Math.Comput.Sci. (2018) 12:429–451
https://doi.org/10.1007/s11786-018-0385-x

Mathematics in Computer Science

On a Combinatorial Framework for Fault Characterization

Charles J. Colbourn · Violet R. Syrotiuk

Received: 27 October 2017 / Revised: 27 April 2018 / Accepted: 28 April 2018 /
Published online: 20 September 2018
© Springer Nature Switzerland AG 2018

Abstract Covering arrays have been widely used to detect the presence of faults in large software and hardware systems. Indeed, finding failures that result from faulty interactions requires that all interactions that may cause faults be covered by a test case. However, finding the actual faults requires more, because the failures resulting from two potential sets of faults must not be the same. The combinatorial requirements on test suites to enable a tester to locate the faults are developed, and set in the context of similar combinatorial search questions. Test suites known as locating and detecting arrays to locate faults both in principle and in practice generalize covering arrays, thereby addressing combinatorial fault characterization. In common with covering arrays, these locating and detecting arrays scale logarithmically in size with the number of factors, but unlike covering arrays they support complete characterization of the interactions that underlie faults.

Keywords Covering array · Combinatorial testing · Locating array · Detecting array · Compressive sensing · Experimental design

Mathematics Subject Classification 68R05 · 62K05 · 05B40 · 05B15

1 Faults, Failures, and Testing

Software systems are complex systems whose correctness and performance are affected by numerous factors, including the components from which they are built, the physical and computational environment in which they are run, and the inputs they are provided, among others. This paper explores fundamental questions concerning combinatorial requirements for a test plan to be able to reveal, isolate, and characterize faults. In order to make

Research of CJC was supported in part by the Software Test & Analysis Techniques for Automated Software Test program supported by OPNAV N-84, United States Navy. Research of CJC and VRS was supported in part by the National Science Foundation under Grant No. 1421058.

C. J. Colbourn (✉) · V. R. Syrotiuk
Computing, Informatics, and Decision Systems Engineering, Arizona State University, PO Box 878809,
Tempe, AZ 85287-8809, USA
e-mail: colbourn@asu.edu

V. R. Syrotiuk
e-mail: syrotiuk@asu.edu

these questions precise, we must determine the types of faults that are to be examined, the types of test cases that are feasible, the manner in which such test cases are executed, and the test outcomes that are possible. Our objective is to explore determinations of each that are neither so specific that they cannot apply to real systems, nor so generic that few conclusions can be drawn.

Roughly speaking, a test case in a test plan can be executed and the output or behaviour of the system measured. Interpreting the outcome of a test case execution requires knowing what the "expected" or "correct" outcome is; deviation from this is a *failure*. In some cases, the magnitude of the deviation can be measured, while in others only the presence of a significant deviation can. In an attempt to classify outcomes, one uses a reference model of correct performance. Often such models are incomplete, in which case classifying the outcome as failed or normal is problematic. Indeed, the reference model itself may not correctly capture the operation that the developers or users expect. These concerns make the classification of outcomes in terms of their status as failures difficult in any complex system. We do not attempt to resolve this very challenging problem here; rather we make no assumption that the reference model is correct, and treat a failure as evidence that either the reference model or the system itself is incorrect.

When a test case execution results in failure, but the reference model is correct, the system contains a *fault*, or perhaps many faults. Depending on the nature of the fault, it may result in failure for many different test cases. We cannot fix failures, except by correcting faults; but we cannot find faults without witnessing the failures that they cause.

One major goal of testing is to ensure that when a test plan is executed, a failure will be witnessed whenever a fault is present. A second, equally important, goal is to isolate or characterize the fault so that it can be corrected. Without any limitations on the nature of faults, neither of these can be accomplished. More realistic objectives are to reveal the presence of, or perhaps characterize, restricted types of faults that may be present.

Suppose that the type of potential faults to be treated has been specified. When a fault is present, the first goal requires that the test plan contain a test case whose execution results in failure. Once a failure is encountered, a fault must be present. Because the (unknown) correspondence between faults and failures is many-to-many, further test case executions are typically needed to characterize a fault, not just find a failure. Then the regime in which the test plan is executed plays a substantial role. If the test plan can be adapted to alter future test cases when a failure is encountered, testing is *adaptive* or *sequential*. When it is fixed, independent of the failure outcomes, testing is *nonadaptive* or *predetermined*. We focus on nonadaptive testing but comment in places on the adaptive regime.

In order to make the ideas and the framework precise, we first explore simpler testing problems and outline their combinatorial foundations. Then we develop the combinatorial testing framework that is widely used in software interaction testing, and examine the combinatorial basis for fault characterization. Although much of the ensuing discussion simply surveys relevant work, we focus particularly on probabilistic methods, not previously applied to these problems, and their algorithmic consequences.

2 Main Effects

In order to develop the combinational characterization, we first explore some problems in which faults arise either from a single factor, or from a single set of factors. The impact of a single factor is a *main effect*.

2.1 Combinatorial Group Testing

A simple scenario is useful to formalize some basic concepts. See [25] for a much more complete discussion. Suppose that there are k items. Each may be *defective* (faulty) or not. A *test case* is a subset of the items; it results in failure if and only if the test case contains a defective item. Hence test cases are "pools" of items.

A test plan to reveal the presence of a defective is remarkably simple: Form one pool containing all items, and check for failure. If no failure arises, no item is defective; if a failure arises, at least one item is defective. This idealized situation demonstrates clearly the difference between revealing the presence of a defective and finding

the defective items, because for the test plan with one test case, failure gives no information about which items are defective.

How can the defective items be found nonadaptively? When any number of items can be defective, there are 2^k possible sets of defective items. Each test case has two possible outcomes, so when N test cases are run there are 2^N sets of outcomes possible. To characterize the set of defective items, then, it must happen that $2^N \geq 2^k$, that is $N \geq k$. Indeed in the absence of further information, there can be no better test plan than to test items individually.

Now suppose further that an *a priori* upper bound d on the number of defective items is known (or estimated). Then there are $\sum_{i=0}^{d} \binom{k}{i}$ possible sets of defective items, and a lower bound on the number N of test cases needed is obtained from $2^N \geq \sum_{i=0}^{d} \binom{k}{i}$. When d is small compared to k, this opens the possibility of using much fewer than k test cases.

Combinatorial group testing is the study of test plans to find defectives in a population of k items when at most d are defective. For a test plan to support recovery of a set of at most d defectives, the pattern of failures observed must correspond to a unique set of at most d items. Let us make this precise. Index the test cases by $\{1, \ldots, N\}$. For each item x, let $\rho(x)$ be the set of indices for the set of test cases in which x appears.

Provided that at most d defectives are present, we can recover the set of defectives D if and only if there is no other set D' of at most d defectives for which

$$\rho(D) := \bigcup_{x \in D} \rho(x) = \bigcup_{x \in D'} \rho(x) =: \rho(D').$$

Let $\mathcal{D}$ be a set containing k subsets of $\{1, \ldots, N\}$. If for every $\mathcal{D}_1 \subseteq \mathcal{D}$ with $|\mathcal{D}_1| \leq d$, and every $\mathcal{D}_2 \subseteq \mathcal{D}$ with $|\mathcal{D}_2| \leq d$,

$$\bigcup_{D \in \mathcal{D}_1} D = \bigcup_{D \in \mathcal{D}_2} D \Leftrightarrow \mathcal{D}_1 = \mathcal{D}_2,$$

then $\mathcal{D}$ is a $\overline{d}$*-union-free family*, where $\overline{d}$ represents subsets of size *at most* d. Representing each item x as the set $\rho(x)$, union-free families characterize what is required to find at most d defectives. Recovery is possible in principle if and only if the sets of test case indices form a $\overline{d}$-union-free family, and there are indeed at most d defective items.

We usually write the test cases as an $N \times k$ binary array with rows indexed by test cases and columns indexed by items. Then the result is a $\overline{d}$*-separable matrix*. The union-free condition can be translated to this matrix setting; For every two sets C_1, C_2 of column indices, each of size at most d, there must be at least one row in which the columns indexed by one contain at least one 1 entry while the columns indexed by the other contain only 0 entries.

An easy example arises when $d = 1$ and $k = 2^N - 1$. The columns of a $\overline{1}$-separable matrix are the binary vectors of length N, omitting the all-zero vector. Then every item is tested, and no item appears in precisely the same test cases as any other. Evidently $N \approx \log k$ in this case, and hence a dramatic improvement on individual testing is possible. The logarithmic relationship between the number of test cases needed and the number of items to be tested extends to all fixed values of d, but naturally the constructions to achieve a logarithmic bound are not as simple as when $d = 1$.

If there are more than d defective items, we may mistakenly identify a small set of defective items when the failures actually result from a different set of more than d items. Because defective items always cause a pool test to which they belong to fail, we can examine test cases that differentiate all sets of size at most $d + 1$ rather than d. If there is a set D of at most d items and another set D' of more than d items for which $\rho(D) = \rho(D')$, adjoin any column index from $D' \backslash D$ to D to form D''. Then D'' contains one more column index than does D and $\rho(D) = \rho(D'')$. When the tests are $\overline{(d+1)}$-separable, this cannot occur. Using a $\overline{(d+1)}$-separable matrix, we still cannot guarantee to determine which items are defective when more than d are, but we *can* certify that more than d are defective.

A second important issue is the complexity of characterizing the defective items. Given the set F of failing test cases, there is a unique set D having at most d items for which $\rho(D) = F$. We are faced with the 'inverse' problem of finding D given F. A basic strategy is to examine the test cases that do not fail. Each item participating in a test that passes cannot be defective, and hence we can immediately classify a number of the items as certainly not

defective. Those items that remain, however, may or may not be defective, and we appear to be forced into further testing to determine which are. Instead we can strengthen the original test plan so that once items are determined not to be defective as described, all that remain must be defective. Intuitively, every item that is not defective must appear in at least one test containing no element of the set D of defectives. Let us make this precise. Let $\mathcal{D}$ be a set containing k subsets of $\{1, \ldots, N\}$. If for every $\mathcal{D}_1 \subseteq \mathcal{D}$ with $|\mathcal{D}_1| = d$, and every $D \in \mathcal{D}$,

$$D \subseteq \bigcup_{D \in \mathcal{D}_1} D \Leftrightarrow D \in \mathcal{D}_1,$$

then $\mathcal{D}$ is a *d-cover-free family*. When written as an $N \times k$ matrix, the result is a *d-disjunct matrix*. Equivalently, an $N \times k$ binary matrix is d-disjunct if for every set D of d column indices and every column index $x \notin D$, there is at least one row in which column x contains a 1 but all columns in D contain 0. Then when D is indeed the set of defectives, the required row is the proof that x is not. The simple recovery strategy of examining tests that do not fail then efficiently determines precisely the set of defective items (when there are at most d of them).

Group testing is not widely used in software testing; the simplifying assumptions are too restrictive. Nevertheless it lays a foundation for proceeding to more complicated systems. We consider one extension next.

2.2 Learning a One-Term Boolean Function

Now we treat a related problem, discussed in [8,23]. Suppose that there are k attributes; each is modeled by a logical or boolean variable to indicate whether the attribute is present or absent. Among the 2^k possible assignments of attributes to an test, some have a positive outcome, some a negative one. As an example, in access control schemes, certain combinations of attributes may enable access while others do not. The collection of all positive outcomes (or, if preferred, negative ones) can be expressed as a boolean function of the variables using conjunction, disjunction, and negation. Such a function is a type of *classifier*. A classifier can typically be written in many equivalent ways, so we consider writing the formula as a disjunction of *terms* or *clauses*, in which each term is a conjunction of *literals*, and each literal is a variable or a negated variable.

In computational learning theory, a central question is to learn the classifier function ϕ given evaluations of the function for different attribute assignments. Again a test case assigns boolean values to each of the k attributes, and a pass or fail outcome is observed. And as before, restrictions must be placed on the function ϕ in order to learn the function in 'few' tests. When the classifier function involves only one term, in which no variable is negated, the problem is one of combinatorial group testing. Provided that the term involves at most d variables, a test plan from a $\overline{d}$-separable matrix can find it in principle, and one from a d-disjunct matrix can find it efficiently. But this is a very limited set of classifier functions.

If there is a single term that involves at most d literals, combinatorial group testing techniques do not suffice, for now an item may impact the result through its absence rather than its presence. Nevertheless the idea is the same. Call a term with $\ell \leq d$ literals a *$\overline{d}$-term*. A $\overline{d}$-term is an assignment of truth values to the ℓ attributes, according to whether the corresponding variable is negated or not.

Again form an $N \times k$ array in which the entries are each 0 (false) or 1 (true). A $\overline{d}$-term is a selection of $\ell \leq d$ column indices $(c_1, \ldots, c_\ell)$, and a value $e_i \in \{0, 1\}$ for each $1 \leq i \leq \ell$. A row *covers* the $\overline{d}$-term if there is some row in which the entry in column c_i is e_i for each $1 \leq i \leq \ell$. Indeed for each $\overline{d}$-term D we can write $\rho(D)$ to be the set of row indices for rows in which D is covered.

When for some $\overline{d}$-term D we have $\rho(D) = \emptyset$, the term D for the classifier cannot be revealed. Hence simply discovering whether or not some $\overline{d}$-term affects the classification, we must cover all of them. Again we make this precise.

A *d-universal* matrix is an $N \times k$ array in which for every set $\{c_1, \ldots, c_d\}$ of distinct column indices and every vector $(e_1, \ldots, e_d) \in \{0, 1\}^d$, there is at least one row in which column c_i contains entry e_i, for each $1 \leq i \leq d$. Although we are concerned with terms with $\ell < d$ as well, each participates in $2^{d-\ell}$ terms on d literals and hence must also be covered. The *strength* of the matrix is the largest d for which it is d-universal. Universal matrices of

strength d are necessary and sufficient to reveal the presence of at least one $\overline{d}$-term in the classifier function, but as before the presence of such a $\overline{d}$-term does not ensure that we can characterize which term is involved. Despite the much stronger requirements on universal matrices than on disjunct or separable ones, when d is fixed the growth rate in the number of tests as a function of k continues to be $O(\log k)$. We revisit this later when we consider the generalization to covering arrays.

Following the generalization from union-free to cover-free families (or separable to disjunct matrices), this concern can again be addressed. Suppose that D is a $\overline{d}$-term, D' is a different term with any number of literals, and $\rho(D) = \rho(D')$. If $D \neq D'$, they cannot have the same support (set of variables), so suppose without loss of generality that there is a variable v in D' but not D. Conjoin v to D as the negation of the literal in D' to form a $\overline{(d+1)}$-term D''. A row that covers D'' covers D but not D'. It follows that using the rows of a $(d+1)$-universal matrix as test cases, not only can the set $\rho(D)$ of failures only be consistent with D, but D can be efficiently recovered by ruling out all $\overline{d}$-terms that are covered in passing tests.

When the support of the classifier function involves at most d variables, the function itself can nonetheless involve many terms. The extension to handling all functions supported by at most d variables is immediate.

The extension to functions with more than one term is not straightforward. Although it remains true that any $\overline{d}$-term in the classifier function must result in a positive test, and therefore will be revealed if present, a universal matrix of strength d or $d+1$ can no longer guarantee to characterize or isolate it. We return to this issue later.

2.3 Compressive Sensing

A k-bit *signal* is transmitted over a binary communications channel. Our objective is to make measurements of the channel as the communication is underway, and from the measurements to recover the intended transmission. Although random noise occurs during transmission, we suppose that the noise is small compared to the signal, and that error detection and correction incorporates sufficient redundancy to repair the (hopefully few) transmission errors. As before, if each of the 2^k signals available might be transmitted, our measurements must differentiate among them all. The idea of *compressive sensing* or *compressive sampling* is to determine measurement and recovery strategies when the signal is known to have a sparse representation. If measurement is obtained by "pooling" bits in the transmission and recording whether or not each pool contains a 1, this is a combinatorial group testing problem. Compressive sensing adds the notion that measurement involves forming a linear combination of the entries in each pool.

We begin by describing a specific framework. An *admissible signal* of *dimension* n is a vector in $\mathbb{R}^n$ which is known *a priori* to be taken from a given set $\Phi \subseteq \mathbb{R}^n$. A *measurement matrix* A is a matrix from $\mathbb{R}^{m\times n}$. *Sampling* a signal $\mathbf{x} \in \mathbb{R}^n$ is computing the product $A\mathbf{x} = \mathbf{b}$. Once sampled, *recovery* involves determining the unique signal $\mathbf{x} \in \Phi$ that satisfies $A\mathbf{x} = \mathbf{b}$ using only A and $\mathbf{b}$. If $\Phi = \mathbb{R}^n$, recovery can be accomplished only if A has rank n, and hence $m \geq n$. However for more restrictive admissible sets Φ, recovery can sometimes be accomplished when $m < n$. Given a measurement matrix A, define an equivalence relation $\equiv_A$ so that for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, we have $\mathbf{x} \equiv_A \mathbf{y}$ if and only if $A\mathbf{x} = A\mathbf{y}$. If, for every equivalence class P under $\equiv_A$, the set $P \cap \Phi$ contains at most one signal, then recovery is possible in principle. Because $A\mathbf{x} = A\mathbf{y}$ ensures that $A(\mathbf{x} - \mathbf{y}) = \mathbf{0}$, this can be stated more simply. The *null space* of A, $N(A)$, is the set $\{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{0}\}$. An equivalence class P of $\equiv_A$ can be represented as $\{\mathbf{x} + \mathbf{y} : \mathbf{y} \in N(A)\}$ for any $\mathbf{x} \in P$. Hence recoverability is equivalent to requiring that, for every signal $\mathbf{x} \in \Phi$, there is no $\mathbf{y} \in N(A)\backslash\{\mathbf{0}\}$ with $\mathbf{x} + \mathbf{y} \in \Phi$.

To apply these observations, a reasonable *a priori* restriction on the signals to be sampled is identified, suitable measurement matrices with $m \ll n$ are formed, and a reasonably efficient computational strategy for recovering the signal is provided. A signal is t-*sparse* if at most t of its n coordinates are nonzero. The recovery of t-sparse signals is the topic of *compressive sensing*. An admissible set of signals Φ has *sparsity* t when every signal in Φ is t-sparse. An admissible set of signals Φ is t-*sparsifiable* if there is a full rank matrix $B \in \mathbb{R}^{n\times n}$ for which $\{B\mathbf{x} : \mathbf{x} \in \Phi\}$ has sparsity t. We assume throughout that when the signals are sparsifiable, a change of basis B is applied so that the admissible signals have sparsity t.

A measurement matrix *has* (ℓ_0, t)*-recoverability* when it permits exact recovery of all t-sparse signals. A basic problem is to design measurement matrices with (ℓ_0, t)-recoverability where m is small relative to n, but in such a way that recovery can be accomplished efficiently. Suppose that matrix A has (ℓ_0, t)-recoverability. Then given A and $\mathbf{b}$, recovery of the signal $\mathbf{x}$ can be accomplished in principle by solving the ℓ_0-minimization problem $\min\{||\mathbf{x}||_0 : A\mathbf{x} = \mathbf{b}\}$. This can be done exactly when the (ℓ_0, t)*-null space condition* holds: The null space $N(A)\backslash\{\mathbf{0}\}$ contains no $(2t)$-sparse vector. Recovery remains difficult, and the best available methods resort to an enumerative strategy, essentially listing the possible supports of signals from fewest nonzero entries to most. For each, reduce A and $\mathbf{x}$ to A' and $\mathbf{x}'$, respectively, by eliminating coordinates in the signal assumed to be zero.

Chen et al. [10,24] instead solve the ℓ_1-minimization problem $\min\{||\mathbf{x}||_1 : A\mathbf{x} = \mathbf{b}\}$, using standard linear programming techniques. For this to be effective, it is necessary that for each t-sparse signal $\mathbf{x}$, $\mathbf{x}$ is the unique solution to $\min\{||\mathbf{z}||_1 : A\mathbf{z} = A\mathbf{x}\}$. This property is (ℓ_1, t)-recoverability. For $\mathbf{y} \in \mathbb{R}^n$ and $C \subset \{1, \ldots, n\}$, define $\mathbf{y}_{|C} \in \mathbb{R}^n$ to be the vector such that $(y_{|C})_\gamma = y_\gamma$ if $\gamma \in C$ and $(y_{|C})_\gamma = 0$ otherwise. A necessary and sufficient condition for (ℓ_1, t)-recoverability is: Whenever $\mathbf{y} \in N(A)\backslash\{\mathbf{0}\}$ and every $C \subset \{1, \ldots, n\}$ with $|C| = t$, $||\mathbf{y}_{|C}||_1 < \frac{1}{2}||\mathbf{y}||_1$ (see [24,69]).

The contrast with the group testing and learning approaches arises from the measurement of real-valued factors rather than binary factors; the measurement as a linear combination of main effects rather than as a union of them; and the resulting recovery from real-valued measurements.

2.4 Statistical Design of Experiments

In compressive sensing applications, the signal $\mathbf{x}$ is often taken to be a sequence in $\{0, 1\}^n$ or $\{-1, 1\}^n$, while the measurement matrix contains arbitrary reals. Most often, statistical design of experiments [50] is concerned with the magnitude of the entries in the 'signal', and hence recovers a linear model for a response in $\mathbb{R}^n$ using measurement matrices (*design matrices*) with entries from a small finite set such as $\{0, 1\}$ or $\{-1, 1\}$. When a factor has s levels, the design matrix represents $s - 1$ of the factor's levels as a column. When the columns of the design matrix are indexed by the factors' levels, the objective is to determine the effect of each factor on the response by producing a *linear model*, a linear combination of factor levels. If the design matrix has fewer rows than columns, it must happen that some column C (factor's level) can be written as a linear combination of other columns $\mathcal{C}$. When this occurs, C is *confounded* with $\mathcal{C}$, and any effect of C can equally well be attributed to $\mathcal{C}$. In general, the *alias structure* of a design matrix for main effects contains all pairs of distinct sets of column indices $\{(\mathcal{C}, \mathcal{C}')$: a nonzero linear combination of columns in $\mathcal{C}$ is equal to a (possibly different) linear combination of columns in $\mathcal{C}'\}$.

The alias structure can only be empty when the design matrix has row rank equal to its number of columns, and hence the determination of a linear model requires a number of tests at least equal to the number of factor levels minus the number of factors. When equal, the design is *saturated*. Given *a priori* information on the maximum number t of factor levels that can affect the response, if a set of t or fewer columns is aliased with a set of $t + 1$ or more, only the former can be the cause of the effect on the response. Hence the alias structure can be simplified by removing all pairs of sets in which one has more than t column indices. *Supersaturated designs* [13] address this situation, and employ a number of tests that is in general insufficient to estimate all main effects. The machinery of (ℓ_0, t)-recoverability from compressive sensing could be brought to bear here; in the design of experiments literature, more often experimenters attempt to select design matrices which perform well according various alphabet optimality measures [38].

When building statistical models, a further option arises. One can consider the set of all possible linear models and select among them; this is done using the *search linear model* [30]. We do not explore this here, because it focuses on measurement rather than on fault location.

2.5 Towards Interactions

Each of the approaches outlined so far is concerned with the detection of the significance of choosing a level for a factor on a response. For fault characterization, each informs us concerning the effect of the failure of one factor's

level on faults in response to those. The methods vary in terms of what type of response can arise and what type of measurement can be done. But their differences are few in comparison with their similarities. In each case, the observation that few factor levels are significant is used to reduce the number of measurements taken. And in each the objective is to characterize the response, i.e., to determine exactly which factor levels are implicated.

We have treated these situations first, because they concern the simplest case in which factor levels do not interact significantly. However, fault characterization for large software systems must address interactions. Of the methods discussed so far, only statistical design of experiments makes a serious attempt to treat interactions. In that setting, the following are often observed:

Effect Hierarchy Lower strength interactions tend to have larger effects on the response than do higher strength ones.

Effect Heredity When an interaction of strength t has a significant effect on the response, *weak* effect heredity states that at least one lower strength interaction contained in it also tends to a significant effect while *strong* effect heredity says that all tend to.

Effect Sparsity The number of interactions having a significant effect on the response tends to be relatively small.

None of these is a hard and fast rule, and none tells us the strength of testing that might be needed. They indicate, however, that in the absence of specific knowledge about the system, testing should focus on failures that result from faults caused by interactions of small strength.

3 Combinatorial Coverage and Covering Arrays

To see the result of an interaction that may cause a failure, some test must include it. Indeed when the presence of a specific interaction in a test ensures a failure, our first task simply asks that each interaction of interest appear in at least one test. This is the province of *combinatorial testing* [32,34,40,41,54]. This provides an economical means to determine whether the system has any interaction of choices resulting in a failure.

We develop a formal combinatorial framework. There are k factors $F_1, \dots, F_k$. Each factor F_i has a set of s_i possible values (*levels*) $S_i = \{v_{i1}, \dots, v_{is_i}\}$. A *test* is an assignment of a level from S_i to F_i for each i with $1 \le i \le k$. A test, when executed, can *pass* or *fail*. For any t-subset $I \subseteq \{1, \dots, k\}$ and levels $\nu_i \in S_i$ for $i \in I$, the set $\{(i, \nu_i) : i \in I\}$ is a *t-way interaction*, or an interaction of *strength* t. Thus a test on k factors contains (*covers*) $\binom{k}{t}$ interactions of strength t. A *test suite* is a collection of tests; the *outcomes* are the corresponding set of pass/fail results.

Let $A = (a_{rc})$ be an $N \times k$ array in which the entries in the ith column are from S_i. A t-way interaction $\{(c_i, \nu_i) : 1 \le i \le t, \nu_i \in S_{c_i}\}$ is *covered* in row r of A if $a_{rc_i} = \nu_i$ for $1 \le i \le t$. For an interaction T, the set $\rho_A(T)$ is the set of row indices for rows in which T is covered. For a set $\mathcal{T}$ of interactions, $\rho_A(\mathcal{T}) = \cup_{T \in \mathcal{T}} \rho_A(T)$.

Let $\mathcal{I}_t$ be the set of all t-way interactions for an array, and let $\overline{\mathcal{I}_t}$ be the set of all t-way interactions of strength *at most* t. Consider a t-way interaction $T \in \overline{\mathcal{I}_t}$ of strength less than t. Any t-way interaction T' of strength t that contains T necessarily has $\rho(A, T') \subseteq \rho(A, T)$. A subset $\mathcal{T}'$ of interactions in $\overline{\mathcal{I}_t}$ is *independent* if there do not exist $T, T' \in \mathcal{T}'$ with $T \subseteq T'$.

An array A is a *covering array* for a set $\mathcal{T}$ of interactions when, for every $T \in \mathcal{T}$, we have $\rho_A(T) \neq \emptyset$; see Table 1. It is a *mixed covering array* when factors may have different numbers of levels, and *uniform* when all factors have the same number. The notation MCA$(N; t, k, (s_1, \dots, s_k))$ is used when there are k factors having $s_1, \dots, s_k$ levels, and all t-way interactions are covered in N tests cases. The simpler notation CA$(N; t, k, v)$ is used when all factors have v levels.

Table 1 Covering arrays

MCA$(N; t, k, (s_1, \dots, s_k))$	$\rho_A(T) \neq \emptyset$ for all $T \in \overline{\mathcal{I}_t}$
CA$(N; t, k, v)$	$\rho_A(T) \neq \emptyset$ for all $T \in \overline{\mathcal{I}_t}$ and $v = s_1 = \cdots = s_k$

Based on classification of real systems [44,45] and subsequent research, coverage typically focuses on t-way interactions with $t \leq 6$ (an effect hierarchy is empirically observed in the systems considered). Covering all interactions of interest is desired; often covering most but not all can be effective [43]. Using covering arrays as test suites is intended to reveal the presence or absence of failures, and hence the presence or absence of faults. There is a very large literature on the construction and existence of covering arrays; here we focus on their uses in *finding* faults.

3.1 Rate of Fault Detection

Given a number of factors k, a number of levels v for each factor, and a strength t, the *minimum size* problem asks for the smallest N for which a $\mathrm{CA}(N; t, k, v)$ exists. This question has been the focus of most of the literature on covering arrays. Consider the application in the determining the presence or absence of interaction faults (see [12,42], for example). Naturally we hope that all tests will be run, and none will reveal a fault. However, if there is a fault, and the tests of the array are run sequentially (i.e., executing the test corresponding to the first row, then the second, and so on), the first test in which a failure is encountered enables us to certify the software as faulty, and to attribute the failure to one or more of the interactions covered in this test. Let us suppose that we have no *a priori* information about which t-way interaction might be faulty, if any. Then the natural objective is to ensure that, after running the ith test, we have covered the largest number of t-way interactions that *any* collection of i tests could cover. The best we might hope for is that this holds for every number i of tests run, until all t-way interactions are covered.

In order to measure the goodness of a test suite at detecting a fault early, let $\mathcal{T}$ be the set of all t-way interactions. For every t-way interaction $T \in \mathcal{T}$, compute the index $\phi(T)$ of the first row of the test suite that covers T. If T is the only faulty interaction, exactly $\phi(T)$ tests are executed in order to detect the presence of a fault. Therefore if every t-way interaction is equally likely to be the faulty one, the expected time to detect the fault is $\sum_{T\in\mathcal{T}} \phi(T)$ divided by the total number of t-way interactions. Denote by $\Lambda(t, (v_1, \ldots, v_k))$ the number of t-way interactions to cover for a test suite of strength t, having k factors with v_i levels for $1 \leq i \leq k$. Then a simple recursion can be used to compute this number: $\Lambda(t, (v_1, \ldots, v_k)) = 0$ if $t > k$; 1 if $t = 0$; and $v_1 \Lambda(t-1, (v_2, \ldots, v_k)) + \Lambda(t, (v_2, \ldots, v_k))$ otherwise.

The sum $\sum_{T\in\mathcal{T}} \phi(T)$ can also be calculated more directly. For each test S_i, $1 \leq i \leq N$, compute the number τ_i of t-way interactions that are covered by S_i but not covered by S_j for any $1 \leq j < i$. Then there are exactly τ_i t-way interactions T for which $\phi(T) = i$. Let $u_i = \sum_{\ell=i}^{N} \tau_\ell$, so that u_i is the number of interactions that are covered in a test numbered i or larger (that is, the number of uncovered interactions before executing the ith test). Because the total number of t-way interactions is $\Lambda(t, (v_1, \ldots, v_k))$, we obtain an explicit formula for the expected time to fault detection, when there is exactly one fault to be found:

$$\frac{\sum_{i=1}^{N} i\tau_i}{\Lambda(t, (v_1, \ldots, v_k))} = \frac{\sum_{i=1}^{N} u_i}{\Lambda(t, (v_1, \ldots, v_k))} \tag{1}$$

The denominator in this ratio is independent of the particular test suite chosen, as is $\sum_{i=1}^{N} \tau_i = \Lambda(t, (v_1, \ldots, v_k))$. Therefore to reduce expected time to fault detection, the only opportunity is to cover more interactions earlier in the test suite; hence we want ‘early coverage’.

When there are multiple faults, one could ask for the time to find the first, or the time to find all. In our context, finding the first is the more natural extension. This can again be easily calculated. If there are s faulty interactions and we have no *a priori* information about their location, the expected number of tests to detect the presence of a fault is

$$\Phi_s = \frac{\sum_{i=1}^{N} \binom{u_i}{s}}{\binom{\Lambda(t, (v_1, \ldots, v_k))}{s}} \tag{2}$$

When no faults are detected during the test execution of a CA($N; t, k, v$) all $N = \Phi_0$ tests are run; when s are present, one expects to run Φ_s to find evidence of at least one fault. In the literature, certifying the presence of a fault and certifying the absence of one are treated as two sides of the same coin. However, Bryce and Colbourn [6] showed that for certain parameters, *no CA*($N; t, k, v$) *minimizes both* Φ_0 *and* Φ_1. See also [7] for methods to improve the rate of fault detection, and [57] for methods that incorporate constraints.

Arguably, covering arrays are directed at certifying the absence of faults, not their presence, because they are optimized for the situations in which all tests are run. Hence comparisons of the ability of combinatorial test suites that cover all interactions with testing methods that are designed to reveal faults early (see, e.g., [3,59]) may not be using the most appropriate combinatorial test suites.

We are primarily concerned with the use of combinatorial testing to characterize the faults, not just to tell us when at least one is present. Consider the small test suite in Table 2, based on a CA(6; 2, 5, 2); tests are run sequentially until a failure is found, with the result shown.

Which interaction can cause the fault? Because a covering array of strength two has been employed, we seek an explanation using 1-way or 2-way interactions. If an interaction fault always results in a test failure for a test that covers the interaction, then we can be sure that no 1-way interaction fault is present, because each appears in at least one of the first three passing tests. Test 4 contains $\binom{5}{2} = 10$ 2-way interactions; of these, two appear in at least one of the first three passing tests and cannot be faulty. So there are eight possible 2-way interactions that could cause the failure, and there is no information about which of them are faulty.

Testing is often divided into *operational testing* in which the presence or absence of faults is explored, and *debug testing* in which the fault is characterized and corrected [28,33]. In this setting, combinatorial testing using covering arrays addresses the operational testing, but typically makes no effort to isolate potential faults beyond their presence in a failing test.

3.2 Restricting the Possible Faults

Further information can be obtained by executing all tests even after a failure is detected. We give an example next. For the CA(6;2,5,2) in Table 3, a response for each test is listed in the adjacent column.

Can we explain these responses? First, we try to locate faults due to main effects (one-way interactions). The second test passes, so all (factor, level) pairs in it are known not to be faulty. Therefore in Table 4a, that considers only the second test, when factor 1 is set to one, the run is not faulty. Similarly, for factors 2, 3, 4, and 5 set to zero,

Table 2 Test suite and responses

Tests	Factors 1	2	3	4	5	Response
1	0	1	1	1	1	Pass
2	1	0	1	0	0	Pass
3	0	1	0	0	0	Pass
4	1	0	0	1	1	Fail
5	0	0	0	0	1	–
6	1	1	0	1	0	–

Table 3 Covering array and responses

Tests	Factors 1	2	3	4	5	Response
1	0	1	1	1	1	Fail
2	1	0	1	0	0	Pass
3	0	1	0	0	0	Fail
4	1	0	0	1	1	Pass
5	0	0	0	0	1	Pass
6	1	1	0	1	0	Pass

Table 4 Locating faults due to main effects

(a) Test 2

Factors	0	1
1		✓
2	✓	
3		✓
4	✓	
5	✓	

(b) All Successful Tests

Factors	0	1
1	✓	✓
2	✓	✓
3	✓	✓
4	✓	✓
5	✓	✓

Table 5 Locating faults due to 2-way interactions

(a) Test 2

Factors	00	01	10	11
1, 2			✓	
1, 3				✓
1, 4			✓	
1, 5			✓	
2, 3		✓		
2, 4	✓			
2, 5	✓			
3, 4			✓	
3, 5			✓	
4, 5	✓			

(b) All Successful Tests

Factors	00	01	10	11
1, 2	✓	{1,3}	✓	✓
1, 3	✓	{1}	✓	✓
1, 4	✓	{1}	✓	✓
1, 5	{3}	✓	✓	✓
2, 3	✓	✓	✓	{1}
2, 4	✓	✓	{3}	✓
2, 5	✓	✓	✓	{1}
3, 4	✓	✓	✓	{1}
3, 5	✓	✓	✓	{1}
4, 5	✓	✓	✓	✓

one, zero, and zero, respectively. This is indicated by a check-mark (✓). Repeating for each one-way interaction for each successful test, no single (factor, level) error accounts for the faults; see Table 4b.

Therefore, to explain the responses we turn to 2-way interactions. Because the second test passes, all 2-way interactions in it are known not to be faulty; Table 5a records the results. Repeating for each 2-way interaction in a test that passes, those interactions not found to pass in this way in Table 5b form a set of *candidate faults*. In this example, there are nine interactions in the set of candidate faults. In Table 5b, these are indicated by giving the indices of the subset of the failed tests in which the interaction is covered. The 2-way interaction $T = \{(1, 0), (2, 1)\}$ has $\rho(T) = \{1, 3\}$, and it is the only 2-way interaction T for which this holds. Hence if there is a single fault, it must be $\{(1, 0), (2, 1)\}$, and we have located the fault.

Our success for one response is not sufficient, however. If only run 1 fails, because $\rho(\{(1, 0), (3, 1)\}) = \{1\} = \rho(\{(2, 1), (3, 1)\})$, there are at least two equally plausible explanations using only a single two-way interaction.

Nevertheless, testing with covering arrays can reduce the set of interactions that might be faulty. See [41] for further discussion of this. See also [9,29,67] for more sophisticated uses of covering arrays to learn a model of the faulty interactions, including in the presence of intermittent and masked faults.

3.3 Coverage is Not Enough

Although covering arrays can reveal the presence of faults, they necessitate a second round of (debug) testing to characterize the faults. Naturally, this is a reasonable testing methodology when the location of the potential faults can be sufficiently restricted. As we have seen, however, if the objective is to find a failure, attribute it to a small candidate set of interaction faults, and pursue further testing on these, it not at all clear that a covering array is the "right kind" of test suite. Indeed the expected time to fault detection is a more informative metric than is the size of the test suite, and these are different.

We develop the combinatorial requirements for determining which interactions are faulty, without the need for a second round of testing.

4 Locating and Detecting Arrays

In order to see that an interaction T causes a fault, a test suite A must have $\rho_A(T) \neq \emptyset$. In other words, A must be a covering array because coverage is necessary. But we have seen that coverage is not enough to find faulty interactions without further testing. We turn to combinatorial group testing, computational learning, compressive sensing, and the design of experiments to understand how they avoid further testing.

The statistical design of experiments targets interactions among factors, and in that sense most closely matches our objectives here. Its focus on measurement of the effects on a numerical response is quite different than ours, however. Because we are primarily concerned with pass/fail outcomes, we are not interested in separating the effects of two interactions that both cause faults. Rather we seek the identity of the faulty interactions. In this sense, fault characterization bears more similarities with combinatorial group testing, computational learning, and compressive sensing.

Each can be viewed as finding one t-way interaction, or as finding t 1-way interactions, but as currently developed do not find multiple interactions each of strength more than one. Hence they do not address our question directly, but they provide the conceptual framework.

In each case, a suite A of tests is executed and responses are collected, which may be pass/fail or insignificant/significant. A set F of test indices specifies the failing responses. The goal in each case is to determine the defective or significant items by determining which sets could yield the failing responses F. The statistical design of experiments extends this to determine not just the items (1-way interactions) but also higher strength interactions.

Although the details differ, the conceptual basis for combinatorial fault characterization is the same as in these better studied topics. We develop a combinatorial basis for fault characterization next.

4.1 The Combinatorial Basis for Fault Characterization

Here we follow the presentation in [20]. Imagine that there is a set $\mathcal{T}$ of faulty interactions. Testing using array A we observe that $\rho_A(\mathcal{T})$ is the set of failing tests, and we wish to recover (characterize) $\mathcal{T}$. Suppose that faults, if present, are caused by interactions of strength at most t (an effect hierarchy assumption). Even when interactions causing faults are independent, it may not be possible to uniquely determine $\mathcal{T}$ given $\rho_A(\mathcal{T})$. For example, if factor F_i has levels $v_{i1}, \ldots, v_{is_i}$ and each of $\{(F_i, v_{ij}) : 1 \leq j \leq s_i\}$ causes a fault, then all tests fail, and there is no way to determine which interactions cause faults. Moreover, there are

$$\tau = 1 + \sum_{0<i_1<\cdots<i_t<k} \left[\prod_{j=1}^{t} s_{i_j} - 1 \right]$$

mutually independent t-way interactions. If any set of t-way interactions can be the causes of the faults, one would need to distinguish among all 2^τ possible sets, requiring at least τ tests. In this case, we encounter again the need for a number of tests linearly related to the number of interactions, as for standard statistical design of experiments. At first, this suggests that a substantial reduction in the number of tests requires one to abandon the determination of the interactions causing faults. If we suppose, however, that d interactions cause faults and want to determine which, then we need only distinguish among $\binom{\tau}{d}$ sets, not 2^τ. To formulate arrays for testing, we therefore assume limits on both the number of interactions causing faults and their strengths. Of course, we must presume not just effect hierarchy and heredity to limit the strengths of interactions considered, but also the sparsity of effects to limit the number of interactions to be found.

As in [19], this leads to a variety of arrays A for testing a system with N tests and k factors having $(s_1, \ldots, s_k)$ as the numbers of levels, defined in Table 6. Each variant specifies the number and strength of the interactions considered. In the notation, d specifies an exact value for the number, while $\overline{d}$ specifies an upper bound of d; similarly t specifies an exact value for the strength, while $\overline{t}$ specifies an upper bound of t. When all factors have the same number of levels v, we can replace $(s_1, \ldots, s_k)$ with v in the notation.

Table 6 Arrays for determining faults

	Locating arrays
(d, t)-LA$(N; k, (s_1, \ldots, s_k))$	$\rho_A(\mathcal{T}_1) = \rho_A(\mathcal{T}_2) \Leftrightarrow \mathcal{T}_1 = \mathcal{T}_2$ whenever $\mathcal{T}_1, \mathcal{T}_2 \subseteq \mathcal{I}_t$, $\lvert\mathcal{T}_1\rvert = d$, and $\lvert\mathcal{T}_2\rvert = d$
$(\overline{d}, t)$-LA$(N; k, (s_1, \ldots, s_k))$	$\rho_A(\mathcal{T}_1) = \rho_A(\mathcal{T}_2) \Leftrightarrow \mathcal{T}_1 = \mathcal{T}_2$ whenever $\mathcal{T}_1, \mathcal{T}_2 \subseteq \mathcal{I}_t$, $\lvert\mathcal{T}_1\rvert \leq d$, and $\lvert\mathcal{T}_2\rvert \leq d$
$(d, \overline{t})$-LA$(N; k, (s_1, \ldots, s_k))$	$\rho_A(\mathcal{T}_1) = \rho_A(\mathcal{T}_2) \Leftrightarrow \mathcal{T}_1 = \mathcal{T}_2$ whenever $\mathcal{T}_1, \mathcal{T}_2 \subseteq \overline{\mathcal{I}_t}$, $\lvert\mathcal{T}_1\rvert = d$, $\lvert\mathcal{T}_2\rvert = d$, and $\mathcal{T}_1$ and $\mathcal{T}_2$ are independent
$(\overline{d}, \overline{t})$-LA$(N; k, (s_1, \ldots, s_k))$	$\rho_A(\mathcal{T}_1) = \rho_A(\mathcal{T}_2) \Leftrightarrow \mathcal{T}_1 = \mathcal{T}_2$ whenever $\mathcal{T}_1, \mathcal{T}_2 \subseteq \overline{\mathcal{I}_t}$, $\lvert\mathcal{T}_1\rvert \leq d$, $\lvert\mathcal{T}_2\rvert \leq d$, and $\mathcal{T}_1$ and $\mathcal{T}_2$ are independent
	Detecting arrays
(d, t)-DA$(N; k, (s_1, \ldots, s_k))$	$\rho_A(T) \subseteq \rho_A(\mathcal{T}) \Leftrightarrow T \in \mathcal{T}$ whenever $T \in \mathcal{I}_t$, $\mathcal{T} \subseteq \mathcal{I}_t$, and $\lvert\mathcal{T}\rvert \leq d$
$(d, \overline{t})$-DA$(N; k, (s_1, \ldots, s_k))$	$\rho_A(T) \subseteq \rho_A(\mathcal{T}) \Leftrightarrow T \in \mathcal{T}$ whenever $T \in \overline{\mathcal{I}_t}$, $\mathcal{T} \subseteq \overline{\mathcal{I}_t}$, $\lvert\mathcal{T}\rvert \leq d$, and $\mathcal{T} \cup \{T\}$ is independent

The definition for a locating array captures the statement that when there are few faulty interactions and each has small strength, every set of failing responses can have at most one corresponding set of faulty interactions. Indeed using a locating array A ensures that there is at most one $\mathcal{T}$ for which $\rho_A(\mathcal{T})$ is a specified set of failed tests, assuming that the permitted number of interactions each having the permitted strength are faulty. How does one recover $\mathcal{T}$ from $\rho_A(\mathcal{T})$? As before, observe that every interaction appearing in a test that passes cannot be faulty. Although many (typically most) interactions are determined not to cause faults in this way, it may nevertheless happen that the effect of an interaction T remains undetected in the presence of a set $\mathcal{T}$ of faulty interactions. This happens precisely when $\rho_A(T) \subseteq \rho_A(\mathcal{T})$ but $T \notin \mathcal{T}$.

This leads to a variant of locating arrays, the detecting arrays. Combinatorial group testing provides the guide to understanding what is needed. The sets of rows in which collections $\mathcal{T}$ appear in a locating array forms a union-free family. We saw earlier that to recover defective items efficiently, one imposes a stronger cover-free requirement on the sets. This is precisely what has been done to define detecting arrays. Using detecting arrays, faulty interactions can be found by simply listing all interactions that appear only within failed tests. Hence they ensure efficient determination of faulty interactions.

We argue that locating and detecting arrays properly belong to combinatorial testing. Indeed they are covering arrays, but they provide information about the location of faults, not just their presence. Despite their fault location capability, as for covering arrays the number of tests grows *logarithmically* in the number of factors when the strength and the maximum number of levels are fixed; for locating and detecting arrays, one needs to assume in addition that the number of faulty interactions is fixed to obtain the logarithmic growth [19]. One might therefore expect that they have been extensively studied, in the same way as covering arrays.

Despite this expectation, the literature on detecting and locating arrays is in its beginning stages. Martínez et al. [48] develop adaptive analogues and establish feasibility conditions for a locating array to exist. In [62,65] the minimum number of rows in a locating array is determined when the number of factors is quite small. In [17] the minimum number of rows for (1, 1)-, ($\overline{1}$, 1)-, (1, $\overline{1}$)-, and ($\overline{1}$, $\overline{1}$)-locating arrays is determined precisely when all factors have the same number of levels. In [46,49] partial results are given for (1, 1)-detecting arrays (there called “Sperner partition systems”). In [19] a construction for locating arrays from higher strength covering arrays is provided. In [16] three recursive constructions for locating arrays of strength two are developed, patterned on recursive constructions for covering arrays.

We expect that locating and detecting arrays can serve as a bridge between combinatorial testing and the statistical design of experiments. The measurement of an interaction depends upon how it is confounded with other interactions [50], while observing the effect of the interaction at all depends on its coverage. Locating and detecting arrays ensure coverage; they also ensure that no two interactions are completely confounded. Hence they address both combinatorial and statistical objectives. It appears natural to strengthen this bridge, by exploiting combinatorial properties of experimental designs in refining the goals of combinatorial testing.

4.2 Probabilistic Methods

Natural methods to analyze and construct locating and detecting arrays use probabilistic arguments and randomized algorithms. Here we focus on the case of (2, 2)-detecting arrays in order to develop the ideas. Suppose that an $N \times k$ array on v symbols is chosen, each entry uniformly at random. What is the probability that it is (2, 2)-detecting?

There are $\tau = \binom{k}{2}v^2$ 2-way interactions. Hence there are $\pi = \binom{\tau}{2}$ pairs of (distinct) 2-way interactions. In order to be (2, 2)-detecting, for every set $\mathcal{T}$ containing two 2-way interactions, and each of the $\tau - 2$ 2-way interactions $T \notin \mathcal{T}$, at least one row must cover T but neither interaction in $\mathcal{T}$. Call $(\mathcal{T}, T)$ an *event*, and say that it is *good* when some row provides covers T but neither member of $\mathcal{T}$, and *bad* when no row does.

Naturally we are interested in the probability that there is no bad event. To determine this, we first group the events according to the distributions of columns and symbols in the pair $\mathcal{T}$ of the event, defining *classes* as follows.

1. $\mathcal{T}$ involves four different columns.
2. $\mathcal{T}$ involves three different columns, with the same symbol in the unique column appearing twice.
3. $\mathcal{T}$ involves three different columns, with different symbols in the unique column appearing twice.
4. $\mathcal{T}$ involves two different columns, with different symbols in both columns.
5. $\mathcal{T}$ involves two different columns, with different symbols in one of the columns, but the same symbol in the other.

Each of the five classes can be further partitioned according to the placement of T with respect to the class of $\mathcal{T}$. Two events $(\mathcal{T}_1, T_1)$ and $(\mathcal{T}_2, T_2)$ have the same *pattern* exactly when there is a permutation of the columns and permutations of the symbols in each column, mapping one to the other. When events have the same pattern, they are in the same class, so patterns refine classes.

For each pattern, we determine how many events have this pattern, and the probability that an event with this pattern is good in a single row whose entries are chosen uniformly at random. To illustrate this, consider the pattern

By convention, the different columns of the diagram represent different columns of the array. Different nodes within a column indicate different symbols appearing in that column. The class is indicated by the two solid lines; in this case the event and pattern have class 3. The dashed line indicates the 2-way interaction T.

How many events have this pattern? First calculate the number of ways to choose the two 2-way interactions in $\mathcal{T}$ so that they have class 3. This is $n_3 = 3\binom{k}{3}v^3(v-1)$. For each choice $\mathcal{T}$ of class 3, in order to obtain the specified pattern, we choose one more of the $(k-3)$ remaining columns, and any of the v symbols in that column; then form pair T by including the already chosen symbol, either from the first or the third column. It follows that the number of events with this pattern is $2n_3(k-3)v$.

Let the *goodness* of a pattern be the probability that an event of this pattern is good in a single row. What is the goodness for an event of our chosen pattern? The 2-way interaction indicated by the dashed line is covered with probability $\frac{1}{v^2}$, but we must ensure that neither 2-way interaction corresponding to the solid lines is also covered. In the second column, $v-2$ symbols lead to a good event, and one leads to a bad event; for the last, we consider the selection in the first column. Then $v-1$ choices in the first column lead to a good event, and one is bad. Hence the goodness is $\frac{1}{v^2}(\frac{v-2}{v} + \frac{v-1}{v^2}) = \frac{v^2-v-1}{v^4}$.

These computations can be done for each pattern of each of the five classes, as shown in Tables 7, 8, 9, 10, and 11. In the table for class i, first the set $\mathcal{T}$ for class i is depicted, and a count n_i of ways that class i arises is given. Then all patterns for this class are depicted, along with the number of events for this pattern, and its goodness.

The counts for classes can be verified in part by noting that $n_1 + n_2 + n_3 + n_4 + n_5 = \pi = \binom{\tau}{2}$. Numbers of events for patterns of class i can be verified in part, for each $1 \leq i \leq 5$, by noting that the numbers of events sum to $n_i(\tau - 2) = n_i(\binom{k}{2}v^2 - 2)$.

The goodness values can be verified using the diagrams. It is striking that goodness varies substantially among events. When $v = 2$, many have the largest goodness $\frac{1}{4}$; however, some have goodness $\frac{2-2}{2^3} = 0$, and hence *cannot* be a good event. Indeed when $v = 2$, no (2, 2)-detecting array can exist [19,48], so there must be such events that

Table 7 Patterns of class 1

Class	Count	
	$n_1 = 3\binom{k}{4}v^4$	
Pattern	**#Events**	**Goodness**
	$n_1\binom{k-4}{2}v^2$	$\frac{((v-1)^2)^2}{v^6}$
	$4n_1(k-4)v$	$\frac{(v^2-1)(v-1)}{v^5}$
	$4n_1(k-4)v(v-1)$	$\frac{v^2-1}{v^4}$
	$2n_1(v-1)^2$	$\frac{v^2-1}{v^4}$
	$4n_1(v-1)$	$\frac{v^2-1}{v^4}$
	$4n_1$	$\frac{(v-1)^2}{v^4}$
	$8n_1(v-1)$	$\frac{v-1}{v^3}$
	$4n_1(v-1)^2$	$\frac{1}{v^2}$
all	$n_1(\binom{k}{2}v^2-2)$	

Table 8 Patterns of class 2

Class	Count	
	$n_2 = 3\binom{k}{3}v^3$	
Pattern	**#Events**	**Goodness**
	$n_2\binom{k-3}{2}v^2$	$\frac{v^3-2v+1}{v^5}$
	$2n_2(k-3)v$	$\frac{v-1}{v^3}$
	$2n_2(k-3)v(v-1)$	$\frac{v^2-1}{v^4}$
	$n_2(k-3)v$	$\frac{(v-1)^2}{v^4}$
	$n_2(k-3)v(v-1)$	$\frac{1}{v^2}$
	$n_2(v-1)^2$	$\frac{1}{v^2}$
	$2n_2(v-1)$	$\frac{v-1}{v^3}$
	n_2	$\frac{v-1}{v^3}$
	$2n_2(v-1)^2$	$\frac{1}{v^2}$
	$2n_2(v-1)$	$\frac{v-1}{v^3}$
	$2n_2(v-1)$	$\frac{1}{v^2}$
all	$n_2(\binom{k}{2}v^2-2)$	

cannot be good. When $v \geq 3$, every event has positive goodness. When $v = 3$, goodness values range from $\frac{16}{729}$ to $\frac{1}{9}$.

This classification can be carried out for any of the variants of locating or detecting arrays, in order to determine a set of patterns of events, with a number of occurrences and goodness for each. In general, suppose that there are m patterns, with numbers of occurrences $r_1(k), \ldots, r_m(k)$ and goodness values $g_1 \ldots, g_m$. (When v is fixed, each number of events is a function of k.) When N rows are chosen at random, the expected number of bad events is

$$B_{v,k,N} := \sum_{i=1}^{m} r_i(k) \times (1 - g_i)^N$$

Using standard probabilistic arguments [2], when this expectation is less than 1, an array having no bad events must exist. Unlike the application to covering arrays, the different goodness values complicate a general analysis. Nevertheless for any type of locating or detecting array with a specified number of symbols, the expectation bound given yields a lower bound on k as a function of N, or an upper bound on N as a function of k, sufficient to ensure the existence of an array of the desired type. In each case for which all goodness values are positive, N is $\Theta(\log k)$.

Table 9 Patterns of class 3

Class	Count	
	$n_3 = 3\binom{k}{3}v^3(v-1)$	
Pattern	**#Events**	**Goodness**
	$n_3\binom{k-3}{2}v^2$	$\frac{v^3-2v}{v^5}$
	$2n_3(k-3)v$	$\frac{v^2-v-1}{v^4}$
	$2n_3(k-3)v(v-1)$	$\frac{v^2-1}{v^4}$
	$2n_3(k-3)v$	$\frac{v-1}{v^3}$
	$n_3(k-3)v(v-2)$	$\frac{1}{v^2}$
	$n_3(v-1)^2$	$\frac{1}{v^2}$
	$2n_3(v-1)$	$\frac{v-1}{v^3}$
	n_3	$\frac{v-2}{v^3}$
	$2n_3(v-1)$	$\frac{1}{v^2}$
	$2n_3(v-2)$	$\frac{1}{v^2}$
	$2n_3$	$\frac{v-1}{v^3}$
	$2n_3(v-1)$	$\frac{v-1}{v^3}$
	$2n_3(v-1)(v-2)$	$\frac{1}{v^2}$
all	$n_3(\binom{k}{2}v^2-2)$	

Table 10 Patterns of class 4

Class	Count	
	$n_4 = \frac{1}{4}k(k-1)v^2(v-1)^2$	
Pattern	**#Events**	**Goodness**
	$n_4\binom{k-2}{2}v^2$	$\frac{v^2-2}{v^4}$
	$4n_4(k-2)v$	$\frac{v-1}{v^3}$
	$2n_4(k-2)v(v-2)$	$\frac{1}{v^2}$
	$n_4(v-2)^2$	$\frac{1}{v^2}$
	$4n_4(v-2)$	$\frac{1}{v^2}$
	$2n_4$	$\frac{1}{v^2}$
all	$n_4(\binom{k}{2}v^2-2)$	

Table 11 Patterns of class 5

Class	Count	
	$n_5 = \frac{1}{2}k(k-1)v^2(v-1)$	
Pattern	**#Events**	**Goodness**
	$n_5\binom{k-2}{2}v^2$	$\frac{v^2-2}{v^4}$
	$2n_5(k-2)v$	$\frac{v-1}{v^3}$
	$n_5(k-2)v$	$\frac{v-2}{v^3}$
	$n_5(k-2)v(v-1)$	$\frac{1}{v^2}$
	$n_5(k-2)v(v-2)$	$\frac{1}{v^2}$
	$n_5(v-2)$	$\frac{1}{v^2}$
	$2n_5(v-1)$	$\frac{1}{v^2}$
	$n_5(v-1)(v-2)$	$\frac{1}{v^2}$
all	$n_5(\binom{k}{2}v^2-2)$	

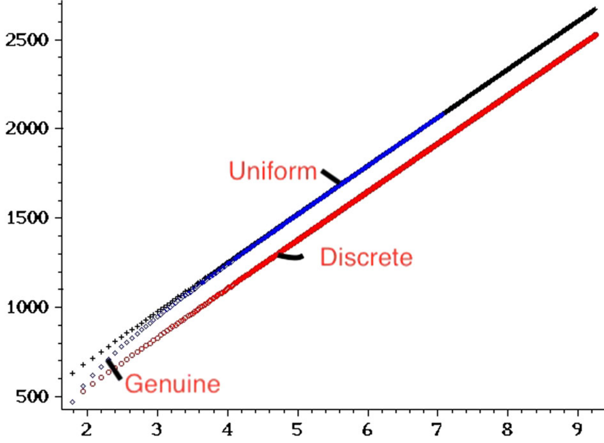

Fig. 1 Genuine, uniform, and discrete bounds: (2, 2)-detecting arrays, $v = 3$

For (2, 2)-detecting arrays, asymptotically all events (as $k \to \infty$) have the lowest goodness value. Rather than using the genuine goodness values, a uniform bound is obtained by treating all goodness values as equal to the lowest.

One can improve the bound by choosing one row at a time, ensuring for each row selection that the number of bad events does not exceed the expectation for a random array with this number of rows; this strategy is developed for covering arrays in [5,14]. We treat this using uniform goodness values. The key observation is that, after each row is selected, although the expected number of bad events may be fractional, the actual number is an integer. Replacing the fractional expectation by its integer floor after each row is selected may reduce the number of rows used. We call this the *discrete* bound.

In Fig. 1 we compare the genuine, uniform and discrete bounds for (2, 2)-detecting arrays with $v = 3$. Shown vertically is the number of rows. Horizontally, graphs are indexed by the natural logarithm of the number of columns. The uniform bound yields the largest number of rows; the genuine bound, although somewhat better when the number of columns is quite small, appears to converge quickly to the uniform bound. On the other hand, except when the number of columns is very small, the discrete bound yields a substantial improvement throughout this range, despite using pessimistic goodness values. (Although in Fig. 1 the bounds appear to be 'close', in the range

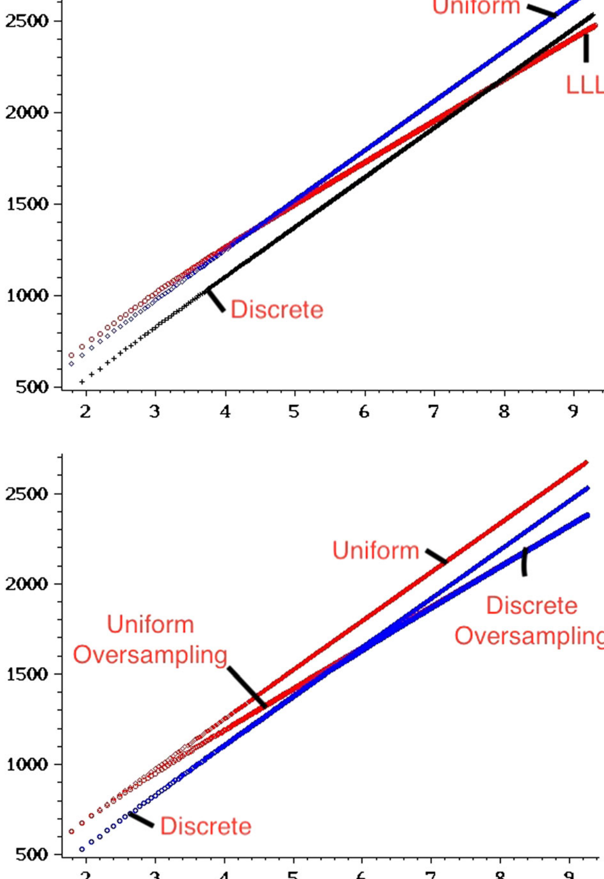

Fig. 2 Uniform, discrete, and LLL bounds: (2, 2)-detecting arrays, $v = 3$

Fig. 3 Uniform and discrete: the effect of oversampling

examined the discrete bound yields nearly twice as many columns as the uniform bound for the same number of rows.)

When applied to covering arrays, these techniques can be improved upon by using the symmetric version of the Lovász Local Lemma (LLL) [18,27,31,58]. Because goodness values vary when considering locating and detecting arrays, the symmetric version of LLL does not apply for the genuine goodness values. However, each event nonetheless has limited dependence on other events, and the *asymmetric* version of LLL [2,26] does apply. Moreover, the symmetric version of LLL does apply when using the uniform goodness values. The technical details are involved, but standard, so we omit them here. In Fig. 2 we compare the LLL bound with the uniform and discrete bounds, again for (2, 2)-locating arrays with $v = 3$. For few columns, the discrete bound yields fewer rows than the uniform bound, which in turn yields fewer than LLL. However, the LLL bound overtakes both, yielding the fewest rows for larger numbers of columns.

Finally we discuss a bound based on oversampling (as in [18]; also called post-processing [4]). Consider $B_{v,\kappa,N}$, the expected number of bad events when N rows are chosen on κ columns. There is some array with at most $\lfloor B_{v,\kappa,N} \rfloor$ bad events. Naturally if $\lfloor B_{v,\kappa,N} \rfloor \geq 1$, no guarantee that an array of the desired type exists having κ columns. Nevertheless, for each bad event $(\mathcal{T}, T)$ we can remove one of the columns of T to avoid the bad event. Hence there must be an array of the desired type having N rows and $\max(0, \kappa - \lfloor B_{v,\kappa,N} \rfloor)$ columns. Indeed to get k columns we can determine whether there exists a $\kappa \geq k$ so that $\lfloor B_{v,\kappa,N} \rfloor \leq \kappa - k$; when there is, we must have an array of the desired type with at least k columns. This *oversampling* strategy can be applied in conjunction with any of the genuine, uniform, or discrete bounds. In Fig. 3 we show the improvements obtained by oversampling for the uniform and discrete bounds.

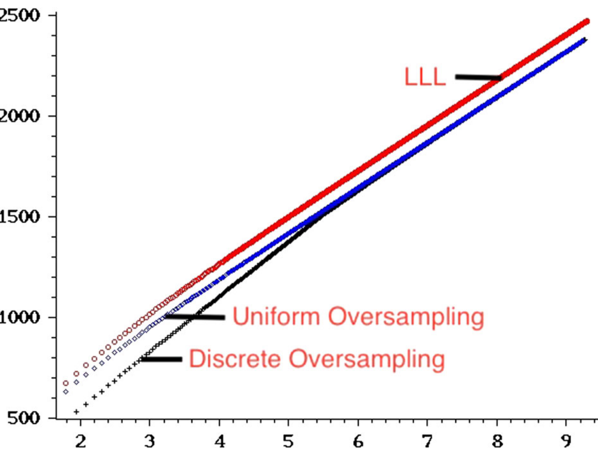

Fig. 4 Uniform and discrete bounds with oversampling versus the LLL bound

While oversampling has a more dramatic effect on the uniform bound, it also makes a useful improvement to the more accurate discrete bound. Oversampling does not appear to apply directly to the LLL bounds, because the Lovász Local Lemma does not bound the number of bad events (except when it is zero).

Therefore it is of interest to compare the LLL bound with those obtained from oversampling. This is done in Fig. 4. In this range, LLL is consistently outperformed by both the uniform and the discrete bounds with oversampling. Perhaps more surprising is that, although the discrete bound is initially much better than the uniform one, they appear to converge as k increases. One explanation is that the discrete bound makes most improvements when the expected number of bad events is small; oversampling, rather than examining arrays with very few bad events, instead removes columns. This reduces the relative advantages of discrete over uniform.

4.3 Computational Methods

It can safely be said that at the present time, producing a locating or detecting array with few tests for a specific set of parameters is challenging, and is the subject of current research. Few testing problems have so far employed locating and detecting arrays [1,21,60].

For covering arrays, AETG [14] popularized techniques that add one test at a time, greedily selecting a test to (attempt to) maximize the number of newly covered interactions; such methods can lead to a number of tests that is logarithmic in the number of factors [5]. The adaptation of such methods to locating arrays use the Discrete bound, applying efficient derandomization of the techniques of Stein [63], Lovász [47], and Johnson [37] Preliminary results using this strategy are reported in [61]. The LLL bound can also be derandomized using the techniques of Moser and Tardos [51]; again, preliminary results are given in [61].

For large systems, one-test-at-a-time methods involve tracking a very large number of interactions. For this reason, one-factor-at-a-time methods have been widely used. IPO [64] introduced such methods, which form the basis of the ACTS system (see [41]). Again here, vertical and horizontal growth would need to address coverage in differing sets of tests. The issue is (perhaps) complicated by the introduction of new interactions as new factors are added; if information about the tests in which all interactions appear must be maintained, then the storage advantage over the one-test-at-a-time method disappears. Can this be overcome?

In [53] a technique is developed for reducing the number of tests in an existing covering array by exploiting redundancy in the coverage. A naive application of this approach to locating arrays results in failure to distinguish sets of tests in which interactions appear, and hence the post-optimization method as it stands is ineffective for locating arrays. Nevertheless local modifications of a locating array can sometimes be made so as to ensure that a test becomes unnecessary. What is needed is not only a strategy for finding such modifications, but also a demonstration that they work in practice (if indeed they do).

When more extensive computation is possible, metaheuristic search techniques such as simulated annealing (for example, [66]) have proved to be very effective for covering arrays. Their adaptation for locating and detecting arrays appears to be relatively straightforward; the primary issue is how to measure the proximity of an array to the locating array desired, because the array may fail for lack of coverage, or because of the inability to distinguish interactions. We expect that simulated annealing can address these concerns well. Related efforts on metaheuristic search for $(1, 2)$-locating arrays appears in [39,52]. See also [70].

5 Diversity

We have seen that in order to locate significant interactions, it is not sufficient to cover them (although it is necessary). The sets of tests in which interactions are covered must be different in order to distinguish the effects of the interactions. This objective does not appear to have been addressed explicitly in the combinatorial testing methods for software. However, we believe that it relates closely to objectives that have been extensively studied. We discuss the connections next.

A covering array for a specified strength t may fail to cover all τ-way interactions for $\tau > t$. Consequently, Dalal and Mallows [22] suggest employing test suites with high diversity. The τ-*diversity* of an $N \times k$ array is the ratio of the number of (distinct) τ-way interactions that are covered to the total number of τ-way interactions covered in tests, $N\binom{k}{\tau}$. Larger τ-diversity can result from reducing the number of tests, or by covering as many τ-way interactions as possible in the available number of tests. Increasing the τ-diversity improves the cost-effectiveness of covering interactions.

Chen et al. [11] discuss the connection between "adaptive random testing" and diversity. They argue that this and related measures of diversity enhance the ability to find faults, and that such test suites can be effectively generated using a one-test-at-a-time strategy making decisions based on test distances. Hartman and Raskin [35] suggest ensuring coverage of t-way interactions, while maximizing τ-diversity for $\tau > t$. These observations are borne out in [55], where the rate of fault detection (in a simplified abstract model) is shown to improve with higher diversity.

Large diversity still targets determining the presence of faults, not identification of specific faults. Nevertheless, locating arrays indirectly address diversity. To see this, consider two t-way interactions $T_1 = \{(f_{i_1}, v_{i_1}), \cdots (f_{i_t}, v_{i_t})\}$ and $T_2 = \{(f_{j_1}, v_{j_1}), \cdots (f_{j_t}, v_{j_t})\}$. They are *consistent* if whenever $i_a = j_b$, we have $v_{i_a} = v_{j_b}$ (i.e., they can appear together in a test). When consistent, $T = T_1 \cup T_2$ is a τ-way interaction for $t < \tau \leq 2t$. Now suppose that $\rho(T_1) = \rho(T_2)$, so that the array is not $(1, t)$-locating. Then $\rho(T_1) = \rho(T)$. It follows that among the τ-way interactions consistent with T_1, only one is covered. Hence, inability to locate a t-way interaction that is covered more than once appears to reduce τ-diversity for $\tau > t$. Conversely, increasing τ-diversity appears to serve as a useful surrogate for enhancing the combinatorial ability to locate.

While diversity focuses on making tests as different as possible, locating arrays focus on distinguishing among the sets of tests in which interactions appear. The relationship between these two related but different objectives merits study.

6 Summary

We reiterate the main combinatorial observations made throughout the paper.

In order to witness a fault we must encounter a failure. This is not controversial, but it has an important implication: *In order to witness an interaction fault, some test case must cover it.* This directed us to the well-studied area of combinatorial testing.

To gain information about the location of at least one fault, we must consider at least one test failure. One test failure ensures that a fault is present, but only narrows down the set of interactions that may cause the fault. If further testing can be done, this points to finding the first failure quickly, and hence to minimizing the expected time to fault detection. We have shown that minimizing the expected time to first failure is different from minimizing the

size of a test suite that covers all interactions. Despite this, the two objectives are almost always conflated in the literature.

To distinguish one set of faults from another, the sets must result in failures in different sets of tests. This observation brings to bear the connections with combinatorial group testing, computational learning, compressive sensing, and statistical design of experiments. Indeed in the latter context, we ask that sets of interactions not be aliased with other sets, to avoid their effects being confounded. Unless exhaustive testing is done, some aliasing is inevitable. Nevertheless combinatorial testing can ensure that no two "small" sets of low strength interactions are confounded by appearing in the same sets of tests. This provides the combinatorial framework for locating arrays.

An efficient recovery strategy to determine the faults requires more. While compressive sensing and design of experiments can rely on the algebra of linear equations to effect fast recovery, different techniques are needed for combinatorial testing. Patterned on earlier work in combinatorial group testing, the combinatorial requirements underlying locating arrays can be strengthened to define detecting arrays, for which recovery is efficient and easily implemented.

7 Practical Limitations and Implications

We have concentrated on the underlying theory, but outline questions for testing in practice.

First, when covering arrays are used to limit the set of potential faults, which covering arrays should be used? Although small test suites appear to be needed, little research has been done on finding test suites that reduce the expected time until the first fault is detected. When a failure in testing triggers a change to a debug mode, techniques to ensure that this transition is expected to occur early (when it occurs at all) are needed. Moreover, the smaller the set of fault candidates, the easier the debug testing should be. It would be natural to proceed with testing until few fault candidates remain, but no work appears to address this directly except for methods that characterize a fault completely.

Secondly, we have made numerous assumptions that may not hold in practice. We mention a few. Test outcomes are assumed here to be reliable, but in many applications both false positives and false negatives may arise because of noise and environmental conditions. To handle rare and intermittent incorrect outcomes, tests can be replicated and outlier outcomes discounted. Alternatively, one can require that the sets of rows covering different sets of rows disagree in more than one row; extensions of locating and detecting arrays with increased separation among the sets of rows are natural [61].

A further problem arises when a faulty interaction can have its effect masked by another set of interactions (*inhibitors*), so that the fault does not produce a failure. (See [68].) Again unless substantial information is known about the manner in which interactions can mask others, little can be done. However, either increasing diversity or imposing locating conditions has the effect of ensuring that each set of interactions appears in the absence of each other set, so a failure is still observed despite the presence of certain sets of inhibitors. What may be lost is the observation of a failure in each test covering the faulty interaction, so fault characterization cannot proceed in the manner proposed here.

Perhaps the most serious concern is that in designing a combinatorial test suite, certain tests may be impossible to carry out, or they may be so costly that the testing budget cannot support them. A *constraint* is an interaction that cannot be tested [56]. Substantial effort has been invested in producing covering arrays that violate none of a set of constraints (see [15,41]). In [36] locating and detecting arrays in the presence of constraints are formulated.

Finally, a practical limitation arises from the paucity of effective tools to construct locating and detecting arrays. Each of these can be better understood by computing, for every combinatorial test suite, all pairs of small sets of low strength interactions that arise in the same set of rows.

References

1. Aldaco, A.N., Colbourn, C.J., Syrotiuk, V.R.: Locating arrays: a new experimental design for screening complex engineered systems. SIGOPS Oper. Syst. Rev. **49**(1), 31–40 (2015)
2. Alon, N., Spencer, J.H.: The Probabilistic Method. Wiley-Interscience Series in Discrete Mathematics and Optimization, 3rd edn. Wiley, Hoboken (2008)
3. Bach, J., Schroeder, P.J.: Pairwise testing: a best practice that isn't. In: 22nd Annual Pacific Northwest Software Quality Conference, pp. 180–196 (2004)
4. van den Berg, E., Candès, E., Chinn, G., Levin, C., Olcott, P.D., Sing-Long, C.: Single-photon sampling architecture for solid-state imaging sensors. Proc. Natl. Acad. Sci. **110**(30), E2752–E2761 (2013)
5. Bryce, R.C., Colbourn, C.J.: A density-based greedy algorithm for higher strength covering arrays. Softw. Test. Verif. Reliab. **19**, 37–53 (2009)
6. Bryce, R.C., Colbourn, C.J.: Expected time to detection of interaction faults. J. Comb. Math. Comb. Comput. **86**, 87–110 (2013)
7. Bryce, R.C., Colbourn, C.J., Kuhn, D.R.: Finding interaction faults adaptively using distance-based strategies. In: 2011 18th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS), pp. 4–13 (2011)
8. Bshouty, N.H., Costa, A.: Exact learning of juntas from membership queries. Lect. Notes Artif. Intell. **9925**, 115–129 (2016)
9. Chandrasekaran, J., Ghandehari, L.S., Lei, Y., Kacker, R., Kuhn, D.R.: Evaluating the effectiveness of BEN in localizing different types of software fault. In: 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 26–34 (2016)
10. Chen, S.S., Donoho, D.L., Saunders, M.A.: Atomic decomposition by basis pursuit. SIAM J. Sci. Comput. **20**(1), 33–61 (1998)
11. Chen, T.Y., Kuo, F.C., Merkel, R.G., Tse, T.: Adaptive random testing: the art of test case diversity. J. Syst. Softw. **83**(1), 60–66 (2010)
12. Chen, T.Y., Leung, H., Mak, I.K.: Adaptive random testing. In: 9th Asian Computing Science Conference Advances in Computer Science—ASIAN 2004, Higher-Level Decision Making, pp. 320–329 (2004)
13. Cheng, C.S., Tang, B.: Upper bounds on the number of columns in supersaturated designs. Biometrika **88**(4), 1169–1174 (2001)
14. Cohen, D.M., Dalal, S.R., Fredman, M.L., Patton, G.C.: The AETG system: an approach to testing based on combinatorial design. IEEE Trans. Softw. Eng. **23**, 437–44 (1997)
15. Cohen, M.B., Dwyer, M.B., Shi, J.: Constructing interaction test suites for highly-configurable systems in the presence of constraints: a greedy approach. IEEE Trans. Softw. Eng. **34**, 633–650 (2008)
16. Colbourn, C.J., Fan, B.: Locating one pairwise interaction: three recursive constructions. J. Algebra Comb. Discrete Struct. Appl. **3**, 125–134 (2016)
17. Colbourn, C.J., Fan, B., Horsley, D.: Disjoint spread systems and fault location. SIAM J. Discrete Math. **30**, 2011–2016 (2016)
18. Colbourn, C.J., Lanus, E., Sarkar, K.: Asymptotic and constructive methods for covering perfect hash families and covering arrays. Designs Codes Cryptogr. **86**, 907–937 (2018)
19. Colbourn, C.J., McClary, D.W.: Locating and detecting arrays for interaction faults. J. Comb. Optim. **15**, 17–48 (2008)
20. Colbourn, C.J., Syrotiuk, V.R.: Coverage, location, detection, and measurement. In: 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 19–25. IEEE Press (2016)
21. Compton, R., Mehari, M.T., Colbourn, C.J., De Poorter, E., Syrotiuk, V.R.: Screening interacting factors in a wireless network testbed using locating arrays. In: IEEE INFOCOM International Workshop on Computer and Networking Experimental Research Using Testbeds (CNERT) (2016)
22. Dalal, S.R., Mallows, C.L.: Factor-covering designs for testing software. Technometrics **40**, 234–243 (1998)
23. Damaschke, P.: Adaptive versus nonadaptive attribute-efficient learning. Mach. Learn. **41**, 197–215 (2000)
24. Donoho, D.L., Huo, X.: Uncertainty principles and ideal atomic decomposition. IEEE Trans. Inf. Theory **47**, 2845–2862 (2001)
25. Du, D.Z., Hwang, F.K.: Combinatorial Group Testing and Its Applications, 2nd edn. World Scientific Publishing Co., Inc., River Edge (2000)
26. Erdős, P., Lovász, L.: Problems and results on 3-chromatic hypergraphs and some related questions. In: Infinite and finite sets (Colloq., Keszthely, 1973 Vol. II, pp. 609–627. Colloq. Math. Soc. János Bolyai, Vol. 10. North-Holland, Amsterdam (1975)
27. Francetić, N., Stevens, B.: Asymptotic size of covering arrays: an application of entropy compression. J. Comb. Des **25**, 243–257 (2017)
28. Frankl, P.G., Hamlet, R.G., Littlewood, B., Strigini, L.: Evaluating testing methods by delivered reliability. IEEE Trans. Softw. Eng. **24**(8), 586–601 (1998)
29. Ghandehari, L.S., Chandrasekaran, J., Lei, Y., Kacker, R., Kuhn, D.R.: BEN: a combinatorial testing-based fault localization tool. In: 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 1–4 (2015)
30. Ghosh, S., Shirakura, T., Srivastava, J.N.: Model identification using search linear models and search designs. In: Entropy, search, complexity, Bolyai Soc. Math. Stud., vol. 16, pp. 85–112. Springer, Berlin (2007)
31. Godbole, A.P., Skipper, D.E., Sunley, R.A.: t-covering arrays: upper bounds and Poisson approximations. Comb. Probab. Comput. **5**, 105–118 (1996)
32. Grindal, M., Offutt, J., Andler, S.F.: Combination testing strategies—a survey. Softw. Test. Verif. Reliab. **5**, 167–199 (2005)

33. Hamlet, D., Taylor, R.: Partition testing does not inspire confidence (program testing). IEEE Trans. Softw. Eng. **16**(12), 1402–1411 (1990)
34. Hartman, A.: Software and hardware testing using combinatorial covering suites. In: Golumbic, M.C., Hartman, I.B.A. (eds.) Interdisciplinary Applications of Graph Theory, Combinatorics, and Algorithms, pp. 237–266. Springer, Norwell (2005)
35. Hartman, A., Raskin, L.: Problems and algorithms for covering arrays. Discrete Math. **284**, 149–156 (2004)
36. Jin, H., Tsuchiya, T.: Constrained locating arrays for combinatorial interaction testing. CoRR arXiv:1801.06041 (2018)
37. Johnson, D.S.: Approximation algorithms for combinatorial problems. J. Comput. Syst. Sci. **9**, 256–278 (1974)
38. Jones, B., Majumdar, D.: Optimal supersaturated designs. J. Am. Stat. Assoc. **109**(508), 1592–1600 (2014)
39. Konishi, T., Kojima, H., Nakagawa, H., Tsuchiya, T.: Finding minimum locating arrays using a SAT solver. In: 2017 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICST Workshops 2017, Tokyo, Japan, March 13–17, 2017, pp. 276–277 (2017)
40. Kuhn, D.R., Bryce, R.C., Duan, F., Ghandehari, L.S.G., Lei, Y., Kacker, R.N.: Combinatorial testing: theory and practice. Adv. Comput. **99**, 1–66 (2015)
41. Kuhn, D.R., Kacker, R., Lei, Y.: Introduction to Combinatorial Testing. CRC Press, Boca Raton (2013)
42. Kuhn, D.R., Kacker, R., Lei, Y.: Combinatorial coverage as an aspect of test quality. Crosstalk pp. 19–23 (2015)
43. Kuhn, D.R., Mendoza, I.D., Kacker, R., Lei, Y.: Combinatorial coverage measurement concepts and applications. In: Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013 Workshops Proceedings, Luxembourg, Luxembourg, March 18–22, 2013, pp. 352–361 (2013)
44. Kuhn, D.R., Reilly, M.: An investigation of the applicability of design of experiments to software testing. In: Proceedings of 27th Annual NASA Goddard/IEEE Software Engineering Workshop, pp. 91–95. IEEE, Los Alamitos, CA (2002)
45. Kuhn, D.R., Wallace, D.R., Gallo, A.M.: Software fault interactions and implications for software testing. IEEE Trans. Softw. Eng. **30**, 418–421 (2004)
46. Li, P.C., Meagher, K.: Sperner partition systems. J. Comb. Des. **21**(7), 267–279 (2013)
47. Lovász, L.: On the ratio of optimal integral and fractional covers. Discrete Math. **13**(4), 383–390 (1975)
48. Martínez, C., Moura, L., Panario, D., Stevens, B.: Locating errors using ELAs, covering arrays, and adaptive testing algorithms. SIAM J. Discrete Math. **23**, 1776–1799 (2009-2010)
49. Meagher, K., Moura, L., Stevens, B.: A Sperner-type theorem for set-partition systems. Electron. J. Combin. **12**, Note 20 (electronic) (2005)
50. Montgomery, D.C.: Design and Analysis of Experiments, 8th edn. Wiley, New York (2012)
51. Moser, R.A., Tardos, G.: A constructive proof of the general Lovász local lemma. J. ACM **57**(2), Art. 11, 15 (2010)
52. Nagamoto, T., Kojima, H., Nakagawa, H., Tsuchiya, T.: Locating a faulty interaction in pair-wise testing. In: 20th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2014, Singapore, November 18–21, 2014, pp. 155–156 (2014)
53. Nayeri, P., Colbourn, C.J., Konjevod, G.: Randomized postoptimization of covering arrays. Eur. J. Comb. **34**, 91–103 (2013)
54. Nie, C., Leung, H.: A survey of combinatorial testing. ACM Comput. Surv. **43**(2), #11 (2011)
55. Nie, C., Wu, H., Niu, X., Kuo, F., Leung, H.K.N., Colbourn, C.J.: Combinatorial testing, random testing, and adaptive random testing for detecting interaction triggered failures. Inf. Softw. Technol. **62**, 198–213 (2015)
56. Petke, J.: Constraints: The future of combinatorial interaction testing. In: 2015 IEEE/ACM 8th International Workshop on Search-Based Software Testing (SBST), pp. 17–18 (2015)
57. Petke, J., Cohen, M.B., Harman, M., Yoo, S.: Practical combinatorial interaction testing: empirical findings on efficiency and early fault detection. IEEE Trans. Softw. Eng. **41**(9), 901–924 (2015)
58. Sarkar, K., Colbourn, C.J.: Upper bounds on the size of covering arrays. SIAM J. Discrete Math. **31**, 1277–1293 (2017)
59. Schroeder, P.J., Bolaki, P., Gopu, V.: Comparing the fault detection effectiveness of n-way and random test suites. In: Proceedings of International Symposium on Empirical Software Engineering (ISESE04), pp. 49–59 (2004)
60. Seidel, S.A., Mehari, M.T., Colbourn, C.J., De Poorter, E., Moerman, I., Syrotiuk, V.R.: Analysis of large-scale experimental data from wireless networks. In: IEEE INFOCOM International Workshop on Computer and Networking Experimental Research Using Testbeds (CNERT) (2018)
61. Seidel, S.A., Sarkar, K., Colbourn, C.J., Syrotiuk, V.R.: Separating interaction effects using locating and detecting arrays. In: International Workshop on Combinatorial Algorithms (2018)
62. Shi, C., Tang, Y., Yin, J.: Optimal locating arrays for at most two faults. Sci. China Math. **55**(1), 197–206 (2012)
63. Stein, S.K.: Two combinatorial covering theorems. J. Comb. Theory Ser. A **16**, 391–397 (1974)
64. Tai, K.C., Yu, L.: A test generation strategy for pairwise testing. IEEE Trans. Softw. Eng. **28**, 109–111 (2002)
65. Tang, Y., Colbourn, C.J., Yin, J.: Optimality and constructions of locating arrays. J. Stat. Theory Pract. **6**(1), 20–29 (2012)
66. Torres-Jimenez, J., Rodriguez-Tello, E.: New upper bounds for binary covering arrays using simulated annealing. Inf. Sci. **185**(1), 137–152 (2012)
67. Yilmaz, C., Cohen, M.B., Porter, A.: Covering arrays for efficient fault characterization in complex configuration spaces. IEEE Trans. Softw. Eng. **31**, 20–34 (2006)
68. Yilmaz, C., Dumlu, E., Cohen, M.B., Porter, A.: Reducing masking effects in combinatorial interaction testing: A feedback driven adaptive approach. IEEE Trans. Softw. Eng. **40**(1), 43–66 (2014)

69. Zhang, Y.: On theory of compressive sensing via ℓ_1-minimization: Simple derivations and extensions. Tech. Rep. CAAM TR08-11, Rice University (2008)
70. Zhou, W., Zhang, D.: Sole error locating array and approximate error locating array. In: 2012 IEEE International Conference on Computer Science and Automation Engineering, pp. 480–483 (2012)