Locating Arrays: Construction, Analysis, and Robustness

by

Stephen Seidel

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2018 by the
Graduate Supervisory Committee:

Violet R. Syrotiuk, Chair
Charles J. Colbourn
Douglas C. Montgomery

ARIZONA STATE UNIVERSITY

December 2018

ABSTRACT

Modern computer systems are complex engineered systems involving a large collection of individual parts, each with many parameters, or factors, affecting system performance. One way to understand these complex systems and their performance is through experimentation. However, most modern computer systems involve such a large number of factors that thorough experimentation on all of them is impossible. An initial screening step is thus necessary to determine which factors are relevant to the system's performance and which factors can be eliminated from experimentation.

Factors may impact system performance in different ways. A factor at a specific level may significantly affect performance as a main effect, or in combination with other main effects as an interaction. For screening, it is necessary both to identify the presence of these effects and to locate the factors responsible for them. A locating array is a relatively new experimental design that causes every main effect and interaction to occur and distinguishes all sets of d main effects and interactions from each other in the tests where they occur. This design is therefore helpful in screening complex systems.

The process of screening using locating arrays involves multiple steps. First, a locating array is constructed for all possibly significant factors. Next, the system is executed for all tests indicated by the locating array and a response is observed. Finally, the response is analyzed to identify the significant system factors for future experimentation. However, simply constructing a reasonably sized locating array for a large system is no easy task and analyzing the response of the tests presents additional difficulties due to the large number of possible predictors and the inherent imbalance in the experimental design itself. Further complications can arise from noise in the system or errors in testing.

This thesis has three contributions. First, it provides an algorithm to construct locating arrays using the Lovász Local Lemma with Moser-Tardos resampling. Second, it gives an algorithm to analyze the system response efficiently. Finally, it studies the robustness of the analysis to the heavy-hitters assumption underlying the approach as well as to varying amounts of system noise.

DEDICATION

I dedicate this thesis to my mom for being my only elementary school teacher, to my dad for being my only high school teacher, and to both for instilling all of my values and, by their examples, teaching me how to act in every aspect of my life.

ACKNOWLEDGMENTS

First and foremost, I would like to acknowledge and thank my supervisor and committee chair, Dr. Violet Syrotiuk, for constantly supporting me. She reviewed every chapter of this thesis multiple times, and provided direction throughout the entire process. Without her, this thesis would not have been completed and I would never have even applied to graduate school. I am greatly indebted to her for all of her help and support. Special thanks also goes to my co-supervisor and committee member, Dr. Charles Colbourn, for his valuable insight in combinatorics.

I would like to thank Dr. Douglas Montgomery for also serving on my committee, and providing his insight and expertise in statistics.

My dad, Dr. Mark Seidel, has been my mentor throughout graduate school and took time to proofread this thesis and provide feedback as well.

Thank you to my best friend for keeping me sane by talking to me daily on the phone, and convincing me to continue when I had every intention of dropping out.

This work is supported in part by the U.S. National Science Foundation under Grant No. 1421058.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

Large complex engineered systems involve many separate components interacting to ensure the proper functionality of the whole system. Examples of these complex engineered systems are computer networks such as the internet. Computer networks include separate components in many layers spanning from the physical to the application layer that all affect the proper functionality of the network. There are parameters (*factors*) at every layer that can be adjusted to different values (*levels*) to affect the output (*response*) of the system. Levels can therefore be assigned to the factors of the system, and a response can be observed.

Suppose we are interested in maximizing throughput for a particular network. Many network factors at multiple layers can affect the response (the throughput of the network). For example, the type of cable at the physical layer, the protocol at the transport layer, and the number of sockets at the application layer all might affect the network throughput. The effects of these factors may be due to the assignment of a particular level to one factor (*main effect*), or it may be due to a combination of multiple assignments (*interaction*). For example, the assignment of an optical fiber as the type of cable may significantly affect throughput as a main effect, while the combination of assigning TCP as the transport protocol and at the same time using a high number of sockets may significantly affect throughput as an interaction. We refer to a factor or interaction as significant when its effects are noticeable over any noise in the system.

Experimentation is crucial to understanding the behavior of complex engineered systems, because analytical approaches are no longer feasible when the system is large. But before experiments can be performed, we must determine the factors on which to perform the experimentation. This initial step of choosing the significant factors to include in experimentation, and eliminating the insignificant factors, is known as *screening*. In this work, we explore techniques for screening a large number of system factors, using initial experimentation with locating arrays, to determine the significant factors and interactions. Once screening has been performed, further experimentation should be done on the significant factors to determine *how* they affect the system. Our interest, however, is purely on the initial screening step that finds significant factors.

Similarly to [1], [2], we define k system factors $F_1, F_2, \ldots, F_k$. Each factor F_j has a set $L_j = \{v_{j1}, \ldots, v_{j\ell_j}\}$ of ℓ_j possible levels (values). A *test* is an assignment of a level from L_j to F_j for each factor. For the purpose of conciseness, this work generally omits the factor index when referring to factor levels in assignments, i.e., we use v_i instead of v_{ji}. A *t-way interaction* in a test is any collection of t assignments, and is said to have *strength* t. An assignment of v_{ji} to F_j is written as $F_j = v_i$, and a t-way interaction is written as the t assignments separated by $t-1$ ampersands. For example, $F_a = v_x$ & $F_b = v_y$ is a 2-way interaction, and is said to have strength 2. A main effect is a single assignment that is simply a 1-way interaction. Each test therefore includes (*covers*) $\binom{k}{t}$ t-way interactions. An *experimental design* of size n is a collection of n tests. When running the system, a test results in one or more output response measurements. An *experiment* consists of running the system for every test in the experimental design. Usually, an experiment is represented as an $n \times k$ array, A, with tests corresponding to rows and factors corresponding to columns. An element in the i-th row and j-th column contains a level from L_j assigned to F_j in the i-th test.

Following [1], for a t-way interaction T, denote the set of rows (tests) in A where T is covered as $\rho(A, T)$, and for a set $\mathcal{T}$ of interactions, define $\rho(A, \mathcal{T}) = \bigcup_{T \in \mathcal{T}} \rho(A, T)$.

Experimental designs used for screening fall into many different categories. Full-factorial designs include all possible combinations of levels of each factor [3] and cover all t-way interactions where $1 \leq t \leq k$. The size of a full-factorial design is exponential in the number of factors. Fractional-factorial designs contain a fixed fraction of the tests in a full-factorial design and therefore also grow exponentially in the number of factors. In general, complex engineered systems involve a large number of factors, making full-factorial and fractional-factorial experimental designs impractical for screening.

Saturated designs include a number of tests equal to one more than the number of factors [3], i.e., $n = k + 1$. Saturated designs are helpful when only a small fraction of the system factors are expected to contribute to the response. Supersaturated designs employ even fewer tests than saturated designs [3], i.e., $n < k + 1$. While they are not widely used, supersaturated designs are potentially useful when very few factors are expected to contribute to the response [3]. Both saturated and supersaturated designs conveniently involve running a small number of tests, but they only attempt to estimate main effects. It may be impossible to estimate the effects of all interactions with these designs because a significant interaction may be missing completely from the design, or it may be indistinguishable from (*confounded* with) another interaction. These designs often require advanced analysis techniques as well.

Covering arrays of strength t are experimental designs that cover every t-way interaction in at least one test [1]. Covering arrays can reveal the presence of an interaction that has a significant impact on the system response, but they do not necessarily identify the specific significant interaction [1]. Suppose a system has $k = 3$

factors, F_1, F_2, F_3 and the number of levels are $\ell_1 = 2, \ell_2 = 3, \ell_3 = 3$. Table 1 is a covering array A of strength 2 for this system. Define $T_1 = (F_3 = v_3 \;\&\; F_1 = v_2)$ and $T_2 = (F_3 = v_3 \;\&\; F_2 = v_2)$. Then $\rho(A, T_1) = \{4\}$ and $\rho(A, T_2) = \{4\}$. Suppose that T_1 is significant but T_2 is insignificant. Thus test 4 produces a significantly different response (because of T_1) and the covering array reveals the *presence* of a significant interaction. However, it is impossible to determine, from only the tests in Table 1, whether T_1 or T_2 is significant (or if both are). For the purpose of screening, we must determine the factors that are significant, and therefore must determine exactly which interaction is significant.

Table 1. Covering Array - First Example

Covering Array A			
Test	F_1	F_2	F_3
1	v_2	v_3	v_2
2	v_2	v_1	v_1
3	v_1	v_1	v_2
4	v_2	v_2	v_3
5	v_2	v_3	v_1
6	v_1	v_3	v_3
7	v_1	v_2	v_1
8	v_2	v_2	v_2
9	v_1	v_1	v_3

This work focuses on using a new experimental design for screening - a *locating array* (LA) [1]. As discussed by Colbourn and McClary [1], an experimental design is (d, t)-*locating* if $\rho(A, \mathcal{T}_1) = \rho(A, \mathcal{T}_2) \Leftrightarrow \mathcal{T}_1 = \mathcal{T}_2$ whenever $\mathcal{T}_1, \mathcal{T}_2$ are any sets of t-way interactions where $|\mathcal{T}_1| = d$, and $|\mathcal{T}_2| = d$. When $|\mathcal{T}_1| \leq d$ and $|\mathcal{T}_2| \leq d$ and $\mathcal{T}_1, \mathcal{T}_2$ are any sets of interactions with strength at most t, the array is $(\overline{d}, \overline{t})$-*locating*. Thus while the covering array in Table 1 is unable to distinguish between T_1 and T_2 because $\rho(A, T_1)$ and $\rho(A, T_2)$ are equal, both a (d, t) and $(\overline{d}, \overline{t})$-locating array guarantee that

$\rho(A, \mathcal{T}_1)$ and $\rho(A, \mathcal{T}_2)$ must be different when $\mathcal{T}_1$ and $\mathcal{T}_2$ are different sets. For the purpose of screening, a locating array provides a set of tests to locate any set $\mathcal{T}$ of interactions that significantly affect the response. Locating arrays are covering arrays with additional desirable properties, and are especially useful because they grow logarithmically in the number of factors when the number of levels is fixed for all factors [4].

In this work, we focus on locating arrays where $d = 1$ because we make the assumption that the most significant interaction stands out in the response (more significant than all other significant interactions) and it is not necessary to locate more than one interaction at the same time. We also assume that after the most significant interaction is located and its effect is removed from the response, then the second most significant interaction again stands out. This assumption of a "heavy-hitters" pattern in the significant interactions has been used with locating arrays [5], [6], and is discussed in more detail in Chapter 4 and tested in Chapter 5. Without this assumption, one needs to examine locating arrays with $d \geq 1$, and this investigation is reserved for future work. We also focus on interactions with strength at most 2, and ignore interactions with strength 3 or higher because the sparsity of effects principle indicates that most higher strength interactions are often negligible [3], [7]. However, much of this work pertaining to interactions of strength 2 can be extended to interactions with higher strength. For the remainder of this work, we refer to a 1-way interaction as a *main effect* and to a 2-way interaction as simply an *interaction.* We refer to both main effects and interactions as *terms* that can be included in a linear model of a response.

Table 2 is an example of a $(\overline{1}, \overline{2})$-locating array for the same system used in Table 1, and is constructed by adding 3 extra rows to the covering array A to create a locating array A'. The 3 extra rows are separated from the others by a horizontal line. All further locating arrays discussed in this work are $(\overline{1}, \overline{2})$-locating arrays.

Table 2. Locating Array - First Example

Locating Array A'			
Test	F_1	F_2	F_3
1	v_2	v_3	v_2
2	v_2	v_1	v_1
3	v_1	v_1	v_2
4	v_2	v_2	v_3
5	v_2	v_3	v_1
6	v_1	v_3	v_3
7	v_1	v_2	v_1
8	v_2	v_2	v_2
9	v_1	v_1	v_3
10	v_1	v_3	v_1
11	v_2	v_1	v_2
12	v_1	v_2	v_3

Recall $T_1 = (F_3 = v_3 \ \& \ F_1 = v_2)$ and $T_2 = (F_3 = v_3 \ \& \ F_2 = v_2)$. Now $\rho(A', T_1) = \{4\}$ and $\rho(A', T_2) = \{4,12\}$. The sets of rows where the two interactions are covered is now different because the three additional rows have created a locating array. Thus the two interactions are now distinguishable from each other and it is possible to determine which is significant.

Screening using locating arrays involves the following steps. First, a locating array is constructed for the possible significant factors. Second, the system is executed for every test indicated by the locating array and a response is measured. Third, the response is analyzed to determine which factors are significant and which are not. The final screening result is the set of significant factors.

This thesis contributes to the screening process in three areas: construction of locating arrays, analysis of the response, and robustness of the analysis. Two construction algorithms for locating arrays are discussed in Chapter 3. Tables containing locating array sizes for different numbers of factors and levels are also provided in Chapter 3. An efficient analysis algorithm that employs linear models to determine the significant factors in the system is discussed in Chapter 4 and its results are validated with previous analysis approaches. Because response measurements in real systems include noise, the effects of noise in the screening process and coping strategies are investigated in Chapter 5. Robustness to violation of the "heavy-hitters" assumption is also tested in Chapter 5. Finally, directions for future work are discussed in Chapter 6. Some of the work presented in Chapters 3 and 4 has been published in [2] and [8].

Chapter 2

RELATED WORK

This chapter investigates other work related to the construction of locating arrays and analysis of their response. Randomized approaches for construction of covering and locating arrays are introduced in Section 2.1, while relevant approaches for response analysis, including two approaches designed exclusively for locating arrays, are discussed in Section 2.2.

2.1 Construction Work

General construction strategies for locating arrays are found in [4]. These strategies include using the Stein-Lovász-Johnson framework and using the Lovász Local Lemma with Moser-Tardos resampling. In [4], the requirements for a locating array are presented as events that can be either *good* or *bad*. Good events are requirements that are satisfied and bad events are requirements that are violated. A collection of tests is a locating array when no bad events occur. The events are classified and further divided into patterns, and then the probability of a pattern being good is calculated for each pattern [4]. One can then determine the expected number of bad events when n tests are randomly chosen, and a locating array with no bad events is guaranteed to exist when this expectation is less than one [4]. Graphs in [4] give the necessary locating array sizes for the expectation of bad events to be less than one for different numbers of columns in the array.

Following [2], the Stein-Lovász-Johnson framework [9]–[11] indicates that a test (row) be added that reduces the number of bad events by at least the reduction that would be expected from a test chosen randomly from all possible tests. A locating array can thus be constructed by selecting one row at a time and ensuring at each selection that the number of bad events in the array does not exceed the expected number of bad events for a random array of the same size [4]. Graphs in [4] provide bounds on locating array sizes for the row-at-a-time approach for different numbers of columns in the array. These graphs also show that locating array sizes grow logarithmically in the number of factors when the number of levels is fixed for all factors.

The Lovász Local Lemma [12] is used in [13] for the construction of covering arrays with Moser-Tardos resampling [14] and can be applied to the construction of locating arrays as well [4]. The construction process is also modeled as avoiding *bad* events (situations where the array is not a valid covering or locating array) [13] and these events depend on the columns of the array. Moser-Tardos resampling indicates that when the sufficient condition for the Lovász Local Lemma is satisfied, then a solution can be found (where no bad events occur) with a randomized polynomial time algorithm [14].

Following [13], suppose E is a bad event to be avoided, then denote $\mathrm{vbl}(E)$ as the minimum subset of array columns on which E depends. For construction, the Moser-Tardos randomized polynomial time algorithm searches for any bad event E, and then randomly resamples all columns of $\mathrm{vbl}(E)$. This process of searching and resampling is repeated until no more bad events exist and the array is a valid covering or locating array. The sufficient condition for the lemma, however, requires that the event probabilities be sufficiently small. To reduce these probabilities, tests (rows) must be added to the array.

2.2 Analysis Work

After construction is complete, the tests indicated by the locating array must be run, and analysis must be performed on the results to determine the significant factors. Analysis therefore uses the response of the tests and the corresponding locating array to determine factor significance. The result of the analysis step is the set of significant factors.

Locating arrays, however, provide unique challenges in analysis. First, the number of potentially significant main effects and interactions is large. For example, a $(\overline{1}, \overline{2})$-locating array with $k = 100$ factors and $\ell = 5$ levels for every factor, distinguishes approximately 125,000 main effects and interactions. Any analysis strategy for locating arrays would thus need to analyze all of these possible terms. Second, locating arrays may exhibit high imbalance. All main effects and interactions are covered in a locating array, but some terms are covered much more than others. In the locating array in Table 2, some interactions appear three times (e.g., $F_3 = v_3$ & $F_1 = v_1$) while others appear just once (e.g., $F_3 = v_3$ & $F_2 = v_3$). Analysis strategies for locating arrays should not be biased towards terms that are covered more often than others.

Many approaches exist to determine which terms are significant when the number of possible terms, or predictors, is large. There are model-based approaches, where models are built to explain the response, and model-free approaches, where the significant terms are found without building an explicit model [15]. In model-based approaches, there are different types of models that can be built. Among others, there are linear models, along with models that use parameterization to produce complex models [15].

Colbourn et al. [6] introduce the use of a compressive sensing matrix with a locating array to generate prospective terms for linear modeling. The compressive sensing matrix contains a separate column for each term. Table 3 is the compressive sensing matrix for the locating array in Table 2. For each column, +1 is placed in the rows where the term is covered (the rows in $\rho(A', T)$, where T is the main effect or interaction), and -1 is placed in the remaining rows. To conserve space, every +1 is written as + and every -1 is written as $-$ in the table. We also include a column with +1 in every row as the *INTERCEPT* to account for any effects that exist in the system regardless of the factor assignments. Interestingly, in any compressive sensing matrix corresponding to a locating array, every column must be unique because the set of rows covered by each term must be unique. This work exclusively investigates linear modeling of the response using the columns of a compressive sensing matrix. While transformations can be applied to the predictors (columns of the compressive sensing matrix) to produce complex models [15], this work focuses exclusively on screening using linear models with an optional transformation applied to the response.

Sure independence screening (SIS) is a screening approach for a high number of terms using linear models [15]. In SIS, a response vector (response of the locating array tests) is modeled with terms chosen from a pool of predictor vectors (columns of the compressive sensing matrix). Predictor vectors are individually ranked using the dot product of the predictor vector and the response vector divided by the length of the vectors, known as the Pearson correlation [15]. This ranking of predictor vectors indicates their correlation with the response vector. Finally, the predictors that are highly correlated with the response are chosen as the result of the screening process.

Table 3. Compressive Sensing Matrix - First Example

Compressive Sensing Matrix																													
INTERCEPT	$F_1 = v_1$	$F_1 = v_2$	$F_2 = v_1$	$F_2 = v_2$	$F_2 = v_3$	$F_3 = v_1$	$F_3 = v_2$	$F_3 = v_3$	$F_2 = v_1$ & $F_1 = v_1$	$F_2 = v_1$ & $F_1 = v_2$	$F_2 = v_2$ & $F_1 = v_1$	$F_2 = v_2$ & $F_1 = v_2$	$F_2 = v_3$ & $F_1 = v_1$	$F_2 = v_3$ & $F_1 = v_2$	$F_3 = v_1$ & $F_1 = v_1$	$F_3 = v_1$ & $F_1 = v_2$	$F_3 = v_1$ & $F_2 = v_1$	$F_3 = v_1$ & $F_2 = v_2$	$F_3 = v_1$ & $F_2 = v_3$	$F_3 = v_2$ & $F_1 = v_1$	$F_3 = v_2$ & $F_1 = v_2$	$F_3 = v_2$ & $F_2 = v_1$	$F_3 = v_2$ & $F_2 = v_2$	$F_3 = v_2$ & $F_2 = v_3$	$F_3 = v_3$ & $F_1 = v_1$	$F_3 = v_3$ & $F_1 = v_2$	$F_3 = v_3$ & $F_2 = v_1$	$F_3 = v_3$ & $F_2 = v_2$	$F_3 = v_3$ & $F_2 = v_3$
+	-	+	-	-	+	-	+	-	-	-	-	-	-	+	-	-	-	-	-	-	+	-	-	+	-	-	-	-	-
+	-	+	+	-	-	+	-	-	-	+	-	-	-	-	-	+	+	-	-	-	-	-	-	-	-	-	-	-	-
+	+	-	+	-	-	-	+	-	+	-	-	-	-	-	-	-	-	-	-	+	-	+	-	-	-	-	-	-	-
+	-	+	-	+	-	-	-	+	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	+	-
+	-	+	-	-	+	+	-	-	-	-	-	-	-	+	-	+	-	-	+	-	-	-	-	-	-	-	-	-	-
+	+	-	-	-	+	-	-	+	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	+
+	+	-	-	+	-	+	-	-	-	-	+	-	-	-	+	-	-	+	-	-	-	-	-	-	-	-	-	-	-
+	-	+	-	+	-	-	+	-	-	-	-	+	-	-	-	-	-	-	-	-	+	-	+	-	-	-	-	-	-
+	+	-	+	-	-	-	-	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	+	-	-
+	+	-	-	-	+	+	-	-	-	-	-	-	+	-	+	-	-	-	+	-	-	-	-	-	-	-	-	-	-
+	-	+	+	-	-	-	+	-	-	+	-	-	-	-	-	-	-	-	-	-	+	+	-	-	-	-	-	-	-
+	+	-	-	+	-	-	-	+	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	+	-

In some cases, however, issues can arise in SIS. For example, if an insignificant term is highly correlated with a significant term [16], then SIS is likely to rank both terms highly and fails to eliminate the insignificant term. Iterative SIS [16] is similar to the SIS screening approach but is done in steps. In the i-th step, iterative SIS uses the SIS approach to select k_i relevant terms, and the residual vector that remains after selecting the terms replaces the response vector in the $(i + 1)$-st step [16]. The k_i variable for each step can be chosen in a variety of ways, but if it is always equal to one, then iterative SIS is a form of matching pursuit [16].

The critical aspect of SIS is how correlation is determined between the predictor vectors (columns of the compressive sensing matrix) and the response vector. Previously, we discussed the use of the Pearson correlation to determine the ranking of the predictors. However, other ranking strategies also exist in SIS, including the marginal rank correlation which can behave better than the Pearson correlation in some situations [15].

We now discuss two screening experiments using locating array designs in Section 2.2.1 [6] and Section 2.2.2 [5]. In both, the analysis of the associated data resembles a "heavy-hitters" algorithm to construct a response model. Both maintain a current model and a current residual vector. The model is initialized as an intercept term with coefficient equal to the mean of the measured response data, while the residuals are equal to the data minus the model's predictions of it.

In each iteration, the most significant term impacting the response vector is selected to add to the model. The two analysis methods differ, however, in *how* the most significant term is selected. Least squares is then used to update the model coefficients. As in iterative SIS, the residuals are then calculated and replace the response vector in the next iteration. These steps are repeated until a stopping criterion is met. The two analysis methods make use of a limit on the number of terms and a desired R^2 (coefficient of determination) threshold.

2.2.1 Orthogonal Matching Pursuit

Orthogonal matching pursuit (OMP) is a compressive sensing-based analysis approach used for term selection in [6]. In each iteration, a term is selected to add to the model based on the dot product of the normalized residuals with each candidate term's normalized column in a compressive sensing matrix. The term yielding the highest-magnitude dot product is added to the model, after which ordinary (linear) least squares is used to update the model coefficients. A dot product is used because a dot product of zero indicates a complete lack of correlation between a term and the data, a dot product of 1 indicates perfect correlation, and a dot product of -1 indicates a perfect anti-correlation. Also, the dot product is linear in each argument fitting well with using least squares to update the model coefficients. This approach is similar to iterative SIS with $k = 1$ and using the Pearson correlation. Because OMP can only add terms to a model, a tree is developed in [6] in a depth-first manner to represent terms that have been added, and backjumping provides a mechanism for exploring alternative models.

2.2.2 Statistical Approach

Along with the challenge of a large number of possible terms, locating arrays also present difficulties because of their imbalance. In [5], to cope with the imbalance in coverage, the factors are grouped according to the number of times each value is covered in the locating array. In each iteration of the heavy hitters algorithm, the first step selects the most significant main effect or two-way interaction from each group using the Wilcoxon rank sum test and the Mann-Whitney U-test [17]–[19]. Then from these candidates, the most significant main effect or two-way interaction overall is selected using the Akaike information criterion [20]. Weighted least squares is also used to update the model coefficients.

2.3 Summary

The Lovász Local Lemma with Moser-Tardos resampling has been used for covering array construction [13]. Chapter 3 focuses on using the same resampling strategy, but for locating array construction. Analysis results are also presented from several locating arrays constructed using this approach. Chapter 4 explores the analysis step by extending the work in [5], [6].

Chapter 3

CONSTRUCTION

The first step in screening using locating arrays is to construct the locating array. In this step, one chooses k factors $F_1, F_2, \ldots, F_k$, to be used in screening and determines the number of levels ℓ_j for each factor F_j, and then constructs a valid locating array. Producing any locating array is trivial. A full-factorial screening experiment is a locating array since every possible combination of level-to-factor assignments exists in the design. However, we focus on cases where many possible factors have been chosen for screening. In such cases, the size of a full-factorial experiment is extremely large. The large system introduced in Section 2.2 with $k = 100$, for example, contains over 10^{69} tests if a full-factorial experiment were to be used. Such sizes are infeasible, and this chapter discusses strategies to construct locating arrays that are more useful in practice.

3.1 Desirable Properties

Certain desirable properties exist when considering the construction of locating arrays along with their usages. First, it is desirable for locating arrays to have few rows (or tests). Because every test takes some amount of time to complete, locating arrays with billions of rows are undesirable and likely impossible to use. However, "few" is a relative attribute, and this property is therefore strongly dependent on how practical it is to run multiple tests in a particular environment. Second, it is desirable for locating arrays to produce accurate results when they are used. This second property is intuitive, but it is not completely clear what types of locating arrays produce accurate results. We hypothesize that locating arrays can be constructed to produce more accurate results through the use of a particular attribute that we refer to as *separation*. We first motivate separation with an example and then formally define it.

Suppose we construct a locating array with hundreds of rows. Next, we construct the associated compressive sensing matrix, but observe that two of its columns are quite similar and differ in only a single row. If we then attempt to use this compressive sensing matrix to produce accurate results, the two similar columns are likely (almost) indistinguishable. Furthermore, if the term corresponding to one column is truly significant while the term corresponding to the other column is insignificant, then it is likely difficult to determine which of the two is the important term (and which to discard). Finally, if an error in measurement occurs in the unique test where the two columns differ, then the locating property of the array is lost entirely. Therefore, we hypothesize that it is beneficial for every column in the compressive sensing matrix to differ from every other column in at least δ rows. In this case, we refer to the associated locating array as exhibiting separation δ.

A formal definition of separation δ is the minimum difference between any two columns of an associated compressive sensing matrix in the number of rows where the columns differ. Thus it is the number of rows in which any two terms are guaranteed to differ for a particular locating array. If a locating array, A, exhibits separation δ, then for every two terms T_1, T_2 where $T_1 \neq T_2$, we have that $|(\rho(A, T_1) \cup \rho(A, T_2)) \setminus (\rho(A, T_1) \cap \rho(A, T_2))| \geq \delta$.

For example, Table 4 shows a locating array for factors F_1, F_2 both with levels v_1, v_2, v_3, followed by three columns from the corresponding compressive sensing matrix.

In this example we focus on only these three columns for simplicity and do not show the remainder of the compressive sensing matrix. Three unique pairs of columns exist from the three compressive sensing columns shown. The first pair is columns one and two, and this pair of columns differs in four rows (5, 12, 13, and 16). The second pair is columns one and three, and this pair of columns differs in two rows (5 and 12). The third pair is columns two and three, and this pair of columns differs in two rows (13 and 16). The minimum difference between these three pairs is two rows, and the locating array in Table 4 therefore exhibits separation $\delta = 2$.

Table 5 continues the example in Table 4 by adding another row to the locating array (and compressive sensing matrix). The second and third pairs both differ in the additional test. The minimum difference between the pairs is now 3 rows and the separation of this locating array is $\delta = 3$.

Finally, we repeat the test added in Table 5 to continue the separation example in Table 6. The second and third pairs both differ again in the additional test. The minimum difference between the pairs is now 4 rows and the separation of this locating array is $\delta = 4$. This example shows how strategically adding rows to a locating array can increase its separation.

Table 4. Separation Example - $\delta = 2$

Locating Array		
Test	F_1	F_2
1	v_1	v_3
2	v_2	v_2
3	v_2	v_3
4	v_3	v_3
5	v_3	v_1
6	v_1	v_3
7	v_3	v_2
8	v_3	v_2
9	v_1	v_2
10	v_2	v_2
11	v_2	v_3
12	v_3	v_1
13	v_2	v_1
14	v_3	v_3
15	v_1	v_2
16	v_2	v_1

Partial Compressive Sensing Matrix			
Test	$F_2 = v_1$	$F_2 = v_1$ & $F_1 = v_1$	$F_2 = v_1$ & $F_1 = v_2$
1	-1	-1	-1
2	-1	-1	-1
3	-1	-1	-1
4	-1	-1	-1
5	1	-1	-1
6	-1	-1	-1
7	-1	-1	-1
8	-1	-1	-1
9	-1	-1	-1
10	-1	-1	-1
11	-1	-1	-1
12	1	-1	-1
13	1	-1	1
14	-1	-1	-1
15	-1	-1	-1
16	1	-1	1

Table 5. Separation Example Continued - $\delta = 3$

Locating Array		
Test	F_1	F_2
17	v_1	v_1

Partial Compressive Sensing Matrix			
Test	$F_2 = v_1$	$F_2 = v_1$ & $F_1 = v_1$	$F_2 = v_1$ & $F_1 = v_2$
17	1	1	-1

Table 6. Separation Example Continued - $\delta = 4$

Locating Array		
Test	F_1	F_2
18	v_1	v_1

Partial Compressive Sensing Matrix			
Test	$F_2 = v_1$	$F_2 = v_1$ & $F_1 = v_1$	$F_2 = v_1$ & $F_1 = v_2$
18	1	1	-1

Note that an array must have separation of at least one to satisfy the locating property. However, we hypothesize that when locating arrays exhibit higher separation, they produce more accurate results when used. Evidence supporting this hypothesis is presented in Section 5.3.

3.2 Initial Greedy Approach

A greedy approach can be used to quickly create a simple locating array without extra separation. This approach begins with any array of tests, A, that does not have the locating property (it can be empty), and adds rows in a greedy fashion until the locating property is satisfied. This greedy fashion of adding rows relies on a score indicating how close A is to having the locating property. One possible, and effective, way to achieve this scoring, is to count the unique pairs of identical columns in the compressive sensing matrix, M, for A. This scoring approach yields a lower score when A improves and is closer to having the locating property.

Suppose that A is an empty array. Then all columns in M are also empty and are therefore all identical. Any combination of two columns in M is a pair of identical columns, and there are then $\binom{m}{2}$ unique pairs of identical columns where m is the number of columns in M. It is not possible for the number of identical columns in M to be more than $\binom{m}{2}$ so this is the largest possible score for A. Intuitively, an empty array is as far as possible from having the locating property and therefore exhibits the maximum score.

Suppose now that A is a valid locating array. Then not a single pair of identical columns exists in M because the locating property is satisfied, and the score of A is 0, the minimum score. Therefore, the score simply counts the deficiencies in A, and once the score is 0, A has no deficiencies and construction is complete.

Algorithm 1 lists the detailed steps of this greedy approach. The algorithm begins with an empty array, A, with 0 rows and then adds rows until A is a locating array. It is important to note that adding any row to A cannot worsen its score (adding a row cannot create more unique pairs of duplicate columns in the compressive sensing matrix). First, a row of random factor assignments corresponding to the *factors* parameter is generated and added to A. The entries of this row are origininally marked as not *finalized* which means they can still be changed to improve the score of the locating array. All pairs of duplicate columns in the compressive sensing matrix are then found, and all columns included in these pairs are iterated through. For each column, the algorithm attempts to update and finalize the factor assignments affecting this column so that the column entry in the most recently added row changes, and the column is no longer part of a duplicate pair. This update is the crux of the algorithm, and only occurs if the affecting factor assignments have not yet been marked as finalized. The score of A is checked after the update and then the array is reverted to its previous state. After all columns have been checked, the algorithm keeps the column updates that produced the best score improvement in A. The most recently added row now contains factor assignments that are marked as finalized and these assignments can no longer change. The algorithm then repeats the process of checking all pairs of duplicate columns until the score cannot be further improved. Finally, another row is added and the entire process repeats until the array contains no deficiencies and is a valid locating array.

This approach promises continuous improvements to the score since every row can be used to fix at least one problem, and perhaps more if they exist. The score continues to get better as rows are added and, therefore, this approach is guaranteed to complete the creation of a locating array. The algorithm is greedy because it chooses, and keeps, a current best option at every iteration.

When Algorithm 1 first begins to execute, before many rows have been added to the array, a large number of duplicate pairs exist and the loop on line 12 can take an unreasonable amount of time. In this situation, however, any random row likely eliminates a large portion of these duplicates and improves the score significantly. It is then of little consequence if all iterations of the loop on line 12 are not performed when a large number duplicate pairs exist. Thus, adding a time bound to the loop is a reasonable approach to decrease execution time with little change to the resulting locating array.

Unfortunately, this approach would require significant modification to construct locating arrays with separation greater than one, because it only attempts to eliminate all duplicate columns. When no duplicate columns exist in the compressive sensing matrix, the locating property is satisfied, but the resulting locating array is only

Algorithm 1 Greedy_Construction($factors$)

```
Require: List of factors (including their levels)
Ensure: A valid locating array
 1: $A \leftarrow$ [empty array with 0 rows]
 2: while $A.getScore() > 0$ do
 3:   $row \leftarrow$ [random row of valid factor assignments for $factors$]
 4:   for $assignment \in row$ do
 5:     $assignment.finalized \leftarrow False$
 6:   end for
 7:   Add $row$ to $A$, and keep $row$ as reference
 8:   while $True$ do
 9:     $M \leftarrow createCSM(A)$
10:     $dPairs \leftarrow$ [all pairs of duplicate columns in $M$]
11:     $bestA \leftarrow A$
12:     for $pair \in dPairs$ do
13:       for $column \in pair.columns$ do
14:         $Acopy \leftarrow$ [copy of $A$]
15:         for $factor \in column.factors$ do
16:           $assignment \leftarrow row.assignments(factor)$
17:           if NOT $assignment.finalized$ then
18:             Update $assignment$ to change final entry of $column$ in $M$
19:             $assignment.finalized \leftarrow True$
20:           end if
21:         end for
22:         if $A.getScore() < bestA.getScore()$ then
23:           $bestA \leftarrow A$
24:         end if
25:         $A \leftarrow Acopy$
26:       end for
27:     end for
28:     if $bestA.getScore() < A.getScore()$ then
29:       $A \leftarrow bestA$
30:     else
31:       $break$
32:     end if
33:   end while
34: end while
35: return $A$
```

guaranteed to have separation of one. Our initial greedy approach needs to be changed for creating locating arrays with specific properties including higher separation.

Table 7 shows factor and level inputs in the first column, and the sizes, in terms of rows, of the successfully created locating arrays using our initial approach in the second column. In the first column, the exponent refers to the number of factors in the locating array, while the base refers to the number of levels for each factor, i.e., 3^{10} means 10 factors with 3 levels each.

Table 7. Locating Array Sizes - Generated by the Initial Greedy Approach

Type	Number of Rows, $\delta = 1$
2^{10}	13
2^{15}	16
2^{20}	18
2^{50}	23
2^{75}	27
2^{100}	29
3^{10}	30
3^{15}	33
3^{20}	37
3^{50}	47
3^{75}	54
3^{100}	60
4^{10}	47
4^{15}	54
4^{20}	60
4^{50}	81
4^{75}	95
4^{100}	105
5^{10}	71
5^{15}	83
5^{20}	90
5^{50}	127
5^{75}	149
5^{100}	165
$5^{10}2^{10}$	72

3.3 Randomized Approaches

Randomized approaches to locating array construction allow for additional constraints with minimal changes to their algorithms and implementations. In a randomized approach, parts of the array are randomly generated repeatedly until the result is a valid locating array that also satisfies any possible constraints. For example, we might generate a random array repeatedly until it is a locating array with separation $\delta = 2$ or 3. We might specify additional constraints as well, but the main randomized approach remains the same. It provides the significant benefit that one can change the constraints, and even the type of array to generate, with minimal changes. This also contrasts sharply with our initial greedy approach that was highly specific to the construction of locating arrays with separation $\delta = 1$.

The core of a randomized approach lies in what we refer to as the *checker*. The checker verifies that the array is a locating array and that it satisfies any additional constraints. Because a randomized approach may take many iterations, it is important for the checker to be efficient. Slow checkers will likely cause the randomized approach to be ineffective.

A checker for a locating array with separation $\delta = 1$ might simply sort the columns of the compressive sensing matrix, and then search for duplicate columns in a linear fashion which are adjacent because of the sort. However, this approach would not work when checking for locating arrays with separation $\delta = 2$ or higher. In these cases, a pair of columns may differ only once in the first row, and then these columns are not adjacent after a sort and are much more difficult to find. A simple solution is to check all pairs of columns in the compressive sensing matrix and then iterate through all rows of the locating array to check for differences, but this can be time consuming. The number of pairs to check is $\binom{m}{2} = \frac{m\cdot(m-1)}{2} = \frac{m^2}{2} - \frac{m}{2}$ where m is the number of columns in the compressive sensing matrix. This approach has runtime $O(m^2 \cdot n)$ where the compressive sensing matrix is $n \times m$, and it is not useful for checking large locating arrays.

We propose a more efficient recursive checker in Algorithm 2 for locating arrays with higher separation that eliminates many of the pairs to check. A prerequisite for the algorithm is that the compressive sensing matrix must be sorted and converted into a binary tree format. Table 8 displays a partial compressive sensing matrix example along with the same matrix but with sorted columns and merged cells. The columns are sorted in increasing order by row from top to bottom, and horizontally adjacent cells are merged when their corresponding column entries are identical in every row, excluding the rows below the cells to be merged.

The cells are then converted to nodes in the binary tree structure in Figure 1, where the height of the tree corresponds to the number of rows in the compressive sensing matrix. The tree structure allows the compressive sensing matrix to be traversed and analyzed efficiently in Algorithm 2. Every node in the tree represents a group of columns from the sorted matrix whose entries are identical in the first ℓ rows where ℓ is the number of levels between the root and the node. For example, *Node1* represents the first three columns of the sorted matrix, and these columns are identical in the first row because this node is one level below the root of the tree. The *Root* node encompasses all columns because all columns of the sorted matrix are identical in the first zero rows (there are no entries in the first zero rows and so all columns are identical). Paths from the *Root* node to the leaf nodes correspond to column entries in the compressive sensing matrix.

Table 8. Separation checker example - Compressive sensing matrix

Partial CS Matrix				
T0	*T1*	*T2*	*T3*	*T4*
1	1	-1	-1	-1
1	-1	1	-1	-1
1	-1	-1	1	-1
1	-1	-1	-1	1

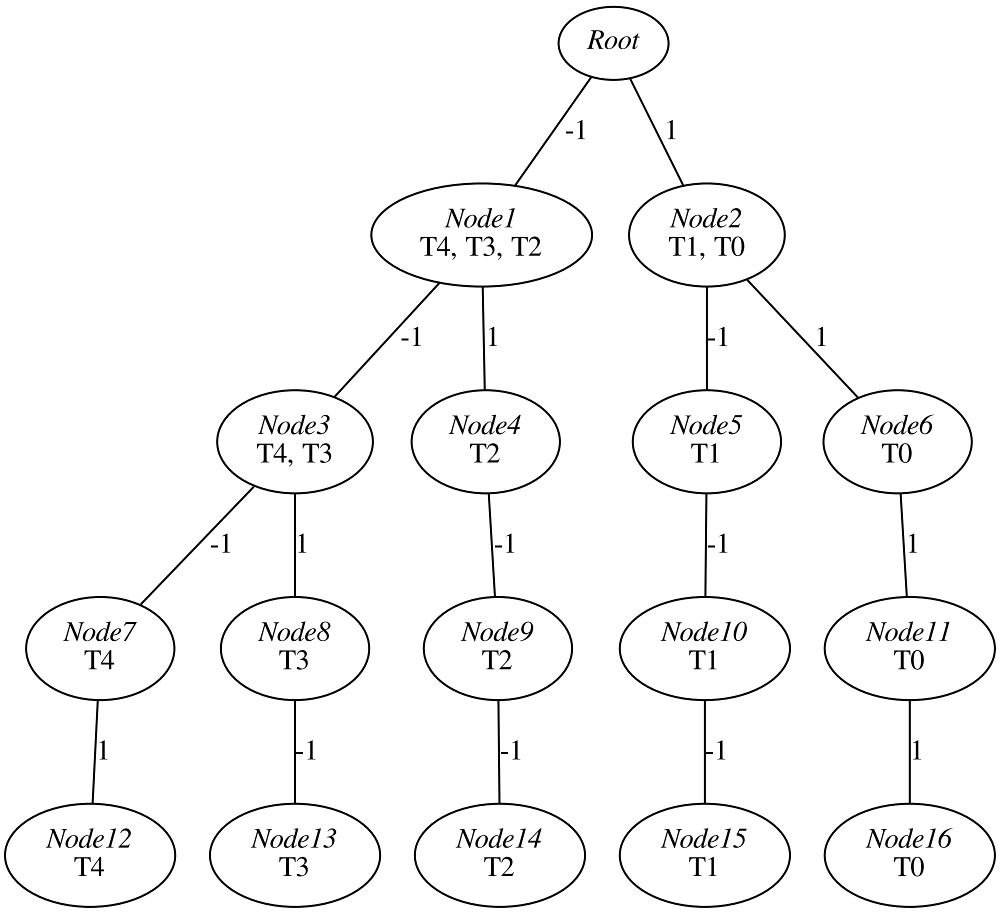
Sorted Partial CS Matrix				
T4	*T3*	*T2*	*T1*	*T0*
-1	-1	-1	1	1
-1	-1	1	-1	1
-1	1	-1	-1	1
1	-1	-1	-1	1

Sorted Partial CS Matrix with Merged Cells				
T4	*T3*	*T2*	*T1*	*T0*
-1			1	
-1		1	-1	1
-1	1	-1	-1	1
1	-1	-1	-1	1

Algorithm 2 checks for any pairs of columns that violate the separation constraint parameter, δ, where one column of the pair belongs to *nodeA* and the other belongs to *nodeB*. The two nodes passed to the algorithm are assumed to be on same level in the tree. If a violating pair exists, then the two columns must both belong to the *Root* node, and the algorithm is therefore first called as Separation_Checker($\delta, Root, Root$). The algorithm then takes the two node arguments, and searches for pairs of columns that violate the separation constraint in their child nodes.

Initially, Algorithm 2 checks four terminating conditions in the following order. First, true is returned, meaning the checker did not find any violating pairs, if $\delta = 0$,

Figure 1. Separation checker example - Binary tree corresponding to sorted compressive matrix



Algorithm 2 Separation_Checker(δ, $nodeA$, $nodeB$)

```
Require: Separation constraint, the two groups of columns to check against each other
Ensure: Whether the locating array satisfies the constraints
 1: if $\delta = 0$ then
 2:     return $True$
 3: else if $nodeA = NIL$ OR $nodeB = NIL$ then
 4:     return $True$
 5: else if $|nodeA.columns \cup nodeB.columns| = 1$ then
 6:     return $True$
 7: else if [current level] $= n$ then
 8:     return $False$
 9: end if
10: $satLL \leftarrow Separation\_Checker(\delta, nodeA.childL, nodeB.childL)$
11: $satRR \leftarrow Separation\_Checker(\delta, nodeA.childR, nodeB.childR)$
12: $satLR \leftarrow Separation\_Checker(\delta - 1, nodeA.childL, nodeB.childR)$
13: if $nodeA.columns \cup nodeB.columns = \emptyset$ then
14:     $satRL \leftarrow Separation\_Checker(\delta - 1, nodeA.childR, nodeB.childL)$
15: else
16:     $satRL \leftarrow True$
17: end if
18: return $satLL$ AND $satRR$ AND $satLR$ AND $satRL$
```

because all pairs of columns satisfy the separation requirement of being different in at least zero rows. Second, true is returned if one of the nodes is *NIL*, because no violating pair of columns can exist when one column from the pair must be part of an empty group (*NIL* node). Third, true is returned if the cardinality of the union of both groups is one, because any violating pair requires at least two columns. Fourth, false is returned, meaning the checker did find violating pairs, if the current level is equal to the total height of the tree, because this means the final level of the tree has been explored, and no more children exist to satisfy the separation requirement.

Finally, Algorithm 2 conquers the two nodes, or groups of columns, with recursive calls. For any *nodeA* and *nodeB*, four children exist after the four terminating conditions are checked:

1. $nodeA.childL$ (the left child of $nodeA$)
2. $nodeA.childR$ (the right child of $nodeA$)
3. $nodeB.childL$ (the left child of $nodeB$)
4. $nodeB.childR$ (the right child of $nodeB$)

Because the algorithm is searching for a violating pair with one column in $nodeA$ and the other in $nodeB$, this pair can exist in four ways among the children:

1. between $nodeA.childL$ and $nodeB.childL$ (both left children)
2. between $nodeA.childR$ and $nodeB.childR$ (both right children)
3. between $nodeA.childL$ and $nodeB.childR$ (left child and right child)
4. between $nodeA.childR$ and $nodeB.childL$ (right child and left child)

The first and second possibilities are between children going the same direction (both left or both right), and no extra information is obtained to separate the child groups of columns by traveling one more level down the tree. However, the third and fouth possibilities are between children going in opposite directions, and these groups are therefore separated by one more row by traveling one more level down the tree. Therefore, the third and fourth recursive calls pass the argument $\delta - 1$. When $nodeA$ is the same as $nodeB$, then the third and fourth possibilities are identical, and the final if-statement in Algorithm 2 ensures that both possibilities are not checked in this case. All four recursive calls must return true, indicating all child group combinations satisfy the separation constraint, for the entire algorithm to return true. The algorithm ensures that if a violating pair of columns exists, then it is found and false is returned, and if not, then true is returned.

Algorithm 2 works efficiently by eliminating possible violating pairs from the search. For example, suppose it is used to check for separation $\delta = 1$ in the tree in

Figure 1. There are $\frac{m^2}{2} - \frac{m}{2}$ pairs to check where m is the number of columns in the compressive sensing matrix. The first call is Separation_Checker$(1, \textit{Root}, \textit{Root})$. This call spawns three recursive calls because *nodeA* and *nodeB* are the same node:

1. Separation_Checker$(1, \textit{Node1}, \textit{Node1})$
2. Separation_Checker$(1, \textit{Node2}, \textit{Node2})$
3. Separation_Checker$(0, \textit{Node1}, \textit{Node2})$ (terminates because $\delta = 0$)

The third recursive call terminates because it is checking for $\delta = 0$ and this eliminates all the pairs with one column in *Node1* and the other in *Node2*. If we assume that *Node1* and *Node2* both have approximately $\frac{m}{2}$ columns, then $(\frac{m}{2})^2 = \frac{m^2}{4}$ pairs are eliminated from the search when the third recursive call is terminated. Therefore, more than half of the possible pairs are eliminated almost immediately, and more quickly follow as the algorithm continues.

3.3.1 Moser-Tardos Resampling

The first randomized approach is Moser-Tardos resampling and is given in Algorithm 3. Before calling the algorithm, we guess what size $(n \times k)$ might be appropriate for a locating array with the required constraints. The algorithm then generates a random $n \times k$ array, A. Following the Moser-Tardos resampling strategy in Section 2.1, it checks (runs the checker on the compressive sensing matrix M) each requirement to be a locating array with the required constraints in an arbitrary but fixed order. If no requirement is violated, then A is a solution. Otherwise, it finds the first pair of columns in M violating the required constraints and all columns of A involved in the violating pair, and randomly resamples these entire columns in A. After resampling part of A, M is updated, and then it checks each requirement again and continues until no requirements are violated.

Algorithm 3 Moser − Tardos_Construction($factors, n, k, constraints$)

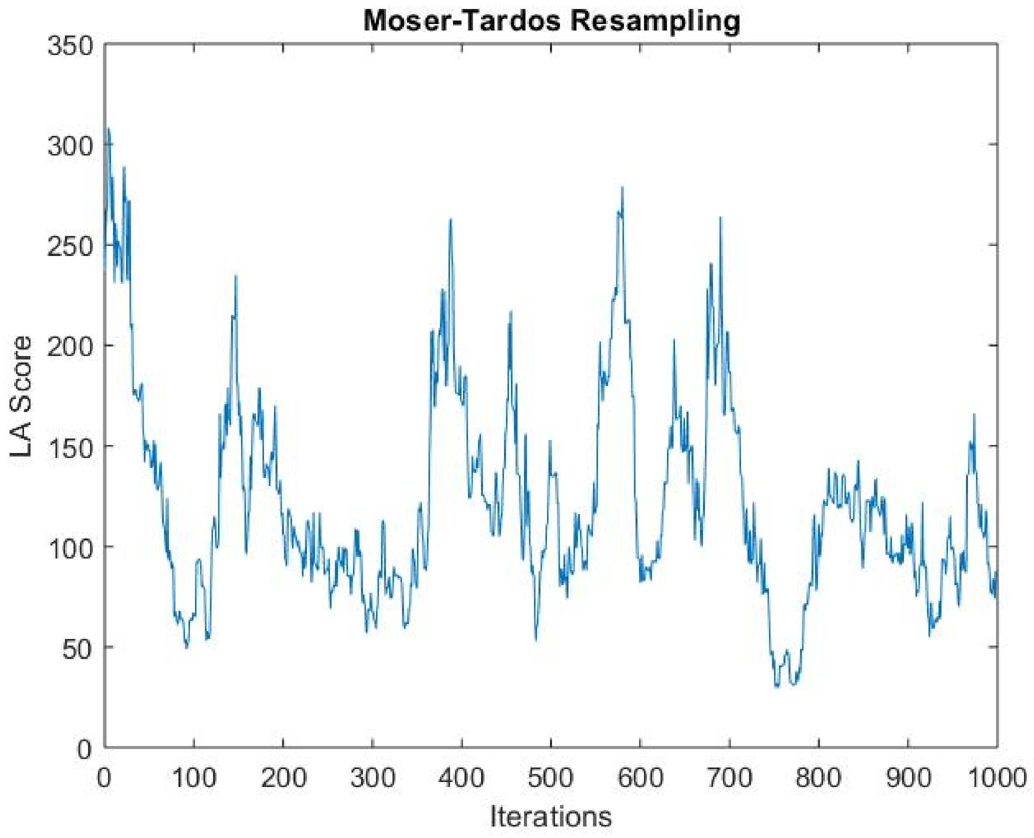
```
Require: List of factors (including their levels), rows in array, columns in array, extra
    constraints
Ensure: A valid locating array
 1: $A \leftarrow$ [random $n \times k$ array of valid factor assignments for $factors$]
 2: $M \leftarrow createCSM(A)$
 3: while $M$ violates any $constraints$ do
 4:     $pair \leftarrow$ the first pair of columns in $M$ violating $constraints$.
 5:     for $column \in pair.columns$ do
 6:         for $factor \in column.factors$ do
 7:             $A.columns(factor).resample()$
 8:         end for
 9:         $M \leftarrow createCSM(A)$
10:     end for
11: end while
12: return $A$
```

One important detail of this approach is that it is not guaranteed to complete. Unlike our initial approach, it may not make continuous progress. When columns are resampled, more requirements may be violated, and the array may never satisfy the locating property or any other constraints.

Suppose we hope to construct a locating array for 100 factors with five levels each with separation $\delta = 1$ using Moser-Tardos resampling. Our initial approach successfuly found a locating array satisfying these constraints with 165 rows.

We now implement the scoring system in Section 3.2 to monitor the progress of Algorithm 3. Figure 2 shows the array score at each iteration of the while loop using Moser-Tardos resampling for an array with 215 rows. Through 1000 iterations, however, it does not successfully construct a valid locating array although it comes close. The score starts at 237, rises to 308, and dips as low as 30 at one point before rising again.

Figure 2. LA score after each iteration using Moser-Tardos resampling.

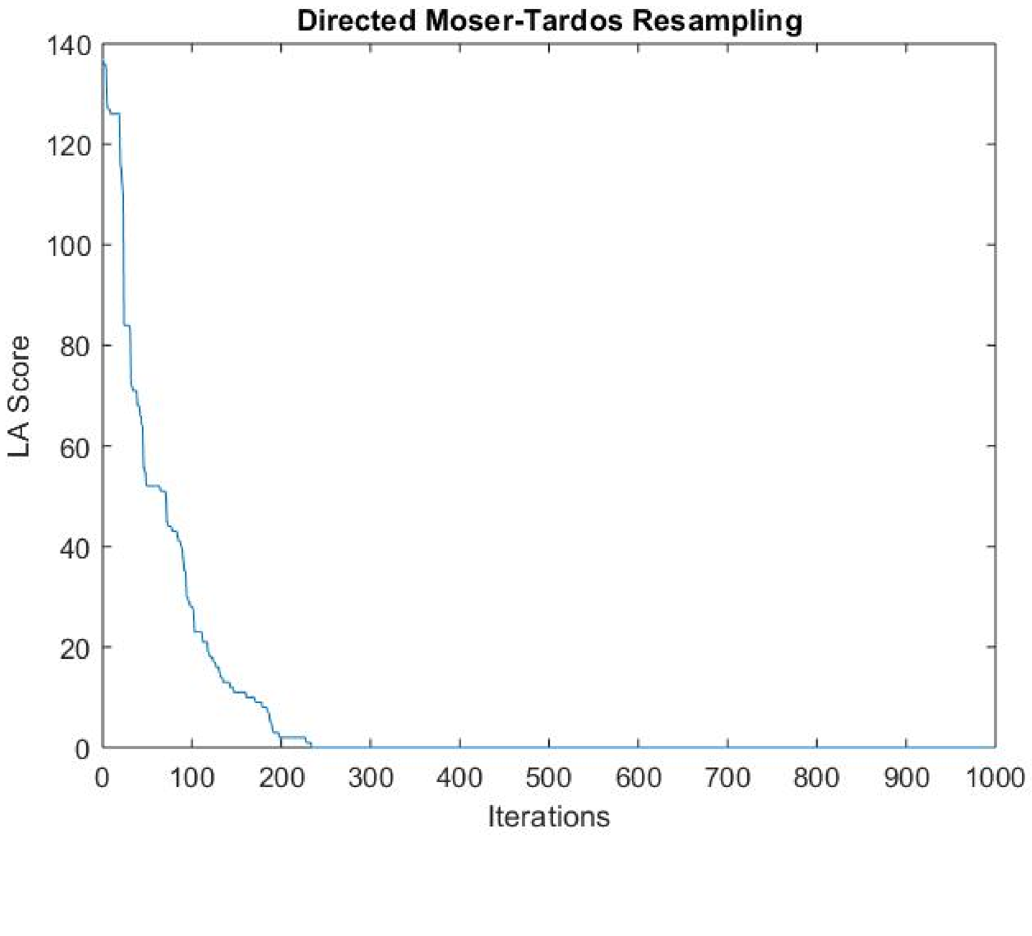


Therefore, in order to construct a valid locating array in this case, one would need to either increase the number of iterations or increase the number of rows in the array. However, we also consider making a minor change to the approach itself to direct the array towards a valid solution instead of resampling purely at random. The updated approach is discussed next.

3.3.2 Directed Moser-Tardos Resampling

Figure 2 shows the array score changing at random to all appearances. In other words, some changes improve the score, while others worsen the score. We therefore introduce a new directed construction approach that differs only slightly from Moser-Tardos resampling in that it does not accept any changes that worsen the score of the array. After every change, the directed approach checks the array score, and if the score worsens, the change is rolled back. It then resamples randomly repeatedly until it finds a change that does not worsen the array score. Figure 3 shows the array score through 1000 iterations for the same construction scenario as Figure 2. The score, in this case, decreases sharply and a locating array is successfully constructed after 234 iterations. Similar to pure Moser-Tardos resampling, however, the directed resampling approach is not guaranteed to finish.

Figure 3. LA score after each iteration using directed Moser-Tardos resampling.



Directed Moser-Tardos resampling can also be easily modified to construct locating arrays with additional constraints such as higher separation. Only the scoring system must be modified to indicate how close the array is to a valid locating array with a separation value δ.

We modify the scoring approach described in Section 3.2 to score an array, A, with a constraint, separation δ, as well. The modified scoring approach first finds the unique pairs in the compressive sensing matrix, M, for A, where the separation requirement is violated (the violating pairs). Then, for every violating pair, the *deficiency* of that pair is the number of additional row differences needed for the pair to have δ row differences. Following the example in [2], if a pair of terms T_1, T_2 exists where $T_1 \neq T_2$ but $|(\rho(A, T_1) \cup \rho(A, T_2)) \setminus (\rho(A, T_1) \cap \rho(A, T_2))| = \mu < \delta$, then T_1, T_2 is a violating pair with deficiency $\delta - \mu$. The score of A is the same as the total deficiency of A which is the sum of the deficiencies of all violating pairs. In other words, the score counts the total additonal row differences that all violating pairs collectively need for the separation constraint to be satisfied. Interestingly, when the only constraint for A is separation $\delta = 1$, then this scoring approach is no different from the one described in Section 3.2.

An interesting aspect of Algorithm 2 is that it is easily modified to score the locating array as well as check its validity. When the fourth terminating condition of the algorithm is satisfied, then the number of pairs with one column in *nodeA* and the other in *nodeB* is multiplied by the remaining separation needed, δ. This is then added to a global variable with the total score which is initialized to zero before the initial call to the checker. When all recursive calls are completed, the global variable holds the correct total score for a locating array with any separation constraint δ.

Furthermore, this scoring change is implemented with almost no additional runtime cost.

Table 9 shows factor and level inputs in the first column, and the separation requirements in the second row. The remainder of the table indicates the size, in terms of rows, needed for directed Moser-Tardos resampling to construct a valid locating array within 1000 iterations, satisfying the separation requirement.

Table 9. Locating array sizes using directed Moser-Tardos resampling.

	Number of Rows			
Type	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$
2^{10}	14	19	24	30
2^{15}	17	22	29	34
2^{20}	19	26	31	37
2^{50}	26	33	40	47
2^{75}	28	36	44	50
2^{100}	31	39	46	53
3^{10}	34	46	57	66
3^{15}	40	52	65	73
3^{20}	44	57	69	79
3^{50}	57	70	83	95
3^{75}	62	76	90	103
3^{100}	67	81	94	107
4^{10}	65	86	104	122
4^{15}	76	96	116	133
4^{20}	82	104	122	141
4^{50}	106	129	148	168
4^{75}	116	138	159	179
4^{100}	123	146	166	188
5^{10}	110	141	165	194
5^{15}	126	156	185	212
5^{20}	138	169	198	225
5^{50}	173	208	236	
5^{75}	189	223	256	
5^{100}	202	235	266	
$5^{10}2^{10}$	110	139	172	197

Table 9 was generated using a binary search technique on the locating array size. We began with an upper bound on the number of rows in the array that easily satisfied all constraints, and a lower bound, an empty array with 0 rows. A binary search technique was then used to find the smallest possible size for a locating array to be constructed in 1000 iterations under the specified constraints. In the column under $\delta = 4$, some of the cells are blank because the checker took an extended amount of time in these cases. Not surprisingly, the locating arrays in Table 9 with separation $\delta = 1$ produced by directed Moser-Tardos resampling are larger than those shown in Table 7 and produced by Algorithm 1 which chooses every row carefully to minimize the array score, and ultimately the final array size.

3.4 Summary

The initial greedy approach presented in Algorithm 1 is convenient for creating arrays that satisfy the locating property. It adds rows as needed, and the only necessary input parameter is factor information. However, it is unable to construct locating arrays with additional constraints including higher separation. Instead of modifying Algorithm 1 to incorporate separation, we present randomized approaches that can be applied to construct locating arrays with any additional constraints.

Moser-Tardos resampling given in Algorithm 3 constructs locating arrays with additional constraints. The main requisite of the algorithm is an efficient checker used repeatedly to check if the array satisfies all requirements. Additionally, the algorithm requires an input parameter indicating the size of the locating array.

A modification to Moser-Tardos resampling is directed Moser-Tardos resampling discussed in Section 3.3.2. As shown in Figure 2 and Figure 3, directed Moser-Tardos resampling may construct a locating array in fewer iterations than Moser-Tardos resampling. However, it requires an additional scoring system to indicate how close the array is to satisfying all requirements. Finally, we provide Table 9 to indicate the approximate locating array sizes for several different array types and separation constraints.

Chapter 4

ANALYSIS

Following the construction of a locating array, the experiments indicated by the array are performed, and results are measured as one or more output variables. The results must now be analyzed to determine which factors are relevant. As discussed in Chapter 1, locating arrays grow logarithmically in the number of factors, making the consideration of an order of magnitude more factors in experimentation practical. However, to achieve the logarithmic growth rate, locating arrays can be highly unbalanced, and thus new techniques for the analysis of locating arrays are required.

This chapter discusses two existing approaches for analyzing data collected from experimentation based on a locating array introduced in Section 2.2.2 and Section 2.2.1. It also describes a new time- and space-efficient analysis technique that is able to support large-scale experimentation and cope with noise in measurements. The new analysis technique is then used to validate the results from two screening experiments based on locating arrays, one conducted on the `w-iLab.t` wireless network testbed in Belgium varying 24 factors, and the other conducted in a wireless network simulator varying 75 factors. Our experimental design and analysis techniques are available for use [21] in large-scale screening experiments, the first phase of any goal of experimentation.

4.1 Analysis Motivation

Section 2.2.2 describes an initial existing analysis technique using a statistical approach. The approach could be generalized and automated for analysis in other screening experiments, but does not consider alternate explanations for noise in the system. A second existing analysis technique in Section 2.2.1 provides another approach using a depth-first search (DFS) of an explicit tree. This second approach is general and considers noise, but the algorithm provided is inefficient. The recursive backjumping employed by the DFS algorithm requires many models and decisions to be tracked which makes memory-efficient implementations difficult. The DFS algorithm also provides very little control over the runtime of the algorithm. Our aim is to provide a general, simple, and efficient algorithm for analysis that allows for easy implementation and optimization.

4.2 Introducing a New Approach

We propose a new approach that achieves the same analysis using the same basic "heavy-hitters" method as in existing approaches. However, the algorithm is novel in that it uses a branch-and-bound approach, conducting the search in a breadth-first manner, storing the tree implicitly in a number of priority queues. It bounds the number of models in each queue using R^2. Bounds on the number and size of the queues provide parameters to trade running time and space to achieve exploration of a larger portion of the tree. Pseudocode for the new branch-and-bound BFS tree-based analysis is given in Algorithm 4.

Our new approach borrows many ideas from the previous two approaches, but also introduces new ideas in the process. It uses the same type of “heavy-hitters” algorithm and OMP technique as the existing DFS technique. However, the approach discussed in Section 2.2.1 [6] uses a safety parameter to dictate where to perform back-jumping on the implicit tree. This parameter helps compare multiple other alternative models that each may contain a different number of terms. We eliminate this parameter to simplify the process and switch to a BFS approach, eliminating the need to compare alternative models that may each contain a different number of terms.

Algorithm 4 BFS_Analysis(*terms*, M, *data*, *nTerms*, *nNewModels*, *nModels*)

```
Require: List of candidate terms, compressive sensing matrix, vector of performance
    data, terms per model, number of new models to generate for a particular model,
    number of models to return
Ensure: $nModels$ best models with $nTerms$ terms each
 1: Initialize $nTerms + 1$ priority queues, each with maximum $nModels$
 2: $model_{new} \leftarrow$ [empty model with no terms]
 3: $residuals_{new} \leftarrow data$
 4: enqueue($queue_0, (model_{new}, residuals_{new})$)
 5: for $\ell \leftarrow 0, ..., nTerms - 1$ do
 6:    while $queue_\ell$ has models do
 7:       $(model, residuals) \leftarrow$dequeue($queue_\ell$)
 8:       for $i \leftarrow 1, ..., nNewModels$ do
 9:          $k \leftarrow [i^{th}$ most significant term from $M]$
10:          $model_{new} \leftarrow \mathrm{LS}(terms(model) \cup [terms_k], data)$
11:          $residuals_{new} \leftarrow residuals - model_{new}$
12:          enqueue($queue_{\ell+1}, (model_{new}, residuals_{new})$)
13:       end for
14:    end while
15: end for
16: return $queue_{nTerms}$
```

Ultimately, the new BFS tree analysis algorithm attempts to find the models with the highest R^2 values, each with $nTerms$ terms, in a straightforward manner. It starts by initializing the zeroth priority queue to hold the empty model with $R^2 = 0$. The algorithm then removes the model from the zeroth priority queue, and generates $nNewModels$ models, each with one term from the most significant terms of the compressive sensing matrix, M, selected by OMP. This differs from the DFS tree algorithm by considering multiple models rather than exploring a single model at a time. Least squares is run on each of these models to obtain their R^2 values, and they are then placed in the first priority queue, ranked by R^2.

The process is repeated on the first priority queue to generate new models, each with two terms, that are added to the second priority queue. Each queue only stores the best models by R^2 up to its bound, $nModels$. If a queue is full, then an attempt to enqueue a model with R^2 lower than the model with the lowest R^2 does not succeed. Similarly, an attempt to enqueue a model with R^2 greater than the model with the lowest R^2 evicts that model from the queue. The process is repeated until the final priority queue contains $nModels$ models each with $nTerms$ terms.

The parameters of the BFS tree algorithm, $nModels$, $nNewModels$, and $nTerms$ can be adjusted as desired. Higher values of $nNewModels$ result in more models fit using least squares, thus increasing execution time, but also increasing the likelihood that the models with the best R^2 values are found. Lower values of $nNewModels$ result in fewer models fit, decreasing execution time but possibly resulting in models with lower R^2 values found. Similarly, higher values of $nModels$ lead to a larger bound on the number of models stored and analyzed, increasing execution time. Lower values of $nModels$ risk discarding a model that might become much better, with respect to R^2, when more terms are added. Finally, the parameter $nTerms$ must be chosen carefully to propery fit the model. When $nTerms$ is too small, the analysis does not find all terms relevant to the system and produces poor models, and when $nTerms$ is too large, the analysis overfits the models. In Chapter 6, we discuss choosing $nModels$, $nNewModels$, and $nTerms$ dynamically in future work.

Another issue that must be accounted for is that of duplicate models. It is possible that multiple models with identical terms, but in different orders, are added to the same priority queue. The algorithm must discard such duplicate models.

When the algorithm completes, the final priority queue holds, at most, the number of models given by $nModels$. These models are ordered by R^2 and given as the best models each with $nTerms$ terms.

The execution time and memory usage of Algorithm 4 can be easily adjusted using the parameters $nModels$, $nNewModels$, and $nTerms$. Our approach iterates through $nTerms$ priority queues, extracts $nModels$ from each, finds the best terms using OMP dictated by $nNewModels$, and performs least squares after adding each term. Assuming least squares takes, in the worst case, time $nTerms$, the execution time for this algorithm is $O(\mathit{nTerms}^2 \cdot \mathit{nModels} \cdot \mathit{nNewModels})$. This approach also considers one priority queue at a time, each with size $nModels$. Assuming each model takes constant space, the memory usage is $O(nModels)$.

In screening, we are interested in identifying the most significant factors and two-way interactions impacting performance. One way to determine the screening results is to examine occurrences of the factors in these models and select those occurring most often. We discuss this next.

4.3 Counting Occurrences

The new BFS analysis approach produces and returns the top *nModels* models. In the top models generated, many factors are included multiple times in multiple models. We count the occurrences of each factor in the top *nModels* models, and rank the factors by number of occurrences. Because terms in a model can be interactions between factors, or a factor can appear with different levels, the same factor may occur multiple times in the same model. A factor *occurrence* is defined as any time a factor appears in a model, no matter its level, or if it is part of an interaction. An interaction *occurrence* is defined as any time a pair of factors appear in a model as an interaction, no matter the level of either factor. In the following section, we rank all factors by how often they occur in the top *nModels* models. Interactions are also examined but with a separate ranking. We hypothesize that a significant factor occurs frequently in the top models and is therefore highly ranked, while an insignificant factor rarely appears in the ranking, if at all.

4.4 Validation

We compare our BFS tree algorithm to the DFS tree algorithm in [6]. We use a C++ implementation to generate the top 50 models, with 11 terms each, using our BFS tree algorithm for both voice quality and RF exposure. Finally, we count the occurrences of each factor in the top 50 models and rank all factors by the number of occurrences. We use the same testbed data collected and used in [6]. The full-factorial design for this factor space has over 10^{13} tests while the locating array has only 109 tests.

Table 10 shows the factors ranked by the number of occurrences in our BFS tree results for the voice quality performance metric. The table also indicates the factors selected by the DFS algorithm [6]. Three of the four significant factors for voice quality listed in [6] are the top three factors in our BFS tree results. This indicates a strong correspondence between the output of the DFS and BFS algorithms with the difference likely due to the DFS algorithm exploring a different, or smaller, portion of

Table 10. Significant factors impacting voice quality from data collected from the `w-iLab.t` testbed.

BFS Occurrence Counts		
Count	Factor	In [6]
150	*intCOR*	$\surd$
136	*band*	$\surd$
133	*txpower*	$\surd$
100	*sensing*	
71	*rate*	
48	*udp_mem_pressure*	
48	*ipfrag_low_thresh*	
34	*wmem_max*	
22	*codecBitrate*	$\surd$
16	*txqueuelen*	
15	*mtu*	
14	*channel*	
12	*frameLen*	
9	*wmem_default*	
7	*ROHC*	
6	*codec*	
5	*ipfrag_high_thresh*	
5	*udp_wmem_min*	
4	*udp_rmem_min*	
3	*rmem_max*	
3	*qdisc*	
1	*rmem_default*	
1	*udp_mem_min*	

the search tree since it stops after generating 1024 models. This also likely accounts for the BFS algorithm not agreeing with the fourth significant factor.

Interestingly, our BFS tree analysis identified some potential two-way interactions for voice quality that the DFS tree analysis did not identify. Table 11 shows the top interactions; in this case, they exhibit strong heredity.

Table 12 shows the factors ranked by the number of occurrences in our BFS tree results for the RF exposure performance metric. For exposure, five of the top six

Table 11. Significant two-way interactions impacting voice quality from data collected from `w-iLab.t`.

BFS Occurrence Counts	
Count	Interaction
51	*intCOR* & *band*
49	*intCOR* & *sensing*
49	*sensing* & *band*
48	*udp_mem_pressure* & *ipfrag_low_thresh*
33	*rate* & *band*
30	*rate* & *wmem_max*
11	*txqueuelen* & *frameLen*
9	*txpower* & *channel*
9	*wmem_default* & *codecBitrate*
8	*mtu* & *txpower*

factors match in the two algorithms. The largest number of occurrences of any two-way interaction for exposure was only six, so we do not consider any interaction to be significant.

Table 12. Significant factors impacting RF exposure from data collected from the `w-iLab.t` testbed.

BFS Occurrence Counts		
Count	Factor	In [6]
156	*rate*	$\surd$
152	*txpower*	$\surd$
98	*codecBitrate*	$\surd$
60	*frameLen*	$\surd$
56	*band*	$\surd$
4	*codec*	
4	*ROHC*	
2	*wmem_max*	
2	*ipfrag_low_thresh*	
2	*udp_mem_min*	
2	*qdisc*	
2	*channel*	

The factors screened as significant are plausible. For example, one would expect that transmission power (*txpower*) should have an effect on both exposure and audio quality, and that interference should have an effect on audio quality but not exposure.

Our comparisons show a strong correspondence to the results obtained from the DFS tree algorithm in [6] and our BFS tree algorithm therefore validates the screening analysis in [6]. In addition, our new approach is more memory efficient and provides parameters to control runtime. Indeed, the BFS tree algorithm is able to analyze the even larger-scale data collected from experimentation in simulation described next, where the DFS tree algorithm was unable to complete the analysis because of memory constraints.

In [5], 75 factors of the protocols spanning the MAC to the transport layer, as well as the wireless environment and the simulation environment, having from two to ten values each, were screened in a simulation model of a mobile wireless network. The goal was to determine the significant factors and two-way interactions impacting TCP throughput. The full-factorial design for this factor space is even larger than the testbed experiment; it has over 10^{43} tests in the array! In contrast, the locating array has only 421 tests.

Aldaco et al. [5] perform screening analysis on the data collected from simulation using the method described in Section 2.2.2. A model with 13 terms having 9 unique factors are identified as significant. We use our BFS tree algorithm to perform screening analysis on the same data set. The parameters for our algorithm were set to produce 50 models, each with 13 terms; these models differed very little in R^2 values.

We again counted the occurrences of each factor in the top 50 models and ranked all factors by the number of occurrences. Table 13 shows the resulting ranking and an indicator when the factor is also identified as significant in [5]. Interestingly, the top eight factors, by number of occurrences, are also factors identified by Aldaco et al. [5] as significant. Therefore, eight of the nine factors identified by Aldaco et al. were the top eight factors identified by the BFS tree algorithm. We also counted the occurrences of interactions between two factors and ranked all interactions by the number of occurrences.

Table 14 shows the interactions ranked by the number of occurrences in the BFS tree results and an indicator when the interaction is also identified in [5] as significant. Interestingly, three of the top four interactions, by number of occurrences, were also interactions identified by Aldaco et al. as significant. However, Aldaco et al. did identify a fourth interaction that was not found by the BFS tree algorithm.

Again, our comparisons show a strong correspondence to the results presented in Aldaco et al. [5]. It is therefore reasonable to conclude that our BFS tree analysis algorithm successfully identifies those factors and two-way interactions, that are most significant to the TCP throughput of the mobile wireless network. It is also reasonable to conclude that counting occurrences is a valid approach for determining what factors are most significant from the top *nModels*. A better approach than counting occurrences may exist but we leave this problem for future work in Chapter 6.

4.5 Summary

Algorithm 4 is a simple, general, and efficient analysis approach for the output measurements from locating arrays. The algorithm can be used to analyze any locating array with its corresponding output measurements to find significant factors and interactions. It uses three input parameters that control how many models are

Table 13. Significant factors impacting TCP throughput from data collected from a wireless network simulation.

BFS Occurrence Counts		
Count	Factor	In [5]
148	*MAC_RTSThreshold*	$\surd$
142	*ErrorModel_unit*	$\surd$
100	*TCP_packetSize*	$\surd$
100	*TCP_min_RTO*	$\surd$
100	*ErrorModel_ranvar*	$\surd$
98	*RWP_Area*	$\surd$
53	*ErrorModel_rate*	$\surd$
50	*ARP_flows*	$\surd$
27	*Propagation*	
26	*DSSS_CWMin_CWMax*	
25	*TCP_slow_start_restart*	
16	*TCP_RTTvar_exp*	$\surd$
15	*TCP_maxburst*	
13	*Queue_acksfirst*	
12	*AODV_TTL_START*	
8	*ENER_initialEnergy*	
8	*MAC_ProbeDelay*	
8	*TCP_updated_rttvar*	
5	*TCP_numdupacksFrac*	
3	*AODV_HELLO_INTERVAL*	
3	*TCP_decrease_num*	
3	*MAC_ScanType*	
3	*Queue_DT_queue_in_bytes*	
3	*Queue_interleave*	
2	*Queue_ackfromfront*	
2	*TCP_rttvar_init*	
1	*Queue_DT_summarystats*	
1	*TCP_TRTTVAR_BITS*	

Table 14. Significant two-way interactions impacting TCP throughput in a wireless network simulation.

BFS Occurrence Counts		
Count	Interaction	In [5]
56	*RWP_Area & MAC_RTSThreshold*	
50	*ErrorModel_rate & ErrorModel_unit*	✓
50	*MAC_RTSThreshold & ErrorModel_ranvar*	✓
50	*ErrorModel_unit & ErrorModel_ranvar*	✓
42	*MAC_RTSThreshold & ErrorModel_unit*	
26	*DSSS_CWMin_CWMax & Propagation*	
16	*TCP_min_RTO & TCP_slow_start_restart*	
14	*TCP_maxburst & TCP_RTTvar_exp*	
8	*ARP_flows & TCP_slow_start_restart*	
8	*RWP_Area & Queue_acksfirst*	

stored, how many alternative models to check, and how many terms to add to each model. These parameters are currently static throughout the analysis execution. Future work includes the possibility of modifying these parameters to be dynamic. Furthermore, the algorithm assumes that the significant system factors and interactions follow a "heavy-hitters" pattern. The next chapter investigates this assumption and how the algorithm behaves when the system does not follow this pattern.

Chapter 5

ROBUSTNESS

During the discussion of our new analysis technique in Section 4.2, an assumption was made that the system factors must follow a "heavy-hitters" pattern. We also did not discuss how noise might affect our analysis technique, although noise is always present in real-world applications and often introduced in simulated systems. In this chapter, we discuss "heavy-hitters" and see how robust our analysis is when the assumption is violated. We also discuss the effects of noise in the system on the analysis algorithm. Chapter 3 introduces separation and hypothesizes that higher separation leads to more accurate results in analysis. This chapter investigates how separation might help, particularly when a large amount of noise is introduced. Finally, Section 5.4 discusses an interesting phenomenon that was coincidentally discovered when investigating the effects of noise.

5.1 "Heavy-Hitters" Requirement

In Chapter 4, we discussed how our analysis algorithm assumes a "heavy-hitters" scenario. We call this the "heavy-hitters" requirement. The analysis is thus a method that relies on the assumption that there is one term, a main effect or interaction, that affects the output more significantly than all remaining terms, and when this significant term is removed, there again exists one main effect or interaction that affects the output more significantly than all remaining terms. In other words, the significance of the factors and interactions follows an exponentially decreasing pattern

Table 15. Robustness Scenario 1A - Without Heavy-Hitters

True Model - Does not satisfy "heavy-hitters"	
Coefficient	Term
1	*INTERCEPT*
1	T_3
1	T_4

Partial CS Matrix and Responses					
INTERCEPT	T_1	T_2	T_3	T_4	Response
1	1	-1	-1	-1	1
1	-1	1	-1	-1	1
1	-1	-1	1	-1	-1
1	-1	-1	-1	1	-1

that allows them to be easily identified, one by one, over all other terms. In this section, we begin with a simple example of when the "heavy-hitters" requirement is satisfied, and when it is not. We then move on to more complex examples.

Suppose the true model for a system is given in Table 15. This means that the output of the system is determined by the specific terms listed, along with their coefficients, i.e., the output is a function of the form $a \cdot T_3 + b \cdot T_4 + c \cdot INTERCEPT$. We then attempt to use our analysis algorithm to recover the true terms of the model using only the output of the system. A partial CS matrix along with the output (responses) of the system are also given in Table 15.

A trace of the analysis using Algorithm 4 proceeds:

1. The algorithm begins with a model consisting of only the intercept which is the average of all output values:

$$0 \cdot INTERCEPT$$

2. The residuals are:

$$\begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix}^T$$

3. Absolute dot products calculated for T_1 to T_4 are:

$$2, 2, 2, 2$$

4. Because all absolute dot products are the same, any term may be added depending on noise. Suppose the first term, T_1, is added.
5. Least squares is now run and the model becomes:

$$\frac{1}{3} \cdot INTERCEPT + \frac{2}{3} \cdot T_1$$

6. The residuals are:

$$\begin{bmatrix} 0 & \frac{4}{3} & -\frac{2}{3} & -\frac{2}{3} \end{bmatrix}^T$$

7. Absolute dot products calculated for T_2 to T_4 are:

$$\frac{8}{3}, \frac{4}{3}, \frac{4}{3}$$

8. T_2 has the largest absolute dot product and it is added to the model.
9. Least squares is now run and the model becomes:

$$1 \cdot INTERCEPT + 1 \cdot T_1 + 1 \cdot T_2$$

10. The residuals are:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$$

The analysis using Algorithm 4 stops because it has now explained the response completely. However, the terms in the model recovered do not equal the terms in the true model listed in Table 15. This is because the coefficients are all so close together that they can easily hide each other's effects. Suppose now we use the true model in Table 16 which satisfies the "heavy-hitters" requirement with the same experiments.

A trace of the analysis using Algorithm 4 proceeds:

Table 16. Robustness Scenario 1B - With Heavy-Hitters

True Model - Does satisfy "heavy-hitters"	
Coefficient	Term
1	*INTERCEPT*
4	T_3
10	T_4

Partial CS Matrix and Responses					
INTERCEPT	T_1	T_2	T_3	T_4	Response
1	1	-1	-1	-1	-13
1	-1	1	-1	-1	-13
1	-1	-1	1	-1	-5
1	-1	-1	-1	1	7

1. The algorithm begins with a model consisting of only the intercept which is the average of all output values:

$$-6 \cdot INTERCEPT$$

2. The residuals are:

$$\begin{bmatrix} -7 & -7 & 1 & 13 \end{bmatrix}^T$$

3. Absolute dot products calculated for T_1 to T_4 are:

$$14, 14, 2, 26$$

4. T_4 has the largest absolute dot product and it is added to the model.
5. Least squares is now run and the model becomes:

$$-\frac{5}{3} \cdot INTERCEPT + \frac{26}{3} \cdot T_4$$

6. The residuals are:

$$\begin{bmatrix} -\frac{8}{3} & \frac{8}{3} & \frac{16}{3} & 0 \end{bmatrix}^T$$

7. Absolute dot products calculated for T_1 to T_3 are:

$$\frac{16}{3}, \frac{16}{3}, \frac{32}{3}$$

8. T_3 has the largest absolute dot product and it is added to the model.

9. Least squares is now run and the model becomes:

$$1 \cdot INTERCEPT + 4 \cdot T_3 + 10 \cdot T_4$$

10. The residuals are:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$$

The analysis using Algorithm 4 has now explained the response completely, and it has recovered the same terms as the true model listed in Table 16. Table 15 illustrates how failure to meet the “heavy-hitters” requirement can lead to the recovery of models that appear good, but are not the actual model. Table 16 then shows how meeting the “heavy-hitters” requirement can lead to the recovery of the actual model. However, these scenarios are extremely basic, and we now turn to more interesting and realistic examples. We use a systematic study on synthetic data because the limitations of our recovery must be understood. In real systems, the function we are trying to recover is unknown.

In our more complex examples, we first create a locating array for a set of factors and their levels. Next, we create a model with a set of main effects or interactions. Artificial experimental output measurements are then generated using the chosen true model. The output measurements are initially generated without any noise. Finally, we run our analysis software with the constructed locating array and the measured responses, and evaluate how well the analysis can recover the chosen true model.

Suppose we are interested in a screening experiment with 100 factors, $F_1, F_2, \ldots, F_{100}$. Each of these factors can be set to 3 levels, v_1, v_2, v_3. Significant terms can be main effects or interactions. The locating array constructed with separation parameter $\delta = 1$ has just 70 rows. This locating array is used in the sequel unless specified otherwise.

Consider the scenario with the true network model in Table 17. The true system model is shown at the top of the table, and includes main effects as well as interactions which are indicated by two main effects separated by an ampersand. The parameters for the analysis algorithm are provided just below the true system model. Finally, the occurrence counts are shown at the bottom, and the checkmarks in the last column indicate the factors that can also be found in the true model. The "heavy-hitters" requirement indicates that the coefficients should follow an exponentially decreasing pattern, but it is clear that the true model completely fails to satisfy this. In fact, every single term carries the same coefficient, 0.1, and the coefficients exhibit 0% decrease between them. Yet the analysis found all nine factors in the true model and placed them first. However, the four factors ranked six through nine have occurrence counts that are less than 50 indicating that these factors did not appear in every model. Therefore, although it ranked them first, the true factors were not included in all of the generated models.

The scenario in Table 18 uses a true system model in which coefficients decrease by approximately 9%. The true model is now closer to satisfying the "heavy-hitters" requirement as the term coefficients are no longer the same. Again, the analysis found all nine factors in the true model and placed them first. Of those, only the factors ranked eight and nine have occurrence counts that are less than 50. This means that only two of the true factors were not found in all of the generated models. This is an improvement over the previous scenario. The last two factors on the occurrence count list are F_{23} and F_{69}. These factors are only found in one interaction with a small coefficient (0.12) in the true model. Hence it is not surprising that they were listed last.

Table 17. Robustness Scenario 2A - Heavy-Hitters (0% Decrease)

True Model - Coefficients exhibit 0% decrease	
Coefficient	Term
0.1	$F_{29} = v_1$
0.1	$F_{98} = v_3$ & $F_{34} = v_2$
0.1	$F_{50} = v_2$
0.1	$F_{22} = v_1$
0.1	$INTERCEPT$
0.1	$F_{69} = v_1$ & $F_{23} = v_1$
0.1	$F_{10} = v_2$
0.1	$F_{82} = v_1$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 8$

Occurrence Counts			
Rank	Count	Factor	True
1	50	F_{82}	✓
2	50	F_{50}	✓
3	50	F_{29}	✓
4	50	F_{22}	✓
5	50	F_{10}	✓
6	38	F_{23}	✓
7	35	F_{34}	✓
8	34	F_{98}	✓
9	30	F_{69}	✓
10	6	F_6	

The scenario in Table 19 brings the true model even closer to satisfying the "heavy-hitters" requirement. All coefficients decrease by 33%. Furthermore, the analysis now not only finds all nine factors in the true model, but all significant factors are now found 50 times, and the tenth factor listed is found only twice. This is an extremely sharp drop-off and delineates the nine significant factors from the tenth insignificant factor. Such a table of occurrence counts appears useful for screening.

The next section (5.2) explores how noise affects occurrence counts, and we explore scenarios that satisfy the "heavy-hitters" requirement even more strongly. For the sake

Table 18. Robustness Scenario 2B - Heavy-Hitters (9% Decrease)

True Model - Coefficients exhibit 9% decrease	
Coefficient	Term
0.19	$F_{29} = v_1$
0.18	$F_{98} = v_3$ & $F_{34} = v_2$
0.16	$F_{50} = v_2$
0.15	$F_{22} = v_1$
0.13	$INTERCEPT$
0.12	$F_{69} = v_1$ & $F_{23} = v_1$
0.11	$F_{10} = v_2$
0.1	$F_{82} = v_1$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 8$

Occurrence Counts			
Rank	Count	Factor	True
1	50	F_{98}	$\surd$
2	50	F_{82}	$\surd$
3	50	F_{50}	$\surd$
4	50	F_{34}	$\surd$
5	50	F_{29}	$\surd$
6	50	F_{22}	$\surd$
7	50	F_{10}	$\surd$
8	23	F_{23}	$\surd$
9	13	F_{69}	$\surd$
10	10	F_2	

of completeness, however, and because we contrast against them in the next section, we provide here two more scenarios in Table 20 and Table 21. The scenario in Table 20 changes the coefficients to decrease by 50%, and the coefficients in Table 21 exhibit a 60% decrease.

These final two scenarios both satisfy the "heavy-hitters" requirement. Unsurprisingly, in both cases the analysis finds all nine factors that are present in the true models. Furthermore, the drop-off in occurrence counts between the ninth factor and the tenth factor is clear and indicates which factors are significant. We can therefore

Table 19. Robustness Scenario 2C - Heavy-Hitters (33% Decrease)

True Model - Coefficients exhibit 33% decrease	
Coefficient	Term
1.71	$F_{29} = v_1$
1.14	$F_{98} = v_3$ & $F_{34} = v_2$
0.76	$F_{50} = v_2$
0.51	$F_{22} = v_1$
0.34	$INTERCEPT$
0.23	$F_{69} = v_1$ & $F_{23} = v_1$
0.15	$F_{10} = v_2$
0.1	$F_{82} = v_1$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 8$

Occurrence Counts			
Rank	Count	Factor	True
1	51	F_{69}	$\surd$
2	51	F_{50}	$\surd$
3	51	F_{34}	$\surd$
4	51	F_{23}	$\surd$
5	51	F_{22}	$\surd$
6	51	F_{10}	$\surd$
7	50	F_{98}	$\surd$
8	50	F_{82}	$\surd$
9	50	F_{29}	$\surd$
10	2	F_{99}	

conclude that when no noise is introduced into the system and the coefficients satisfy "heavy-hitters", the analysis algorithm is able to recover the true model accurately. However, when the coefficients do not satisfy "heavy-hitters", the analysis algorithm can produce results from which it can be difficult to draw conclusions with confidence.

5.2 Effects of Noise

In the previous section, we explored how the analysis algorithm can perform better when the true model satisfies the "heavy-hitters" requirement. In the best cases, for

Table 20. Robustness Scenario 2D - Heavy-Hitters (50% Decrease)

True Model - Coefficients exhibit 50% decrease	
Coefficient	Term
12.8	$F_{29} = v_1$
6.4	$F_{98} = v_3$ & $F_{34} = v_2$
3.2	$F_{50} = v_2$
1.6	$F_{22} = v_1$
0.8	$INTERCEPT$
0.4	$F_{69} = v_1$ & $F_{23} = v_1$
0.2	$F_{10} = v_2$
0.1	$F_{82} = v_1$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 8$

Occurrence Counts			
Rank	Count	Factor	True
1	51	F_{69}	$\surd$
2	51	F_{50}	$\surd$
3	51	F_{34}	$\surd$
4	51	F_{23}	$\surd$
5	51	F_{22}	$\surd$
6	51	F_{10}	$\surd$
7	50	F_{98}	$\surd$
8	50	F_{82}	$\surd$
9	50	F_{29}	$\surd$
10	2	F_{99}	

example in Table 21, the analysis finds all significant factors and indicates those that are insignificant. This is shown by the extreme drop-off in the occurrence count between the ninth and tenth factors in the ranking. However, such a clear drop-off in occurrence counts is not seen in the analysis of many screening experiments. Noise in the system often affects the data collected in all experiments causing the drop-off to be more gradual, making it more difficult to distinguish the factors that are significant from those that are insignificant.

In this section, we explore how noise affects our analysis and how different types of true models are affected in different ways. Random uniform noise is therefore added

Table 21. Robustness Scenario 2E - Heavy-Hitters (60% Decrease)

True Model - Coefficients exhibit 60% decrease	
Coefficient	Term
61.04	$F_{29} = v_1$
24.41	$F_{98} = v_3$ & $F_{34} = v_2$
9.77	$F_{50} = v_2$
3.91	$F_{22} = v_1$
1.56	$INTERCEPT$
0.63	$F_{69} = v_1$ & $F_{23} = v_1$
0.25	$F_{10} = v_2$
0.1	$F_{82} = v_1$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 8$

Occurrence Counts			
Rank	Count	Factor	True
1	51	F_{69}	$\surd$
2	51	F_{50}	$\surd$
3	51	F_{34}	$\surd$
4	51	F_{23}	$\surd$
5	51	F_{22}	$\surd$
6	50	F_{98}	$\surd$
7	50	F_{82}	$\surd$
8	50	F_{29}	$\surd$
9	50	F_{10}	$\surd$
10	2	F_{99}	

to our output data measurements and we then attempt to recover the significant factors. The magnitude of the noise is characterized as a percentage of the range of all output measurements. For example, if the smallest output measurement is 1 and the greatest is 11 (before adding artificial noise), then the range is $11 - 1 = 10$. And if 10% artificial noise is added, then 10% of 10 is 1, and the smallest measurement (that was previously 1) becomes a uniform random variable with mean 1, U(0.5,1.5). Similarly, 20% artificial noise causes the smallest output measurement to become a uniform random variable U(0,2). For a given run of our analysis, all random variables are sampled, and their values are used during the execution. In the extreme case with 100% artificial noise, all output measurements could be the same in any given analysis run, in which case no knowledge at all could be discerned from the data by any algorithm.

In this section, we investigate how earlier scenarios might change with the addition of noise. We investigate adding 10% noise to scenarios 2A, 2C, 2D and 2E. This shows us how noise affects true models that satisfy the "heavy-hitters" requirement and how it affects those that do not. In particular, we observe an interesting phenomenon when noise is added to a model that strongly satisfies the "heavy-hitters" requirement.

First we produce scenario 2F in Table 22 by adding 10% noise to scenario 2A in Table 17 which does not satisfy the "heavy-hitters" requirement. The occurrence counts look similar to those from scenario 2A where no noise was added. Here, a gradual drop-off is seen in the occurrence counts, and some of the true factors have slightly different occurrence counts. Particularly, the factors involved in interactions, $F_{34}, F_{98}, F_{23}, F_{69}$, show uncertainty in the occurrence counts because they each must share a coefficient with another factor. However, the top nine factors listed correspond to the nine factors in the true model, and the noise does not have a major effect.

We produce the scenario 2G in Table 23 by adding 10% noise to scenario 2C which better satisfies the "heavy-hitters" requirement. In this scenario, not only are the occurrence counts smaller than those from scenario 2C, but three of the true factors do not appear on the list while two factors not in the true model do. This is in sharp contrast to scenario 2C where all nine true factors appeared as the first nine factors in the occurrence counts table and there was an extremely sharp drop-off after the nine factors. The three factors in the true model missing from the occurrence counts, F_{69}, F_{23}, F_{82} have the smallest coefficients, and are replaced by two factors that are not in the true model at all, F_{14}, F_7. This scenario is more affected by noise than

Table 22. Robustness Scenario 2F - Scenario 2A with Noise (10%)

True Model : Noise 10%	
Coefficient	Term
0.1	$F_{29} = v_1$
0.1	$F_{98} = v_3$ & $F_{34} = v_2$
0.1	$F_{50} = v_2$
0.1	$F_{22} = v_1$
0.1	$INTERCEPT$
0.1	$F_{69} = v_1$ & $F_{23} = v_1$
0.1	$F_{10} = v_2$
0.1	$F_{82} = v_1$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 8$

Occurrence Counts			
Rank	Count	Factor	True
1	50	F_{82}	$\surd$
2	50	F_{50}	$\surd$
3	50	F_{29}	$\surd$
4	50	F_{22}	$\surd$
5	50	F_{10}	$\surd$
6	41	F_{34}	$\surd$
7	40	F_{98}	$\surd$
8	23	F_{23}	$\surd$
9	21	F_{69}	$\surd$
10	8	F_{2}	

scenario 2F in Table 22 because the noise tends to “drown” out terms with coefficients that are small in comparison to the others.

The third scenario adds 10% noise to scenario 2D to produce scenario 2H in Table 24 which satisfies the “heavy-hitters” requirement fairly well. Now, only five factors from the true model appear as the first five factors in the occurrence counts table. There is no indication from the occurrence counts that the other four factors in the true model are significant at all. This is again in sharp contrast with scenario 2D where all nine true factors are clearly significant from the occurrence counts table.

Table 23. Robustness Scenario 2G - Scenario 2C with Noise (10%)

True Model : Noise 10%	
Coefficient	Term
1.71	$F_{29} = v_1$
1.14	$F_{98} = v_3$ & $F_{34} = v_2$
0.76	$F_{50} = v_2$
0.51	$F_{22} = v_1$
0.34	$INTERCEPT$
0.23	$F_{69} = v_1$ & $F_{23} = v_1$
0.15	$F_{10} = v_2$
0.1	$F_{82} = v_1$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 8$

Occurrence Counts			
Rank	Count	Factor	True
1	58	F_{10}	$\surd$
2	55	F_{29}	$\surd$
3	50	F_{98}	$\surd$
4	50	F_{50}	$\surd$
5	50	F_{34}	$\surd$
6	50	F_{22}	$\surd$
7	34	F_{14}	
8	24	F_{7}	

Again, the four true factors missing from the occurrence counts, F_{69}, F_{23}, F_{10} and F_{82} have the smallest coefficients.

Scenario 2I adds 10% noise to scenario 2E which most strongly satisfies the "heavy-hitters" requirement. Similar to the previous scenario, only the five factors with the largest coefficients from the true model are the top five factors in the occurrence counts. The four factors with the smallest coefficients cannot be found in the occurrence counts table. This is again in sharp contrast with scenario 2F where all nine factors in the true model were clearly found significant in the occurrence counts table.

These previous four scenarios 2F, 2G, 2H and 2I show an interesting trend. As the "heavy-hitters" requirement is more and more strongly satisfied in scenarios

Table 24. Robustness Scenario 2H - Scenario 2D with Noise (10%)

True Model : Noise 10%	
Coefficient	Term
12.8	$F_{29} = v_1$
6.4	$F_{98} = v_3$ & $F_{34} = v_2$
3.2	$F_{50} = v_2$
1.6	$F_{22} = v_1$
0.8	$INTERCEPT$
0.4	$F_{69} = v_1$ & $F_{23} = v_1$
0.2	$F_{10} = v_2$
0.1	$F_{82} = v_1$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 8$

Occurrence Counts			
Rank	Count	Factor	True
1	68	F_{29}	$\surd$
2	60	F_{50}	$\surd$
3	52	F_{34}	$\surd$
4	50	F_{98}	$\surd$
5	50	F_{22}	$\surd$
6	32	F_{92}	
7	21	F_{42}	
8	20	F_{44}	
9	14	F_{14}	
10	13	F_{64}	
11	12	F_{76}	
12	12	F_{66}	
13	9	F_{70}	

with noise, the true factors with smallest coefficients (least affecting the true model) start to disappear from the occurrence counts and it becomes harder, or impossible, to distinguish them as significant factors. This is interesting because when these same models are not affected by noise, it is easiest to distinguish the less significant factors when the “heavy-hitters” requirement is most strongly satisfied. Therefore, it appears that noise more highly affects the true models that more strongly satisfy the “heavy-hitters” requirement. This is simply because even small amounts of noise

Table 25. Robustness Scenario 2I - Scenario 2E with Noise (10%)

True Model : Noise 10%	
Coefficient	Term
61.04	$F_{29} = v_1$
24.41	$F_{98} = v_3$ & $F_{34} = v_2$
9.77	$F_{50} = v_2$
3.91	$F_{22} = v_1$
1.56	$INTERCEPT$
0.63	$F_{69} = v_1$ & $F_{23} = v_1$
0.25	$F_{10} = v_2$
0.1	$F_{82} = v_1$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 8$

Occurrence Counts			
Rank	Count	Factor	True
1	51	F_{98}	$\surd$
2	51	F_{34}	$\surd$
3	50	F_{50}	$\surd$
4	50	F_{29}	$\surd$
5	50	F_{22}	$\surd$
6	36	F_{15}	
7	28	F_{32}	
8	28	F_{12}	
9	27	F_{30}	
10	26	F_{95}	
11	12	F_{55}	
12	11	F_{42}	
13	11	F_5	
14	9	F_{87}	
15	8	F_{66}	

easily overpower and drown out the less significant factors. More specifically, when a coefficient is not greater than the noise in the system (dictated by the term with the largest coefficient), then its effects are likely unrecoverable.

Table 26. Robustness Scenario 3A - Separation to Cope With Noise

True Model	
Coefficient	Term
0.5	$INTERCEPT$
0.78	$F_4 = v_1$
4.02	$F_9 = v_1$
0.99	$F_5 = v_2$ & $F_2 = v_1$
2.49	$F_8 = v_2$ & $F_4 = v_1$

Analysis Parameters		
$nModels = 25$	$nNewModels = 50$	$nTerms = 5$

5.3 Separation to Cope with Noise

In Section 5.2, there are scenarios where noise causes serious problems in recovery of the true model with the analysis algorithm. More specifically, when the “heavy-hitters” requirement is satisfied, it is difficult to recover all true factors in the presence of noise. However, in Chapter 3, we introduced the possibility that higher separation in the locating array might provide more trustworthy results. This section explores this possibility and strives to illustrate how higher separation can help our analysis. We use the parameter δ to define the separation of the locating array.

We begin with a few scenarios to illustrate the basic effects of separation. We use the same true model and factors, and compare how two locating arrays respond to different levels of noise when running analysis. Both locating arrays have 10 factors, $F_1, F_2, \ldots, F_{10}$, each with three levels, v_1, v_2, v_3. The first locating array has separation $\delta = 1$ (a standard locating array) and contains 28 rows, while the second locating array has separation $\delta = 3$ and contains 58 rows. The true model along with the analysis parameters are given in scenario 3A in Table 26. This scenario satisfies the “heavy-hitters” requirement fairly well since most coefficients are well differentiated.

The analysis is first executed with no noise and the occurrence counts are shown

Table 27. Separation to Cope With Noise (No Noise)

Occurrence Counts ($\delta = 1$)			
Rank	Count	Factor	True
1	49	F_4	$\surd$
2	28	F_8	$\surd$
3	27	F_9	$\surd$
4	24	F_2	$\surd$
5	23	F_5	$\surd$
6	3	F_{10}	
7	3	F_7	
8	2	F_3	
9	2	F_1	
10	1	F_6	

Occurrence Counts ($\delta = 3$)			
Rank	Count	Factor	True
1	51	F_4	$\surd$
2	29	F_8	$\surd$
3	28	F_9	$\surd$
4	15	F_5	$\surd$
5	14	F_2	$\surd$
6	5	F_7	

in Table 27. Those for $\delta = 1$ are shown first followed by those for $\delta = 3$. Next, Table 28, Table 29 and Table 30 show occurrence counts for noise increasing from 10% to 30% for $\delta = 1$ and $\delta = 3$. Table 28 shows all true factors first and then a distinct drop-off. Interestingly, Table 29 misses one of the true factors when $\delta = 1$, but Table 30 again shows all of the true factors first and then a drop-off. The missing factor F_5 in Table 29 is because the noise is random and eliminated its effects in that particular run. F_5 is also part of an interaction with a smaller coefficient and is therefore more susceptible to noise.

With 40% noise in Table 31, the locating array with separation $\delta = 1$ fails to find

Table 28. Separation to Cope With Noise (10%)

Occurrence Counts ($\delta = 1$)			
Rank	Count	Factor	True
1	48	F_4	$\surd$
2	28	F_9	$\surd$
3	26	F_8	$\surd$
4	23	F_5	$\surd$
5	23	F_2	$\surd$
6	6	F_3	
Occurrence Counts ($\delta = 3$)			
Rank	Count	Factor	True
1	47	F_4	$\surd$
2	28	F_9	$\surd$
3	28	F_8	$\surd$
4	26	F_5	$\surd$
5	26	F_2	$\surd$
6	4	F_{10}	

Table 29. Separation to Cope With Noise (20%)

Occurrence Counts ($\delta = 1$)			
Rank	Count	Factor	True
1	57	F_4	$\surd$
2	34	F_8	$\surd$
3	26	F_9	$\surd$
4	26	F_2	$\surd$
5	6	F_1	
Occurrence Counts ($\delta = 3$)			
Rank	Count	Factor	True
1	41	F_4	$\surd$
2	30	F_8	$\surd$
3	27	F_5	$\surd$
4	26	F_9	$\surd$
5	26	F_2	$\surd$
6	6	F_3	

Table 30. Separation to Cope With Noise (30%)

Occurrence Counts ($\delta = 1$)			
Rank	Count	Factor	True
1	48	F_4	$\surd$
2	33	F_9	$\surd$
3	29	F_8	$\surd$
4	20	F_5	$\surd$
5	16	F_2	$\surd$
6	8	F_7	

Occurrence Counts ($\delta = 3$)			
Rank	Count	Factor	True
1	59	F_4	$\surd$
2	40	F_8	$\surd$
3	26	F_9	$\surd$
4	18	F_2	$\surd$
5	11	F_5	$\surd$
6	6	F_{10}	

all true factors and it introduces a false factor. However, the locating array with separation $\delta = 3$ continues to show all five true factors as the top five factors. The locating array with higher separation appears to do a better job of finding true factors as the other locating array begins to fail.

Now we see how much noise the locating array with higher separation can tolerate before its occurrence counts begin to break down. Table 32 shows the occurrence counts for 50% noise. These occurrence counts are still correct. Those in Table 33 are for 60% noise. A false factor has now been introduced with 60% noise, but the occurrence counts still show all five true factors, though not as the top five.

The occurrence counts in Table 34 are for 70% noise. These occurrence counts still show all five true factors but they are now significantly off. Multiple false factors are found and are ranked fairly high. As expected, the true factors are ranked lower in the

Table 31. Separation to Cope With Noise (40%)

Occurrence Counts ($\delta = 1$)			
Rank	Count	Factor	True
1	49	F_4	$\surd$
2	31	F_8	$\surd$
3	28	F_9	$\surd$
4	17	F_6	
5	17	F_2	$\surd$
6	8	F_3	

Occurrence Counts ($\delta = 3$)			
Rank	Count	Factor	True
1	38	F_5	$\surd$
2	27	F_9	$\surd$
3	26	F_8	$\surd$
4	26	F_4	$\surd$
5	26	F_2	$\surd$
6	10	F_6	

Table 32. Separation to Cope With Noise (50%)

Occurrence Counts ($\delta = 3$)			
Rank	Count	Factor	True
1	34	F_9	$\surd$
2	26	F_4	$\surd$
3	24	F_5	$\surd$
4	24	F_2	$\surd$
5	23	F_8	$\surd$
6	16	F_7	
7	8	F_{10}	

Table 33. Separation to Cope With Noise (60%)

Occurrence Counts ($\delta = 3$)			
Rank	Count	Factor	True
1	34	F_5	$\surd$
2	29	F_9	$\surd$
3	26	F_8	$\surd$
4	26	F_4	$\surd$
5	15	F_7	
6	14	F_2	$\surd$

Table 34. Separation to Cope With Noise (70%)

Occurrence Counts ($\delta = 3$)			
Rank	Count	Factor	True
1	44	F_9	✓
2	32	F_7	
3	25	F_3	
4	22	F_8	✓
5	15	F_4	✓
6	13	F_5	✓
7	11	F_2	✓
8	11	F_1	
9	2	F_{10}	

occurrence counts as more noise is added to the system. Furthermore, the drop-off in the occurrence counts becomes less distinct as more noise is added. Therefore, when the true model is unknown, the drop-off in the occurrence counts can indicate the certainty of the results.

Table 35 shows the occurrence counts with 80% noise. The occurrence counts from 80% noise also introduce false factors, but all five true factors still seem to be significant. Therefore, while the locating array with separation $\delta = 1$ begins to fail with 20% and 40% noise, the locating array with separation $\delta = 3$ still performs well through 60% noise. This supports the hypothesis that higher separation, at least in some cases, leads to more accurate recovery.

Recall that in Section 5.2, scenarios 2G, 2H, and 2I were heavily affected by noise and many true factors were not found in the occurrence counts. In the remainder of this section, a new locating array with a higher level of separation $\delta = 4$ is used to check if separation improves the recovery. The same true models are used and the first is given in Table 36.

Table 35. Separation to Cope With Noise (80%)

Occurrence Counts ($\delta = 3$)			
Rank	Count	Factor	True
1	31	F_9	✓
2	28	F_3	
3	26	F_8	✓
4	24	F_6	
5	22	F_4	✓
6	15	F_5	✓
7	11	F_2	✓
8	7	F_1	

Table 36. Robustness Scenario 3B - Adding Separation to Scenario 2G

True Model : Noise 10%	
Coefficient	Term
1.71	$F_{29} = v_1$
1.14	$F_{98} = v_3$ & $F_{34} = v_2$
0.76	$F_{50} = v_2$
0.51	$F_{22} = v_1$
0.34	$INTERCEPT$
0.23	$F_{69} = v_1$ & $F_{23} = v_1$
0.15	$F_{10} = v_2$
0.1	$F_{82} = v_1$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 8$

Occurrence Counts ($\delta = 4$)			
Rank	Count	Factor	True
1	51	F_{82}	✓
2	51	F_{34}	✓
3	51	F_{22}	✓
4	50	F_{98}	✓
5	50	F_{69}	✓
6	50	F_{50}	✓
7	50	F_{29}	✓
8	50	F_{23}	✓
9	44	F_{10}	✓
10	13	F_6	

The occurrence counts in Table 36 rank all nine true factors as the top nine significant factors. This is a significant improvement over the locating array with $\delta = 1$ used that resulted in only six true factors occurring. It is apparent that higher separation helps recover more true factors.

The second true model that caused trouble previously in Section 5.2 is shown in Table 37 with the new occurrence counts after using a locating array with separation $\delta = 4$. Now, the occurrence counts show six true factors even with separation $\delta = 4$. This is a very slight improvement over the locating array with separation $\delta = 1$ in Section 5.2 where just five true factors were shown. It is interesting that even though higher separation was used, the true factors could still not be recovered from a true model that strongly satisfies the "heavy-hitters" requirement. This is because the noise in the system (dictated by the term with the largest coefficient), overwhelms the terms with smaller coefficients regardless of the separation.

The third true model that caused issues in Section 5.2 is shown in Table 38 along with the new occurrence counts. These occurrence counts are almost identical to those from the locating array with lower separation. The same five true factors are selected as the top five significant factors, but the four remaining true factors are lost. This suggests that even with higher separation, if a true model strongly satisfies the "heavy-hitters" requirement, then the true factors with smallest coefficients may not be recovered. Furthermore, although higher separation helps recovery in many cases, there are some scenarios where it does not seem to help as much such as in Table 38, where the terms with smaller coefficients are overwhelmed by the noise in the system which is determined by the terms with larger coefficients.

Table 37. Robustness Scenario 3C - Adding Separation to Scenario 2H

True Model : Noise 10%	
Coefficient	Term
12.8	$F_{29} = v_1$
6.4	$F_{98} = v_3$ & $F_{34} = v_2$
3.2	$F_{50} = v_2$
1.6	$F_{22} = v_1$
0.8	$INTERCEPT$
0.4	$F_{69} = v_1$ & $F_{23} = v_1$
0.2	$F_{10} = v_2$
0.1	$F_{82} = v_1$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 8$

Occurrence Counts ($\delta = 4$)			
Rank	Count	Factor	True
1	53	F_{34}	$\surd$
2	53	F_{22}	$\surd$
3	52	F_{29}	$\surd$
4	51	F_{98}	$\surd$
5	50	F_{50}	$\surd$
6	49	F_3	
7	45	F_{78}	
8	36	F_{10}	$\surd$
9	19	F_{59}	

5.4 Systems Involving a Large Number of Terms

This section discusses an interesting observation that was encountered when investigating robustness and factor recovery. These final scenarios show how a large number of terms in the true model can cause issues with the recovery, even with no actual noise added to the true model. We use a locating array with 100 factors, $F_1, F_2, \ldots, F_{100}$, and each of these factors can be set to three levels, v_1, v_2, v_3. None of the scenarios in this section introduces any noise into the system.

The first scenario is given in Table 39 and uses a true model with 16 terms consisting of both main effects and interactions. The true model does not strongly

Table 38. Robustness Scenario 3D - Adding Separation to Scenario 2I

True Model	
Coefficient	Term
61.04	$F_{29} = v_1$
24.41	$F_{98} = v_3$ & $F_{34} = v_2$
9.77	$F_{50} = v_2$
3.91	$F_{22} = v_1$
1.56	$INTERCEPT$
0.63	$F_{69} = v_1$ & $F_{23} = v_1$
0.25	$F_{10} = v_2$
0.1	$F_{82} = v_1$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 8$

Occurrence Counts ($\delta = 4$)			
Rank	Count	Factor	True
1	62	F_{29}	$\surd$
2	56	F_{22}	$\surd$
3	50	F_{98}	$\surd$
4	50	F_{50}	$\surd$
5	50	F_{34}	$\surd$
6	33	F_{78}	
7	33	F_{25}	
8	24	F_4	
9	18	F_{82}	
10	18	F_{66}	
11	17	F_{83}	
12	16	F_{39}	
13	13	F_{76}	
14	11	F_{60}	

satisfy the "heavy-hitters" requirement since some coefficients are close to others. But without any noise, scenario 2A in Table 17 suggests that the significant factors might still be ranked at the top of the occurrence counts. However, the occurrence counts completely fail to recover the significant factors, and instead, show factors that are seemingly random. Only five of the top ten ranked factors are actually significant in the true model.

Table 39 is likely unable to recover the significant factors because the many terms obscure the effects of each other. In the remainder of this section, we remove two terms at a time, beginning with the terms with the smallest coefficients, and observe how the recovery is affected. As terms are removed, they stop obscuring the effects of each other and the recovery is much more successful.

The second scenario is given in Table 40 and uses the same true model as the previous scenario but without the two interactions with the smallest coefficients. There are now 14 terms in the true model and the analysis is run once again with all parameters remaining the same. Now, seven of the top ten ranked factors in the occurrence counts are significant in the true model. Although this is an improvement over the previous scenario, the occurrence counts still fail to recover all significant factors.

The next scenario given in Table 41 uses the same true model as the previous scenario 4B but without the two main effects with the smallest coefficients. There are now 12 terms in the true model and everything else remains the same. The occurrence counts, however, again fail to recover the significant factors.

The scenario in Table 42 uses the same true model as the previous scenario 4C but

Table 39. Robustness Scenario 4A - Systems Involving a Large Number of Terms (16 Terms, $nTerms = 11$)

True Model	
Coefficient	Term
1.68472	$INTERCEPT$
4.74096	$F_{50} = v_2$
4.61089	$F_{98} = v_1$
3.51114	$F_{22} = v_1$
3.06619	$F_{41} = v_1$
2.88918	$F_{68} = v_2$
2.70801	$F_2 = v_3$
2.02174	$F_{82} = v_1$
1.66709	$F_{10} = v_2$
0.645456	$F_{29} = v_1$
0.505764	$F_{43} = v_3$
3.84451	$F_{74} = v_2$ & $F_1 = v_1$
2.65065	$F_{67} = v_3$ & $F_{12} = v_2$
2.6396	$F_{45} = v_2$ & $F_{12} = v_2$
1.90092	$F_{69} = v_1$ & $F_{23} = v_1$
0.113597	$F_{98} = v_3$ & $F_{34} = v_2$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 11$

Occurrence Counts			
Rank	Count	Factor	True
1	97	F_5	
2	79	F_1	$\surd$
3	77	F_{22}	$\surd$
4	64	F_{57}	
5	64	F_{34}	$\surd$
6	50	F_{98}	$\surd$
7	50	F_{88}	
8	50	F_{50}	$\surd$
9	50	F_{11}	
10	48	F_{70}	
11	48	F_{62}	
12	48	F_{47}	
13	46	F_{39}	
14	33	F_{31}	
15	14	F_{77}	
16	12	F_{12}	$\surd$
17	10	F_{68}	$\surd$
⋮	⋮	⋮	

Table 40. Robustness Scenario 4B - Systems Involving a Large Number of Terms (14 Terms, $nTerms = 11$)

True Model	
Coefficient	Term
1.68472	$INTERCEPT$
4.74096	$F_{50} = v_2$
4.61089	$F_{98} = v_1$
3.51114	$F_{22} = v_1$
3.06619	$F_{41} = v_1$
2.88918	$F_{68} = v_2$
2.70801	$F_2 = v_3$
2.02174	$F_{82} = v_1$
1.66709	$F_{10} = v_2$
0.645456	$F_{29} = v_1$
0.505764	$F_{43} = v_3$
3.84451	$F_{74} = v_2$ & $F_1 = v_1$
2.65065	$F_{67} = v_3$ & $F_{12} = v_2$
2.6396	$F_{45} = v_2$ & $F_{12} = v_2$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 11$

Occurrence Counts			
Rank	Count	Factor	True
1	77	F_{41}	✓
2	59	F_2	✓
3	54	F_{98}	✓
4	50	F_{50}	✓
5	50	F_{22}	✓
6	38	F_8	
7	38	F_1	✓
8	37	F_{12}	✓
9	35	F_{37}	
10	34	F_{88}	
11	34	F_{67}	✓
12	32	F_{73}	
⋮	⋮	⋮	
19	17	F_{64}	
20	16	F_{82}	✓
21	14	F_{97}	
22	14	F_{21}	
23	13	F_{94}	
24	10	F_{74}	✓
⋮	⋮	⋮	

Table 41. Robustness Scenario 4C - Systems Involving a Large Number of Terms (12 Terms, $nTerms = 11$)

True Model	
Coefficient	Term
1.68472	$INTERCEPT$
4.74096	$F_{50} = v_2$
4.61089	$F_{98} = v_1$
3.51114	$F_{22} = v_1$
3.06619	$F_{41} = v_1$
2.88918	$F_{68} = v_2$
2.70801	$F_2 = v_3$
2.02174	$F_{82} = v_1$
1.66709	$F_{10} = v_2$
3.84451	$F_{74} = v_2$ & $F_1 = v_1$
2.65065	$F_{67} = v_3$ & $F_{12} = v_2$
2.6396	$F_{45} = v_2$ & $F_{12} = v_2$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 11$

Occurrence Counts			
Rank	Count	Factor	True
1	53	F_{67}	$\surd$
2	53	F_{12}	$\surd$
3	52	F_{10}	$\surd$
4	51	F_{99}	
5	50	F_{50}	$\surd$
6	50	F_{31}	
7	49	F_{13}	
8	48	F_{80}	
⋮	⋮	⋮	
16	26	F_{47}	
17	24	F_{93}	
18	22	F_{98}	$\surd$
19	18	F_{64}	
20	15	F_{26}	
21	15	F_{20}	
22	12	F_{40}	
23	12	F_2	$\surd$
⋮	⋮	⋮	

without the two main effects with the smallest coefficients. There are now ten terms in the true model and everything else remains the same. The true model contains 11 unique significant factors. The occurrence counts display nine significant factors but the remaining two significant factors do not appear.

The scenario in Table 43 uses the same true model as the previous scenario, but the input parameter $nTerms$ is reduced from 11 to eight to stop possible overfitting of the true model that has ten significant terms. The resulting rankings of occurrence counts are almost identical to those in the previous scenario. The top six factors in both scenarios are the same while other factors in the ranking are similar as well.

The final scenario in Table 44 again uses the same true model as the previous scenario 4E but without the two terms with the smallest coefficients. The number of terms in the true model has therefore reduced from 16 in scenario 4A to eight terms in the current scenario. The true model now also contains eight unique significant factors. Interestingly, all eight significant factors are recovered as the top eight factors in terms of occurrence counts. Furthermore, there is a sharp drop-off from the eighth factor in the ranking to the ninth factor. Therefore, this scenario correctly indicates what factors are significant in the true model and which ones are insignificant.

The difference between the first scenario and the final scenario is striking. The first scenario completely fails to recover the significant factors, while the final scenario does so very well. This is interesting because the main difference between these two scenarios is the number of terms in the true model. While the first scenario has 16 terms in the true model, the final scenario has eight terms.

Table 42. Robustness Scenario 4D - Systems Involving a Large Number of Terms (10 Terms, $nTerms = 11$)

True Model	
Coefficient	Term
1.68472	$INTERCEPT$
4.74096	$F_{50} = v_2$
4.61089	$F_{98} = v_1$
3.51114	$F_{22} = v_1$
3.06619	$F_{41} = v_1$
2.88918	$F_{68} = v_2$
2.70801	$F_2 = v_3$
2.02174	$F_{82} = v_1$
3.84451	$F_{74} = v_2$ & $F_1 = v_1$
2.65065	$F_{67} = v_3$ & $F_{12} = v_2$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 11$

Occurrence Counts			
Rank	Count	Factor	True
1	63	F_{22}	$\surd$
2	62	F_{41}	$\surd$
3	56	F_2	$\surd$
4	52	F_{50}	$\surd$
5	50	F_{98}	$\surd$
6	50	F_1	$\surd$
7	44	F_{97}	
8	41	F_{68}	$\surd$
9	35	F_{26}	
10	34	F_{56}	
11	33	F_8	
12	32	F_{31}	
13	30	F_{88}	
14	30	F_{29}	
15	28	F_{67}	$\surd$
16	23	F_{82}	$\surd$
17	16	F_{92}	
18	15	F_{38}	
19	13	F_{58}	
$\vdots$	$\vdots$	$\vdots$	

Table 43. Robustness Scenario 4E - Systems Involving a Large Number of Terms (10 Terms, $nTerms = 8$)

True Model	
Coefficient	Term
1.68472	$INTERCEPT$
4.74096	$F_{50} = v_2$
4.61089	$F_{98} = v_1$
3.51114	$F_{22} = v_1$
3.06619	$F_{41} = v_1$
2.88918	$F_{68} = v_2$
2.70801	$F_2 = v_3$
2.02174	$F_{82} = v_1$
3.84451	$F_{74} = v_2$ & $F_1 = v_1$
2.65065	$F_{67} = v_3$ & $F_{12} = v_2$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 8$

Occurrence Counts			
Rank	Count	Factor	True
1	78	F_{22}	$\surd$
2	66	F_{41}	$\surd$
3	56	F_2	$\surd$
4	51	F_{50}	$\surd$
5	50	F_{98}	$\surd$
6	46	F_1	$\surd$
7	35	F_{67}	$\surd$
8	25	F_{37}	
9	19	F_{68}	$\surd$
10	15	F_{97}	
11	15	F_{58}	
12	13	F_{31}	
13	11	F_{88}	
$\vdots$	$\vdots$	$\vdots$	

Table 44. Robustness Scenario 4F - Systems Involving a Large Number of Terms (8 Terms, $nTerms = 8$)

True Model	
Coefficient	Term
1.68472	$INTERCEPT$
4.74096	$F_{50} = v_2$
4.61089	$F_{98} = v_1$
3.51114	$F_{22} = v_1$
3.06619	$F_{41} = v_1$
2.88918	$F_{68} = v_2$
2.70801	$F_2 = v_3$
3.84451	$F_{74} = v_2$ & $F_1 = v_1$

Analysis Parameters		
$nModels = 50$	$nNewModels = 50$	$nTerms = 8$

Occurrence Counts			
Rank	Count	Factor	True
1	52	F_{41}	$\surd$
2	51	F_{68}	$\surd$
3	51	F_{50}	$\surd$
4	51	F_2	$\surd$
5	50	F_{98}	$\surd$
6	50	F_{22}	$\surd$
7	38	F_1	$\surd$
8	37	F_{74}	$\surd$
9	5	F_{12}	
$\vdots$	$\vdots$	$\vdots$	

5.5 Summary

We began this chapter with a discussion of the "heavy-hitters" requirement. The scenarios in Tables 17, 18, 19, 20, and 21 show how, without any noise, more heavily satisfying the requirement leads to more accurate results. However, the scenarios in Tables 22, 23, 24, and 25 show how, with noise in the system, more heavily satisfying the requirement causes the recovery to fail completely.

Next, the topic of separation in locating arrays is revisited. The scenarios in Tables

27, 28, 29, 30, 31, 32, 33, 34, and 35 show how higher separation helps recovery with noise in the system. However, the scenarios in Tables 36, 37, and 38 show that the benefit of separation has limits and it does not help recovery in some cases.

Finally, the scenarios in Tables 23, 24, and 25 show how noise can cause difficulty in recovering all significant terms. This is likely because the noise can overwhelm some significant factors with small coefficients. In Section 5.4, although no artificial noise is added, the many terms in the true model likely obscure each other's effects and block the recovery. Hence, when the number of terms in the true model is decreased in the final scenario, the analysis is able to recover all significant terms.

Chapter 6

CONCLUSION

This thesis began by introducing locating arrays and providing motivation for their use in the screening process. It showed how covering arrays are useful in determining the presence of a significant t-way interaction, but that coverage is not enough. Screening requires that significant t-way interactions be distinguished from each other so that the relevant system factors may be identified. Locating arrays are therefore a natural solution for the screening process. Next, this thesis discussed construction of locating arrays, analysis strategies, and robustness.

However, throughout this thesis, we only examined locating arrays with $d = 1$ because of a "heavy-hitters" assumption. However, one cannot be sure that the effects of all t-way interactions always follow this pattern. We therefore leave it for future work to examine locating arrays with $d \geq 1$ to cope with terms that do not conform to the "heavy-hitters" pattern.

We also assumed in this thesis that interactions of strength greater than 2 are not significant. But there are certainly situations where interactions of strength 3 or higher significantly affect the response. In this case, the work in this thesis may seek to estimate these effects with main effects and 2-way interactions likely leading to false screening results. We leave it for future work to examine locating arrays with interactions of strength greater than 2.

Following the introduction of locating arrays, this thesis discussed construction approaches for locating arrays. Building on related work, this work showed that locating arrays can be created using the Lovász Local Lemma with Moser-Tardos resampling, and that the construction process can be accelerated by using a scoring procedure for the array. Furthermore, Moser-Tardos resampling can be used to construct locating arrays that have additional constraints including separation δ.

Next, a new analysis approach for locating arrays was presented in Algorithm 4. We build on iterative SIS and OMP in related work to create an algorithm that is both useful and easy to implement. However, several aspects of our analysis algorithm should be investigated in future work. Our analysis approach in Algorithm 4 always keeps the same number of models, *nModels*, in its queues, and always explores the same number of alternatives, *nNewModels*. But there may be certain iterations where it is better to explore more options because of high uncertainty and other iterations where this is not required because of high certainty. Ultimately, while there may be benefits to keeping these as constant parameters to the algorithm, one might instead want to vary these parameters dynamically. Furthermore, one might also want the parameter $nTerms$ to be chosen automatically to correct possible overfitting issues with analysis. This might be done by setting a threshold for R^2, or the analysis might be updated to stop adding terms when R^2 does not improve significantly. Yet another possibility is to track the occurrence counts as more terms are added and stop adding terms based on the drop-off in the factor rankings. This work does not explore these possibilities but leaves them for future work.

One important aspect of locating arrays affecting analysis is their imbalance. Although all main effects and interactions (up to strength t) are covered in a $(\overline{d}, \overline{t})$-locating array, some terms are covered much more than others. Analysis strategies for locating arrays should not be biased towards terms that are covered more often than others. However, OMP uses a dot product that treats all tests equally and ignores the imbalance that is inherent to locating arrays. Suppose a locating array A contains hundreds of tests but a particular interaction T_1 is covered in exactly one test. If T_1 is relevant to the system under test, then it is likely that OMP, which treats all tests equally, will be biased towards terms that affect the response in several tests and T_1 will experience discrimination. There also exist limitations with using R^2 to measure the goodness of models. All tests are again treated equally when calculating R^2, and thus it is likely that the effects of T_1 are ignored with little change to R^2.

One possible solution might be to weight the tests of a locating array based on the percentage of coverage for a particular interaction that each test accounts for. Yet each test involves many interactions that are each covered differently leading to more challenges. To complicate matters even further, system noise affects all tests equally, and placing a higher weight on a particular test will exacerbate noise in that test. In [5], strategies are employed to specifically cope with imbalance in the locating array. Interestingly, this thesis produces nearly identical results for the same screening experiment in [5] even though we do not consider imbalance. It may be interesting to investigate why these approaches that are quite different produce the same results, and if there are other experiments where the approaches might produce different results. We leave this difficult issue of balance and unfair bias of tests to future work.

Algorithm 4 analyzes the locating array response and generates a list of $nModels$ best models. But screening must indicate what factors are significant to include in experimentation. This thesis determines the significant factors by counting the occurrences of each factor in the list of best models generated by the analysis algorithm. Although counting occurrences of factors might be simple, this approach may not be the best way to determine significance. For example, a factor appearing in a main effect with a large coefficient is likely more significant than a factor appearing in a main effect with a small coefficient. A factor is also likely more important when it appears in a main effect than when it appears in an interaction. And the significance of a factor likely also depends on the goodness of the model it appears in (the R^2 of the model). However, this thesis ignores coefficients and model goodness when counting factor occurrences. A factor appearing in an interaction is also counted exactly the same as when it appears in a main effect. However, we leave it for future work to examine and resolve these issues.

Finally, this thesis discussed robustness of our screening approach using locating arrays. We discussed negative effects that may occur when the "heavy-hitters" assumption is not satisfied. Scenarios were provided with systems satisfying the "heavy-hitters" assumption to varying degrees. Furthermore, artificial noise was introduced in several scenarios and the results were compared to the same scenarios but without noise. We showed how separation can help fight against the effects of noise. Scenarios were also given showing systems that included a large number of terms, and these terms obscuring each other's effects.

REFERENCES

[1] C. J. Colbourn and D. W. McClary, “Locating and detecting arrays for interaction faults,” *Journal of Combinatorial Optimization*, vol. 15, pp. 17–48, 2008.

[2] S. A. Seidel, K. Sarkar, C. J. Colbourn, and V. R. Syrotiuk, “Separating interaction effects using locating and detecting arrays,” in *Combinatorial Algorithms [IWOCA 2018], Lecture Notes in Computer Science*, C. Iliopoulos, H. W. Leong, and W.-K. Sung, Eds., vol. 10979, Springer International Publishing, 2018, pp. 349–360.

[3] D. C. Montgomery, *Design and Analysis of Experiments*, 9th. John Wiley and Sons, Inc., 2017.

[4] C. J. Colbourn and V. R. Syrotiuk, “On a combinatorial framework for fault characterization,” *Mathematics in Computer Science*, vol. 12, no. 4, pp. 429–451, Dec. 2018.

[5] A. N. Aldaco, C. J. Colbourn, and V. R. Syrotiuk, “Locating arrays: A new experimental design for screening complex engineered systems,” *SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 31–40, Jan. 2015.

[6] C. J. Colbourn, E. De Poorter, M. T. Mehari, I. Moerman, and V. R. Syrotiuk, “An efficient screening method for identifying parameters and interactions that impact wireless network performance,” *Submitted for publication*,

[7] X. Li, N. Sudarsanam, and D. D. Frey, “Regularities in data from factorial experiments,” *Complexity*, vol. 11, no. 5, pp. 32–45, 2006.

[8] S. A. Seidel, M. T. Mehari, C. J. Colbourn, E. De Poorter, I. Moorman, and V. R. Syrotiuk, “Analysis of large-scale experimental data from wireless networks,” in *IEEE INFOCOM International Workshop on Computer and Networking Experimental Research Using Testbeds (CNERT)*, 2018.

[9] S. K. Stein, “Two combinatorial covering theorems,” *Journal of Combinatorial Theory, Series A*, vol. 16, no. 3, pp. 391–397, 1974, ISSN: 0097-3165.

[10] L. Lovász, “On the ratio of optimal integral and fractional covers,” *Discrete Mathematics*, vol. 13, no. 4, pp. 383–390, 1975, ISSN: 0012-365X.

[11] D. S. Johnson, “Approximation algorithms for combinatorial problems,” *Journal of Computer and System Sciences*, vol. 9, no. 3, pp. 256–278, 1974.

[12] P. Erdős and L. Lovász, "Problems and results on 3-chromatic hypergraphs and some related questions," *Infinite and finite sets (Colloq., Keszthely, 1973 Vol. II, pp. 609–627. Colloq. Math. Soc. János Bolyai)*, vol. 10, pp. 609–627, 1975.

[13] K. Sarkar, "Covering arrays: Algorithms and asymptotics," PhD thesis, Arizona State University, 2016.

[14] R. A. Moser and G. Tardos, "A constructive proof of the general Lovász local lemma," *J. ACM*, vol. 57, no. 2, Art. 11, 15, 2010.

[15] J. Y. Liu, W. Zhong, and R. Z. Li, "A selective overview of feature screening for ultrahigh-dimensional data," *Science China Mathematics*, vol. 58, pp. 2033–2054, 2015.

[16] J. Fan and J. Lv, "Sure independence screening for ultrahigh dimensional feature space," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 70, no. 5, pp. 849–911, 2008.

[17] M. Fay and M. Proschan, "Wilcoxon-Mann-Whitney or t-test? on assumptions for hypothesis tests and multiple interpretations of decision rules," *Statistics Surveys*, vol. 4, pp. 1–39, 2010.

[18] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947.

[19] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, Dec. 1945.

[20] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, Dec. 1974.

[21] M. Mehari, A. Shahid, I. Moerman, and E. De Poorter, "Demo abstract: An intuitive drag and drop framework for wireless network experimentation," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '17, Delft, Netherlands: ACM, 2017. DOI: 10.1145/3131672.3136971.