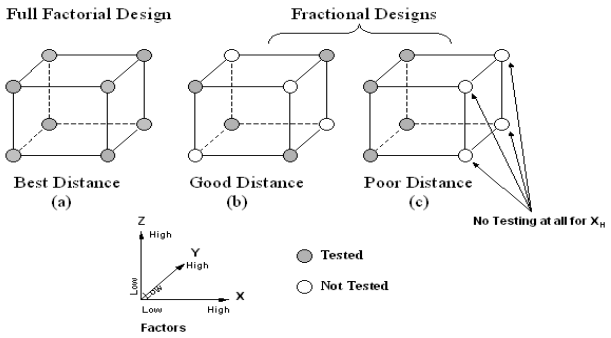

Figure 5: Visual Display of Distance for Different Designs

- The 3 covering vs distance appears to be a sawtooth pattern.

In more detail:

- 2 covering distance is usually greater than $|X'X|$, but performs worse in coming close to full factorial designs
- 3 covering distance (for the best 3 covering designs) is generally worse than both 2 covering and $|X'X|$ designs but performs better in coming close to full factorial designs

Since the best (lowest covering variance) 3 covering designs do not have a high distance but perform better in approxiamating full factorial designs, distance may not be as good an indicator of design capability as originally thought [20].

6. DIFFERENCES BETWEEN COVERING ARRAYS AND D-OPTIMAL DESIGNS

When a covering array tests all two way or three way combinations of factors/levels, the experiment should be able to *identify* two factor interactions and three factor interactions. While D-Optimal designs have a statistical basis concerning *estimation* of these interactions, covering arrays have not been studied from this viewpoint. D-Optimal designs are concerned with *measuring interactions*, while covering arrays are concerned with accurate coverage of interactions. A potential strength of covering arrays is that isolation and detection of information (such as defects) does not necessarily mean having to measure interactions. Here we explore how covering arrays fare in the measurement of interactions, and we consider an array to be better based on its ability to estimate interactions.

The best covering arrays (based on low covering variance) are much better than purely random designs, and in the cases analyzed are better than D-Optimal designs. Generating optimal, or even minimal, covering arrays is computationally difficult. Nevertheless we may not require a minimum covering array, and therefore we may not always have a difficult search problem.

6.1 Covering Arrays, D-Optimal Designs and Aliasing

When using D-Optimal designs with enough runs to estimate two factor or three factor interactions in addition to main effects, no aliasing occurs among the main effects or interactions. A common occurrence when using fewer runs than a full factorial is that when the analysis is completed, it may not be possible to get distinct estimates of all factors and interactions (this is *aliasing*). For example, you cannot discern the difference between the A effect and the BC interaction, and therefore you cannot tell which one is contributing to the variance. Table 10 provides an example; when analyzed the results show the effect of A accounts for over 70% of the variance; since the BC interaction is aliased with A you cannot tell how much variance is accounted for by A or BC, only that the combination of both is 70%. The experimental setup in Table 10 is a covering array of strength 2, and a 2^{3-1} fractional factorial as well. If catching as many defects in code as possible is the goal, then aliasing may not be a concern. If the desire is to calculate a regression equation relating factors to response, then separating the effects estimates (eliminating aliasing) becomes of paramount importance.

Run	Factor A	Factor B	Factor C	Response
	Machine Speed (rpm)	Temp (F)	Pressure (psi)	Quality (defects)
1	20	90	150	5
2	20	75	200	10
3	40	90	200	15
4	40	75	150	7.5

Table 10: Industrial Experiment

Covering arrays do not avoid aliasing as D-Optimal designs do; care must be taken to analyze a covering design to see if aliasing occurs.

6.2 Combining Covering and Optimal Designs

D-Optimal designs offer estimation and covering arrays do not appear to, while covering arrays offer an intuitive notion of coverage. Marrying these methods may have benefits that neither of them has separately. This is the motivation for mixed or hybrid designs. Indeed it may be possible to extend a covering array to a D-Optimal design.

Sequential designs are ones in which a portion of an overall experiment is run, and then augmented based on the analysis of the first set of data. We may be able to design mini-experiments based on covering arrays, and then augment them with additional runs after analyzing the data. The additional runs can be used to gather data that could not be obtained from the original experiment, without wasting runs that would occur if a larger experiment had been designed and run first.

It is common to set up the experiment with a particular model in mind. There may be cases when designing an experiment to estimate main effects plus two factor interactions may produce poor results because of unanticipated three factor interactions. With standard screening designs importance would be placed on fitting the best possible ME+2FI model to the data. With *bias minimizing designs* this is not the case, addressing the possibility of minimizing the error associated with an actual ME+2FI+3FI model is a higher priority. If the process or system actually follows a ME+2FI model then there will be some loss in accuracy to estimate it (over standard optimal designs), however, the loss is usually not significant, and is warranted by the greater risk of estimating the model with a large error. This concern is presently not addressed in screening experiments, nor by using optimal designs. This is an area where progress can be made in designing experiments that produce superior results when the practitioner does not know what type of model the process or system follows.

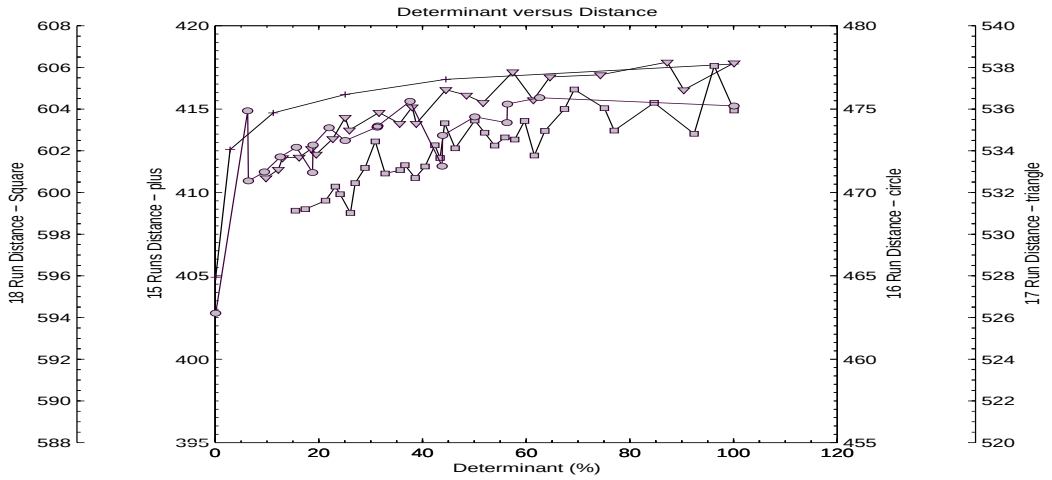

Figure 6: $|X'X|$ versus Distance

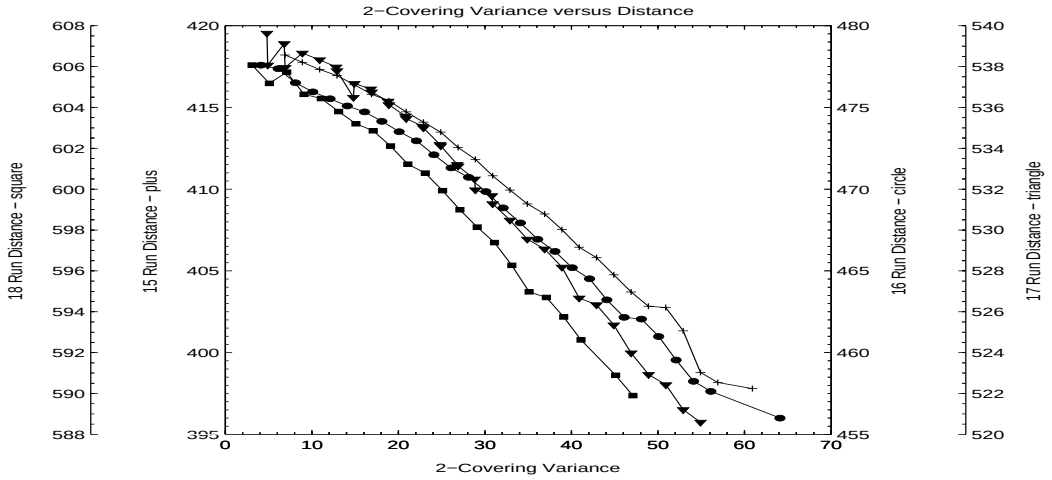

Figure 7: 2 Covering Variance versus Distance

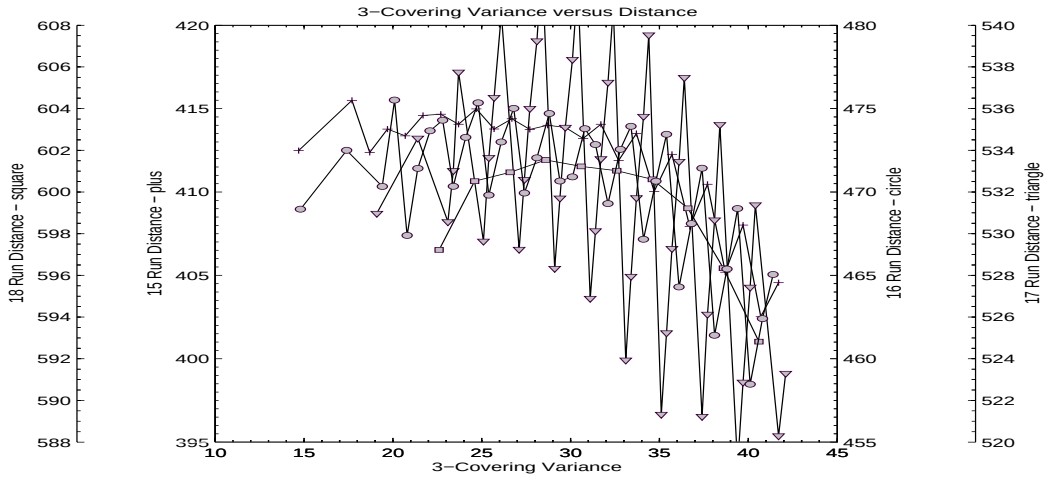

Figure 8: 3 Covering Variance versus Distance

6.3 Orthogonal Properties of Designs

Exact (100%) D-Optimal designs produce an X matrix that is as close to orthogonal as possible; good covering arrays do not always share this property. Figure 4 shows that strength 2 and 3 covering arrays that are good by the covering criterion have a relatively high determinant value ($|X'X|$), but there is no linear relationship. Therefore, while $|X'X|$ is important, it may not play a critical role in producing good experimental designs.

7. CONCLUSIONS

Design of Experiment (DOE) techniques may be applied to software interaction testing in several ways. Currently, interaction testing has been explored using covering arrays. This paper compares the application of covering arrays with that of D-Optimal Designs.

The use of covering arrays in interaction testing has been empirically studied, and several algorithms enable the automatic construction of these interaction test suites. Further, D-Optimal Designs are examined here for interaction testing. Preliminary work indicates that D-Optimal and covering array designs can serve as reasonable estimates of full factorial test results. We have seen that they often outperform randomly generated partial experiments for estimation as well. To exploit the features of D-Optimal designs and covering arrays, we are investigating mixed designs, sequential designs, and bias minimizing designs. Their properties when setting up partial experiments involving categorical factors appear promising as a means to combine the best features of estimation and identification.

The application of DOE to software testing needs to provide further statistical support, as well as to address the practical concerns of the software testing community. Designs tested in preliminary research involve limited numbers of factors, limited mixes of categorical and numerical variable types, limited numbers of levels in categorical factors, and specific subsets of full factorial data. Future research is to consider a broader scope of testing to address these limitations.

8. ACKNOWLEDGMENTS

Research is supported by the Consortium for Embedded and Internetworking Technologies and by ARO grant DAAD 19-1-01-0406.

9. REFERENCES

[1] Alphaworks:IBM. Combinatorial test services tool. http://www.alphaworks.ibm.com/tech/cts.

[2] C. Barrett, M. Drozda, A. Marathe, and M. Marathe. Analyzing interaction between network protocols, topology and traffic in wireless radio networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, pages 1760–1766. IEEE, March 2003.

[3] C. Barrett, A. Marathe, M. Marathe, and M. Drozda. Characterizing the interaction between routing and mac protocols in ad-hoc networks. In *Proceedings of the Third ACM Int. Symp. Mobile Ad Hoc Networking and Computing*, pages 92–103. ACM, June 2002.

[4] T. Berling and P. Runeson. Efficient evaluation of multifactor dependent system performance using fractional factorial design. *IEEE Transactions on Software Engineering*, 29(9):769–781, September 2003.

[5] R. F. Berry. Computer benchmark evaluation and design of experiments: A case study. *IEEE Transactions on Computers*, 41(10):1279–1289, October 1992.

[6] J. Borkowski. A comparison of predicition varaince criteria for response surface designs. *Journal of Quality Technology*, 35(1):70–77, January 2003.

[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*, pages 503–513. SQE, October 1998.

[8] J. Cawse. Experimental design for combinatorial and high throughput materials development. In *GE Global Research Technical Report*, pages 1–27. GE, November 2002.

[9] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–44, July 1997.

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

[11] M. B. Cohen, C. J. Colbourn, P. B. Gibbons, and W. B. Mugridge. Constructing test suites for interaction testing. In *25th International Conference on Software Engineering*, pages 38–48. IEEE, May 2003.

[12] C. J. Colbourn, M. B. Cohen, and R. C. Turban. A deterministic density algorithm for pairwise interaction coverage. In *IASTED Proceedings of the International Conference on Software Engineering*, pages 345–352. SE2004, February 2004.

[13] I. Dunietz, W. Ehrlich, B. Szablak, C. Mallows, and A. Iannino. Applying design of experiments to software testing. In *Proceedings of the 19th International Conference on Software Engineering*, pages 205–215. ACM, May 1997.

[14] R. Fisher. The arrangement of field experiments. *Journal of the Ministry of Agriculture of Great Britain*, 33(6):503–513, October 1926.

[15] P. Goos. *The Optimal Design of Blocked and Split Plot Experiments*. Springer-Verlag, New York NY, 2002.

[16] D. Kuhn and M. Reilly. An investigation of the applicability of design of experiments to software testing. In *Proceeding of the 27th Annual NASA Goddard Software Engineering Workshop*, pages 91–95. NASA, December 2002.

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

[18] M. Lewis. *Pharmaceutical Experimental Design*. Marcel Dekker Inc, New York NY, 2000.

[19] R. Mandl. Orthogonal latin squares: an application of experiment design to compiler testing. *Communications of the ACM*, 28(10):1054–1058, October 1985.

[20] R. Meyers and D. C. Montgomery. *Response Surface Methodology (Second Edition)*. John Wiley and Sons, New York NY, 2002.

[21] T. Mitchell. An algorithm for the construction of 'd-optimal' experimental designs. *Technometrics*, 16(2):203–210, January 1974.

[22] D. C. Montgomery. *Design and Analysis of Experiments (Fifth Edition)*. John Wiley and Sons, New York NY, 2001.

[23] J. Neter, M. Kutner, C. Nachtsheim, and W. Wasserman. *Applied Linear Statistical Models (Fourth Edition)*. McGraw-Hill, New York NY, 1996.

[24] K. Nurmela. Upper bounds for covering arrays by tabu

search. *Discrete Applied Math.*, 138(1):143–152, March 2004.

[25] M. S. Phadke. Planning efficient software tests. *CrossTalk - Journal of Defense Software Engineering*, 10(10):11–15, October 1997.

[26] R. L. Plackett. The design of optimum multifactorial experiments. *Biometrica*, 33(4):305–325, June 1946.

[27] Jmp in statistics version 5.1. SAS.

[28] Design expert. StatEase Inc.

[29] K. C. Tai and L. Yu. A test generation strategy for pairwise testing. *IEEE Transactions on Software Engineering*, 28(1):109–111, January 2002.

[30] Testcover.com. Testcover.com tool. http://www.testcover.com/.

[31] Y. W. Tung and W. S. Aldiwan. Automating test case generation for the new generation mission software system. In *Proceedings of the IEEE Aerospace Conference*, pages 431–437. IEEE, March 2000.

[32] K. Vadde and V. R. Syrotiuk. Factor interaction on service delivery in mobile ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 22(7):1335– 1346, September 2004.

[33] A. W. Williams. Determination of test configurations for pair-wise interaction coverage. Testing of communicating systems: Tools and techniques. In *Proc. IEEE Aerospace Conference*, pages 59–74. ACM, August 2000.

[34] A. W. Williams and R. L. Probert. A measure for component interaction test coverage. In *Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications*, pages 304–311. ACS/IEEE, June 2001.

[35] B. J. Winer. *Statistical Principles in Experimental Design second edition*. McGraw-Hill, New York NY, 1971.

[36] C. Yilmaz, M. Cohen, and A. Porter. Covering arrays for efficient fault characterization in complex configuration spaces. In *Intl. Symp. on Software Testing and Analysis*, pages 45–54. ACM, July 2004.

[37] L. Yu and K. C. Tai. In-parameter-order: a test generation strategy for pairwise testing. In *Proceedings of the Third IEEE International High-Assurance Systems Engineering Symposium*, pages 254–261. IEEE, November 1998.