COVERING ARRAYS

BROOKE LEFEVRE AND EMILY ANDERSON

ABSTRACT. Given their several applications, covering arrays have become a topic of significance over the last twenty years in both the mathematical and computer science fields. A covering array is a $N \times k$ array with strength $t$, $k$ rows of length $N$, entries from the set $\{0, 1, \ldots, v - 1\}$, and all $v^t$ possible combinations occur between any $t$ columns, where $N, k, t,$ and $v$ are positive integers. The focus of this research is to explore the different constructions of strength two and strength three covering arrays, to find better covering arrays (i.e. more cost and time efficient covering arrays), and to see if covering arrays can detect a fault in a system. Through analyzing the covering arrays that we constructed, we were able to successfully prove that in general, a covering array of strength $k + 1$ can detect a single fault between any $k$ or fewer variables in a system. Some areas of future research would include finding the location of a fault in a system or detecting two or more faults in a system.

1. Introduction

A covering array is a $N \times k$ array with strength $t$, $k$ rows of length $N$, entries from the set $\{0, 1, \ldots, v - 1\}$, and all $v^t$ possible combinations occur between any $t$ columns, where $N, k, t,$ and $v$ are positive integers. The set of covering arrays for specific values of $N, k, t,$ and $v$ is denoted by $CA(N; k, t, v)$. See Figure 1 for an example of an element of $CA(4; 3, 2, 2)$.

An important application of covering arrays is being used for both software and hardware testing. Computer programs and complex systems require inputs which then produce certain outputs. This is exactly what a covering array does. Each row in Figure 1 can be thought of as an input, or test, which then, depending on the parameters, either outputs a success or an error.

Testing can become quite expensive and time consuming as the number of combinations and configurations increases for large systems. For example, take a car to be a system with hundreds of different interactions happening all at once. You can have your air conditioning running, your radio blasting, and using your breaks all at once. In order to determine the number of tests we need to run in this system for all of these interactions to work, we want
to find $N$, the minimum number of rows (or tests) that a covering array must have. This is denoted as $CAN(k, t, v)$. Since we want to keep the number of tests low, $N$ is not always the number of $v^k$ combinations. Figure 2 provides a table that gives the minimum number of tests needed for known binary covering arrays.

*Qualitative independence* is when none of the sets $A \cap B$, $A^c \cap B$, $A \cap B^c$, and $A^c \cap B^c$ is empty, where $A$ and $B$ are subsets of $S$ and $S$ is a set having $n$ elements. This means that even though an element $x$ is in $A$, that doesn’t determine whether or not it is in $B$ (and vice-versa). [3] Therefore, qualitative independence and covering arrays are closely related. As a matter of fact, it is for this reason that covering arrays are used for statistical testing. Note that in Figure 1 the columns represent the different sets and the rows represent the variables within those sets.

Throughout the majority of our research, we focused on binary covering arrays, which are the most studied type of covering arrays. A *binary covering array* is a $N \times k$ covering array with $v = 2$, where $v$ can be 0 or 1, and $N, k, t$, and $v$ are positive integers. Figure 1 is also an example of a binary covering array.

$$CA(4; 3, 2, 2)$$

0 0 0
1 0 1
0 1 1
1 1 0

**Figure 1.** This is a $4 \times 3$ binary covering array of strength two ($t = 2$) that has four tests ($N = 4$) and three variables ($k = 3$). It is binary as it only contains the values 0 and 1 ($v = 2$).
robots do not have a common sense of right and left on the line. Each robot can move on the line until it meets another. One wishes to determine the minimum over all possible strategies of the maximum over all possible starting permutations of the robots, of the time by which they can arrive at the same point. Hartman shows that this value is ⌈k/2⌉ + ⌈\log_2 k⌉ + 1, by making use of bounds described in 3.2.1, below.

For given small values of \( s + t \) and \( t \), Table 1 records the smallest number of rows (\( n \), denoted by \( \text{CAN}(s + t, t) \)) in a binary covering array having those parameters, when this is known. When a range is given, the numbers represent lower and upper bounds on the smallest number of rows. This table is included to aid in the exposition.

Much more thoroughgoing tables may be found at the website of Colbourn [28], from which many of the upper bounds in the table were derived; and when not stated that the upper bounds for the intervals in the lower right corner of this table represent recent results and will probably change again soon!

The tables of [28] do not include actual covering arrays (and we have not independently verified in all cases that covering arrays of the indicated sizes exist). For an extensive collection of actual covering arrays with relatively few rows, visit the webpage [87], and follow the link to the covering array library. Also, covering arrays are available at the website of Nurmela [67], and covering arrays can be downloaded, by request, from the website maintained by Torres-Jimenez [81]. The website of Torres-Jimenez includes, in particular, covering arrays yielding all of the upper bounds in Table 1.

<table>
<thead>
<tr>
<th>( s \backslash t )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>21</td>
<td>42</td>
<td>85</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>24</td>
<td>48–52</td>
<td>96–108</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>24</td>
<td>48–54</td>
<td>96–116</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>24</td>
<td>48–56</td>
<td>96–118</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>24</td>
<td>48–64</td>
<td>96–128</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>24</td>
<td>48–64</td>
<td>96–128</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>6</td>
<td>12</td>
<td>24</td>
<td>48–64</td>
<td>96–128</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>7</td>
<td>15</td>
<td>30–32</td>
<td>60–64</td>
<td>120–128</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>7</td>
<td>15–16</td>
<td>30–35</td>
<td>60–79</td>
<td>120–179</td>
</tr>
</tbody>
</table>

Values of \( \text{CAN}(s + t, t) \).

Figure 2. The numbers on the inset of the table represent the minimum number of tests for known binary covering arrays, where \( s + t \) represents the number of columns and \( t \) represents the strength.

2. Constructions

Here we investigate two methods for constructing larger covering arrays from smaller covering arrays.

**Theorem 2.1** (Generalized Direct Product, Theorem 4.1, [2]). When a \( \text{CA}(N; k, 2, v) \) and a \( \text{CA}(M; l, 2, v) \) both exist, a \( \text{CA}(N + M, kl, 2, v) \) also exists.

**Proof.** Let \( A = (a_{ij}) \) be a \( \text{CA}(N; k, 2, v) \) and let \( B = (b_{ij}) \) be a \( \text{CA}(M; l, 2, v) \). Begin by placing \( l \) copies of \( A \) one after another. Below that, attach \( k \) copies of \( B \). We will call this a \( (N + M) \times kl \) array, \( C = (c_{ij}) \). So, either different columns of \( A \) or different columns of \( B \) produce two distinct columns of \( C \). Thus, \( A \otimes B \) covers all possible combinations of two columns. Thus, \( C \in \text{CA}(N + M; 2, kl, v) \). \( \square \)
\[ A = \text{CA}(4;3,2,2) \quad B = \text{CA}(5;4,2,2) \quad C = \text{CA}(9;12,2,2) \]

\[
\begin{array}{ccccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
\end{array}
\]

**Figure 3.** In this example, matrix \( A \) is written \( l \) times. Below that, matrix \( B \) is written \( k \) times. This construction then produces matrix \( C \).

**Theorem 2.2** (Roux Construction, Theorem 1, [1]). When a \( \text{CA}(N; k, 3, v) \), \( \text{CA}(N; k, 3, v) \), and \( \text{CA}(M; k, 2, v) \) exist, a \( \text{CA}(N + M; 2k, 3, v) \) also exists.

**Proof.** Let \( A = (a_{ij}) \) be a \( \text{CA}(N; k, 3, v) \), let \( B = (b_{ij}) \) be a \( \text{CA}(N; k, 3, v) \), and let \( C = (c_{ij}) \) be a \( \text{CA}(M; k, 2, v) \). Begin by placing the \( A = (a_{ij}) \) and \( B = (b_{ij}) \) side by side. By choosing any three columns with indices that are distinct modulo \( k \), all the combinations are covered. The remaining selection consists of a column \( x \) from among the first \( k \), its copy among the second \( k \), and a further column \( y \). In addition, by choosing two columns with indices that are the same modulo \( k \), another combination is also covered because they share the same value. By attaching two \( C \)'s side by side, the second being the complement of \( C \) (meaning all zeros become ones and vice versa), the remaining combinations are covered. Therefore, if we choose two distinct columns from one half, we choose the complement of one of these, which handles all the remaining combinations. Thus, \( D \in \text{CA}(N + M; 2k, 3, v) \). \( \square \)
\[
\begin{array}{cccccc}
A&=CA(8;3,3,2)\quad &B&=CA(8;3,3,2)\quad &C&=CA(4;3,2,2)\quad &D&=CA(12;6,3,2) \\
0&0&0&0&0&0 \\
0&1&0&0&1&0 \\
0&0&0&0&0&1 \\
1&1&0&1&1&0 \\
0&0&1&0&0&0 \\
1&1&0&1&1&1 \\
1&1&1&1&1&1 \\
1&0&0&1&1&0 \\
1&1&1&1&1&0 \\
1&0&1&1&1&1 \\
0&1&1&0&1&0 \\
0&1&1&1&0&0 \\
1&1&0&0&0&1 \\
\end{array}
\]

**Figure 4.** The resulting matrix \(D\) is produced as a result of a quad formation. It places \(A\) in the upper left corner, \(B\) in the upper right corner, \(C\) in the bottom left corner, and \(C\) complement in the bottom right corner.

### 3. Fault Detection

The goal of fault detection is to identify and characterize failures caused by specific combinations of option settings. Recall our car example from earlier. In order for your car to work properly, all of the interactions need to be seamless. Therefore, let’s say whenever you use your breaks your air conditioning stops working. This is an interaction fault. Now, you want to be able to locate the fault. That’s where covering arrays and fault detection come in to play. Covering arrays are a valuable tool as they allow one to detect and locate a fault. In our research, we were able to generalize this idea to covering arrays of strength \(k + 1\) with \(k\) variables.

**Theorem 3.1.** If \(C\) is a covering array of strength \(k + 1\), then we can detect a single fault between any \(k\) variables in the system. Note that if \(C\) is a strength \(k + 1\) covering array, it is also a strength \(k, k - 1, \ldots, 1\) covering array.
Proof. We show the result by contrapositive. Let $s$ be a subset of size $k$ of the columns of $C$, $p(s)$ be a set of values on these columns, and $r$ be a subset of rows of $C$. Define $f : s \times p(s) \rightarrow r$ to be the set of rows that contain the values $p(s)$ on the subset $s$.

Now, take $s_1, s_2$ to be distinct subsets of size $k$ of the columns of $C$ with values $p(s_1), p(s_2)$, respectively. We know that $s_1 \cup s_2$ must contain at least $k + 1$ elements. So, suppose $f(s_1, p(s_1)) = f(s_2, p(s_2))$. Let $r \in f(s_1, p(s_1))$, which means that column $s_1$ contains $p(s_1)$ and column $s_2$ contains $p(s_2)$. Let these contain distinct columns $c_1, c_2, \ldots, c_{k+1}$. In $r$, these values are fixed by our choice of $p(s_1), p(s_2)$. This means that $p(c_1), p(c_2), \ldots, p(c_{k+1})^{\ell}$, where $p(c_{k+1})^{\ell}$ represents a different value from the alphabet, does not appear in any row of $C$. Therefore, $C$ is not a covering array of strength $k + 1$, which contradicts our original assumption. So, $f$ is injective meaning that it has a distinct set of rows for the subset of size $k$ of the columns of $C$ and the set of values on these columns. Thus, we can conclude that if $C$ is a covering array of strength $k + 1$, then we can detect a single fault between any $k$ variables in the system. $\square$

Theorem 3.2. If $C$ is a binary covering array of strength 3, then we cannot detect two faults between any 2 variables in the system.

Proof. Let $s$ be a subset of size two of the columns of $C$, $p(s)$ be a set of values on these columns, and $r$ be a subset of rows of $C$. Now, let $r_1$ and $r_2$ be subsets of rows of $C$ corresponding to $\{0, 1, *, \ldots\}, \{0, 0, *, \ldots\}$ and $\{0, *, 0, \ldots\}, \{0, *, 1, \ldots\}$, respectively, where $*$ is a free choice. These subsets can be displayed in the following way:

\[
\begin{array}{cccc}
0 & 1 & * & \cdots \\
0 & 0 & * & \cdots \\
0 & * & * & \cdots \\
0 & * & 0 & \cdots \\
0 & * & 1 & \cdots \\
\end{array}
\]

First, take columns 1 and 2 and claim that the fault occurs between $\{0, 1\}$ and $\{0, 0\}$. This means that the rows in $r_1$ are an output for this specified fault. Even though the values in the second column are unknown in $r_2$, we do know that each value is either a 0 or a 1. This is true because it is a binary covering array. So, the values contained in columns 1 and 2 in the rows of $r_2$ are either $\{0, 0\}$ or $\{0, 1\}$. Thus, $r_1 \subseteq r_2$. Now, take columns 1 and 3...
and claim that the fault occurs between \{0,1\} and \{0,0\}. Through a similar process, we notice that \( r_2 \subseteq r_1 \). This means that \( r_1 = r_2 \). Therefore, the outputs are not distinct as they are subsets of one another and therefore have overlapping outputs. Thus, a binary covering array of strength three cannot detect two faults between any two variables in the system.

Note: By the properties of covering arrays, we can switch the bits (i.e. 0’s become 1’s and 1’s become 0’s) or switch the order of the rows and still get the same result.

□

4. Conclusion

Altogether, there are several different types of covering arrays depending on the parameters as well as a variety of ways to construct them. In addition, covering arrays play a significant role in both software and hardware testing as well as fault detection. In fact, we found that a covering array of strength \( k + 1 \) with \( k \) variables can detect a single fault in a system. Yet, there is still more to explore and discover in regards to covering arrays. One area for future research might be seeing if it is possible to use a covering array to detect two faults in a system. Other areas might include finding “better” small covering arrays and continuing to improve the construction of both strength two and strength three covering arrays. Nonetheless, given the versatility of covering arrays, it will continue to be topic of interest in both the computer science and mathematical field for years to come.
5. ACKNOWLEDGEMENTS

We would first like to thank the Valparaiso University Mathematics and Statistics Department and the MSEED Program (NSF Grant No. 1068346) for giving us the financial resources and support needed in order to conduct this research. More importantly, we would like to thank Professor Jon Beagley for both his time and guidance throughout summer.

REFERENCES


DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE, VALPARAISO UNIVERSITY, 1900 CHAPEL DRIVE, VALPARAISO, IN 46383

E-mail address: brooke.lefevre@valpo.edu, emily.anderson@valpo.edu