Efficient Exhaustive Search for Binary Complementary Code Sets

Gregory E. Coxson, Naval Research Lab
Jon C. Russo, Lockheed Martin Advanced Technology Labs

Abstract—Binary complementary code sets offer a possibility that single binary codes cannot - zero aperiodic autocorrelation sidelobe levels. These code sets can be viewed as the columns of so-called complementary code matrices, or CCMs. This matrix formulation is particularly useful in gaining the insight needed for developing an efficient exhaustive search for complementary code sets. An exhaustive search approach is described, designed to find all sets of $K$ complementary binary codes of length $N$, for specified $N$ and $K$. Results for several cases are examined.

Keywords: complementary code set, Hadamard matrix, autocorrelation sidelobes, binary code, exhaustive search.

I. INTRODUCTION

Technologies such as radar, sonar, ultrasound imaging and communications often use coded waveform techniques to transmit a signal and to process the return from a target or targets of interest. It is often the case that a weak signal needs to be detected despite the presence of noise and other interference. In such situations, a matched filter is called upon, because of its well-known ability to maximize signal-to-noise ratio [1][6]. For a signal pulse with binary phase coding, the response can be studied as the aperiodic autocorrelation of the binary code. The ideal structure of the autocorrelation is one having all zero elements except for a sharp peak at the center. This type of response characteristic allows the sensor to focus on the target of interest and to be less distracted by interfering objects and noise. Another benefit is that the problem is abstracted and becomes purely the challenge of finding the sequences of ones and minus ones (or possibly, zeros and ones) that provide ideal or near-ideal responses.

An unfortunate reality of binary phase coding is that it is impossible to achieve an ideal autocorrelation sequence. The best achievable with a binary codes is an autocorrelation with a sharp central peak and remaining elements, or “sidelobes”, having peak size 1; the codes that achieve this are the well-known Barker codes [1][6]. However, odd-length Barker codes exist only up to length 13 [12], and for even-length Barkers, only up to length 4, if the well-known “Barker Conjecture” holds true [13]. One reason why it is impossible to do better is that the size of the outermost sidelobes is 1, since it is the product of the size of two unit-magnitude code elements. Fortunately, there are ways around this problem, including mismatched filtering or the use of complementary code sets.

The focus of this paper is on complementary code sets. The key idea is to let go of the goal of using a single binary phase code to achieve an ideal autocorrelation, and instead to use several binary phase codes. If these several codes have autocorrelation sequences that add element-wise to give a sharp peak and all zero sidelobes, then these codes constitute a complementary code set.

This entails another mathematical and computational challenge, that of finding binary code sets having this complementary property. Additionally, there are practical implementation issues; an important one is sensitivity to the Doppler shift of moving targets [1].

One thought when first encountering complementary code sets for the first time is that finding binary complementary sets must be quite a computational challenge, since the lower-dimensional problem of finding near-ideal binary phase codes is already a computational challenge. However, this worry soon dissipates upon a check of the literature. One relatively easy way to compile binary complementary code sets is from the rows or the columns of binary Hadamard matrices, for which a number of construction methods exist [3][9][10][11]. Although binary Hadamard matrices exist only as square matrices of order 1, 2, or a multiple of 4 ([7],[3]), a richer source are the binary complementary code matrices (CCMs) which exist in rectangular forms with fewer restrictions on their dimensions. Complementary code matrices include Hadamard matrices as a special case [2]. The added generality of the complementary code matrices comes at a slight cost – in general, only the columns of CCMs give complementary code sets. Many of the Hadamard matrix constructions may be extended to general CCMs either directly or with at most modest changes in conditions that must be satisfied [2].

The aim of this paper is to show that in addition to simple construction methods, it is possible to conduct exhaustive searches for all complementary code sets of given dimension $N \times K$. As with the binary codes, the complexity grows quickly, but for smaller dimensions, it is possible to find large numbers of sets with reasonable effort. An algorithm is detailed, and some results are described for completed exhaustive searches for several square ($N = K$) cases of lower order, as well as several cases with $K = 2$ (corresponding to Golay binary complementary pairs of particular lengths).

II. DEVELOPMENT OF THE MATRIX FORMULATION

We begin by defining a binary code $x$ of length $N$ as a sequence of elements $x_i$ where $x_i \in \{-1, 1\}$ for $i = 1, \ldots, N$. It will be useful to think of $x$ as a column vector of length $N$:

$$x = [x_1, \ldots, x_N]^T.$$
The autocorrelation of a code \( x \) is defined as the sequence of length \( 2N - 1 \)
\[
\text{ACF}_x = x \ast \overline{x}
\]
where \( \ast \) represents acyclic convolution and \( \overline{x} \) means reversal of vector \( x \). The elements \( \text{ACF}_x(k) \) for \( k = 1 - N, \ldots, N - 1 \) may be written explicitly as sums of pairwise products of the elements of \( x \). Then
- For \( k = 1, \ldots, N - 1 \):
  \[
  \text{ACF}_x(k) = \sum_{i=1}^{N-k} x_i x_{i+k}.
  \]  \( (1) \)
- \( |\text{ACF}_x(N - 1)| = |x_1 \overline{x_N}| = 1 \).
- When \( k = 0 \), \( \text{ACF}_x(k) \) represents the peak of the autocorrelation, which equals \( x_1 x_1 + \ldots + x_N x_N = |x|^2 = N \).
- For \( k = 1 - N, \ldots, -1 \),
  \[
  \text{ACF}_x(-k) = \text{ACF}_x(k).
  \]
Consider the \( N \times N \) outer product of \( x \) and \( x^* \):
\[
Q_x = xx^T.
\]
Let \( k^{th} \) diagonal of \( Q_x \) means the main diagonal if \( k = 0 \), the \( k^{th} \) diagonal above the main diagonal for \( k = 1, \ldots, N - 1 \) and the \( |k|^{th} \) diagonal below the main diagonal for \( k = 1 - N, \ldots, -1 \).

**Lemma 2.1.** For the outer product \( Q_x = xx^T \), the sum of the \( k^{th} \) diagonal is equal to \( \text{ACF}_x(k) \) when \( 1 - N \leq k \leq N - 1 \).

**Proof of Lemma 2.1**

When \( 1 \leq k \leq N - 1 \), the sum of the \( k^{th} \) diagonal of \( Q_x \) is
\[
x_{1}x_{1+k} + \ldots + x_{N-k}x_{N} = \sum_{i=1}^{N-k} x_i x_{i+k},
\]
or \( \text{ACF}_x(k) \).
When \( 1 - N \leq k \leq -1 \), the sum of the \( k^{th} \) diagonal, or the \( |k|^{th} \) diagonal below the main diagonal, is
\[
x_{1-k}x_{1} + \ldots + x_{N}x_{N-k} = \sum_{i=1}^{N+k} x_i x_{i+k},
\]
or \( \text{ACF}_x(-k) = \text{ACF}_x(k) \).
When \( k = 0 \), the sum of the \( 0^{th} \), or main, diagonal is
\[
x_1 x_1 + \ldots + x_N x_N = N
\]
or \( \text{ACF}_x(0) \).

Now consider a set \( \{x_1, \ldots, x_K\} \) where \( x_j \) is a binary code of length \( N \) for \( j = 1, \ldots, K \).

**Definition.** The set \( \{x_1, \ldots, x_K\} \) is a **complementary code set** if
\[
\sum_{j=1}^{K} \text{ACF}_{x_j}(k) = 0
\]
for \( k = 1, \ldots, N - 1 \) and \( k = 1 - N, \ldots, -1 \).

We need the following definitions before making a key observation.

**Definition.** An \( N \times K \) matrix \( Q \) is a **complementary code matrix**, or CCM, if its columns form a complementary code set.

Note that only the columns, not the rows, form a complementary set. For the special case of Hadamard matrices, both the rows and the columns form complementary sets.

**Definition.** An \( N \times N \) matrix \( M \) is **diagonally regular** if the elements of each diagonal, except for the main diagonal, sum to zero.

**Definition.** Given a real \( N \times K \) matrix \( Q \), its row **Gramian** is the \( N \times N \) matrix \( QQ^T \). The \((i,j)\) element of the row Gramian of \( Q \) for any \( 1 \leq i, j \leq N \) is the inner product of row \( i \) and row \( j \) of \( Q \).

The key observation is given in the following theorem.

**Theorem 2.2.** A binary \( N \times K \) matrix \( Q \) is a CCM if and only if \( QQ^T \) is diagonally regular.

**Proof of Theorem 2.2**

Suppose that an \( N \times K \) binary matrix \( Q \) is given. Let \( x_j \) represent its \( j^{th} \) column vector. Then
\[
QQ^T = \sum_{j=1}^{K} x_j x_j^T.
\]
If \( QQ^T \) is diagonally regular, then the sum of the \( k^{th} \) diagonal of \( \sum_{j=1}^{K} x_j x_j^T \) is zero for \( k = 1 - N, \ldots, -1 \) and \( k = 1, \ldots, N - 1 \). Then by Lemma 2.1,
\[
\sum_{j=1}^{K} \text{ACF}_{x_j}(k) = 0
\]
for \( k = 1 - N, \ldots, -1 \) and \( k = 1, \ldots, N - 1 \). It follows that \( \{x_1, \ldots, x_K\} \) is a binary complementary code set. Clearly this argument can be made in reverse, establishing the theorem. □

**III. TOWARD EXHAUSTIVE SEARCH FOR COMPLEMENTARY CODE SETS**

The CCM property given in Theorem 2.2 means that every diagonal of the row Gramian, excepting the main diagonal, sums to zero. This may be viewed in a way that provides some useful insights for developing a search algorithm. Suppose \( Q \) is an \( N \times K \) matrix, and let \( Q_j \) represent the \( j^{th} \) row of \( Q \), for \( j = 1, \ldots, N \). Then for \( Q \) to have the CCM property means that
\[
\sum_{j=1}^{N-k} Q_j Q_{j+k}^T
\]
for \( k = 1, \ldots, N - 1 \). Comparing equations (1) and (2) suggests that the CCM property is much like that of enforcing low sidelobe level in the autocorrelation sequence of a single binary code. Products of pairs of elements in the autocorrelation—sidelobe formulae have been replaced by sums of inner products of pairs of matrix rows corresponding to the diagonal sums of the row Gramian.
One of the properties of the sidelobe expressions given in equation (1) is that the outermost sidelobes of the autocorrelation \(ACF_x(k)\) depend only on outermost elements of the code \(x\). This suggests a branch-and-bound approach to searching for low-sidelobe codes. In this approach, the code is built from the outside in by a depth-first scheme. At each step, the autocorrelation sidelobes are updated, and if any fully computed sidelobe has size higher than a set threshold, then the search backs up to the most recent branch point. The search may also back up if a partially computed sidelobe is so large that it cannot meet the threshold when the sidelobe calculation completes. The branch-and-bound approach was used to good effect for exhaustive searches for low- or optimal-peak-sidelobe binary codes by Cohen, Fox and Baden in [5], and by Coxson and Russo in [8].

The branch-and-bound approach avoids unnecessary searches of numerous sizable subtrees. This can be critical to achieving a fully exhaustive search. Additional speed-ups can be achieved by incorporating other efficiencies into the branch-and-bound approach. Examples are the optimization of low-level computations using bit-level computations or table lookup [4], or taking into account sidelobe-preserving operations to avoid checking sidelobe-equivalent codes [4],[8].

With the binary codes, the optimal peak sidelobe is often unknown ahead of time; hence, the best threshold to use for target sidelobe level often has to be adjusted during the search process. In the case of CCMs, the target is known ahead of time to be 0, making it easier to design a strategy for the “bounding” part of the branch and bound.

A search algorithm for CCMs needs to take account of, and advantage of, the operations that preserve the CCM property. Because one CCM may be transformed into many others that are essentially the same, the list of CCMs provided by the algorithm can be unnecessarily long unless the routine is designed to avoid these redundancies. On the other hand, the possibility exists of significant speed-up unless the routine is designed to avoid these symmetries. The following theorem details these CCM-preserving operations. A proof can be found in [2], where the theorem is given in the more general context of unimodular CCMs (in which elements are restricted to the unit circle rather than the set \([-1, 1]\)).

**Theorem 3.1.** Suppose that \(Q\) is an \(N \times K\) binary CCM. Then the following operations preserve the CCM property and dimensions of the matrix:

(i) Replace any column \(x\) of \(Q\) with \(-x\).
(ii) Replace any column \(x\) of \(Q\) with its reversal \(\overline{x}\).
(iii) Form the product \(A_N Q\), where \(A_N\) is the \(N \times N\) diagonal matrix

\[
A_N = \begin{bmatrix}
-1 & 0 & \ldots & 0 \\
0 & 1 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & (-1)^N 
\end{bmatrix}.
\]

(iv) Form \(QP\) for any \(K \times K\) permutation matrix (for a definition, see [14]).

**IV. The Exhaustive Search Algorithm**

As described before, the search progresses from the outermost code bits to the inner-most, enforcing that the sum of the inner products of the current top rows with those of the corresponding bottom rows is zero. The search starts out with the top bits of each code all set to \(-1\). Holding this first row to all \(-1\)s eliminates searching through redundant codes which are merely negations of the original code. The search then proceeds through all bottom code bits which would result in an inner product of zero with the top row. Note that for the starting top row we have frozen, only bottom rows that are balanced (that is, have an equal number of \(1\)s and \(-1\)s) satisfy the zero inner product criterion, and one possibility is shown below, corresponding to the code set in our previous example. Let \(r_1, r_2, \ldots, r_N\) denote the rows of the \(N \times K\) matrix \(Q\). For a \(10 \times 10\) case, one solution would be:

\[
\begin{align*}
& r_1 = (-1)[1 1 1 1 1 1 1 1 1 1] \\
& \quad \ldots \\
& r_{10} = [-1 -1 -1 1 1 1 1 1 -1 -1]
\end{align*}
\]

The search investigates every bottom row that has an element sum of zero, satisfying \(r_N^T P = 0\). For a given bottom row the search progresses inward to evaluate possible choices for the second and the \((N-1)\)-th row. All possibilities are considered for the second row. For each second row possibility, the \((N-1)\)-th row is chosen such that \((r_2 r_N^T) + (r_1 r_{N-1}^T) = 0\). Note that since \(r_1\) is all \(-1\)s, this is equivalent to \(r_2 r_N^T - \text{sum}(r_{N-1}) = 0\), or \(\text{sum}(r_{N-1}) = (r_2 r_N^T)\), where “sum” denotes the vector sum of elements. So we only have to search over possible values for \(r_{N-1}\) that have this particular sum. For the \(10 \times 10\) example begun above, one possibility would be:

\[
\begin{align*}
& r_1 = (-1)[1 1 1 1 1 1 1 1 1 1] \\
& r_2 = [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
\end{align*}
\]

This process is repeated inward; at the next stage all possible values for \(r_{N-2}\) are considered that satisfy \(\text{sum}(r_{N-2}) = (r_3 r_N^T) + (r_2 r_{N-1}^T)\). At this level and beyond, we can check whether the absolute value of the partial autocorrelation sum is greater than \(N\), and terminate this branch if so, because no single row could sum to compensate this. When the matrix is fully populated, the remaining autocorrelation offsets (those corresponding to more than fifty percent overlap) are checked. If this final test passes, the result is stored; if not, the matrix is discarded and the search continues at the most recent branch.

Several steps are taken to ensure that only a single representative of each equivalence class is checked. One CCM symmetry has already been accounted for by setting the first row constant. Another, the combination of alternating sign and reversal operations, is handled by setting the first element of the second row to 0 (The matrix is represented in a binary bit form in which 0 represents \(-1\), and 1 represents 1.). Furthermore, as the matrix is built step-by-step, two
separate lexicographical orderings are enforced to account for other symmetries, and in the process, eliminate searching large, redundant portions of the search space. The first of the lexicographical orderings involves restricting the binary value of the populated bits of any given column to be less than or equal to its bit-reversed form. This eliminates searching over flipped versions of a given code, for a reduction of $2^K$ in the size of the search space. For our $10 \times 10$ example the factor by which the search space is reduced in size would be 1024. The second lexicographical ordering involves restricting the binary value of each column to be greater than or equal to that of its left neighbor (when it has one). This handles equivalence due to column permutations, reducing the search space further by $K!$. For the modest $10 \times 10$ example, this factor is 10!

V. RESULTS FOR SEVERAL TEST CASES

Because of the computational nature of the search task, the routine was coded in C. Searches were run on one core of a dual-core 2.66 GHz Intel processor.

The routine was tested first for the $6 \times 6$ case (sets of $K = 6$ complementary codes of length $N = 6$), for which the search space has size $2^{36}$, or about 68.7 billion sets. This is a case for which the CCM is square but not a binary Hadamard matrix (since 6 is greater than 2 and not divisible by 4 [3]). The number of matrices checked was 1251, and the number of CCMs found was 42. The sets are listed in the following table, each by a set of 6 decimal integers, each representing a separate binary sequence of length 6 (for an illustration of mapping from decimal integer to binary string, refer to the $10 \times 10$ example at the end of this section):

<table>
<thead>
<tr>
<th>$k$</th>
<th>$x_{k1}$</th>
<th>$x_{k2}$</th>
<th>$x_{k3}$</th>
<th>$x_{k4}$</th>
<th>$x_{k5}$</th>
<th>$x_{k6}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
<td>10</td>
<td>1</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>14</td>
<td>5</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>10</td>
<td>14</td>
<td>1</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>5</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>2</td>
<td>10</td>
<td>5</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>2</td>
<td>10</td>
<td>13</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>2</td>
<td>14</td>
<td>5</td>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>2</td>
<td>14</td>
<td>1</td>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>4</td>
<td>10</td>
<td>13</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>4</td>
<td>14</td>
<td>5</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>6</td>
<td>18</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>14</td>
<td>22</td>
<td>1</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>6</td>
<td>30</td>
<td>5</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>16</td>
<td>12</td>
<td>2</td>
<td>30</td>
<td>5</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>17</td>
<td>12</td>
<td>10</td>
<td>30</td>
<td>1</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>12</td>
<td>18</td>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>19</td>
<td>4</td>
<td>4</td>
<td>22</td>
<td>5</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>6</td>
<td>10</td>
<td>9</td>
<td>3</td>
<td>17</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td>2</td>
<td>10</td>
<td>13</td>
<td>7</td>
<td>25</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>2</td>
<td>14</td>
<td>5</td>
<td>11</td>
<td>25</td>
</tr>
</tbody>
</table>

The algorithm was next tried on the task of finding all distinct $8 \times 8$ CCMs. For this case, the size of the search space is $2^{64}$, or about $1.6 \times 10^{19}$. The algorithm checked 13427561 matrices and found 111025 CCMs. The running time was 1.328 seconds. Since $N = K = 8$, and 4 divides 8, the algorithm is expected to find CCMs equivalent to binary Hadamard matrices. According to Sloane’s Online Encyclopedia of Integer Sequences [17], there is a single equivalence class of $8 \times 8$ binary Hadamard matrices. The 111025 CCMs were checked, and the 71419th CCM returned was found to be Hadamard-equivalent (in particular, the CCM could be transformed into a binary Hadamard matrix by reversing, or “flipping”, several of the columns).

The algorithm was then applied to the task of finding the set of $10 \times 10$ CCMs, for which the search space is $2^{100}$, or about $10^{30}$. The search was discontinued, but not because of the running time; the search generated CCMs so rapidly that it quickly reached the limits of the available memory (several gigabytes).

The algorithm was also tried on several cases of the Golay complementary pairs, corresponding to $K = 2$. Golay pairs are named for Marcel Golay, who applied pairs of binary complementary codes to an application in infrared spectrometry [16]. Borwein and Ferguson have tallied the number of distinct Golay pairs for lengths up to 100 using number-theoretic methods combined with exhaustive search; their results provide a useful performance check for these lower lengths [15]. Our algorithm was tried for Golay-pair lengths 8, 10, 16, 20, and 32 and 34. For 8, 16 and 32, the lengths that can be represented as powers of 2, the number of CCMs returned by the algorithm were, respectively, 5, 36 and 336, which match the tallies in Borwein and Ferguson (see the table on page 984 of [15]). For length 34, the algorithm returned no CCMs, again agreeing with Borwein and Ferguson.

For the remaining lengths that were tried (10 and 20), the
number of CCMs returned by the algorithm did not match the corresponding tallies in Borwein and Ferguson tally, but instead were slightly higher. This difference led to examination of the pairs returned, and to the discovery of a symmetry that occurs for some Golay pairs that is not yet accounted for by the algorithm. In fact, this symmetry has proven to be difficult to deal with in the algorithm proper. Instead, it appears that the best way to account for this symmetry is by a post-processing step to eliminate redundancies in the sets returned by the algorithm. This symmetry is discussed in more detail in the next two paragraphs.

In the case where \( N = 10 \) and \( K = 2 \), corresponding to length-10 Golay pairs, the algorithm returned three complementary pairs. According to Borwein and Ferguson, there are two distinct pairs at length 10. The pairs returned, represented by pairs of integers in decimal, are \([131, 172], [22, 197]\) and \([38, 245]\). Translating these into pairs of binary codes yields the three Golay complementary pairs

\[
\begin{align*}
&x_{11} = [-1, -1, 1, -1, -1, -1, 1, 1] \\
&x_{12} = [-1, -1, 1, -1, 1, 1, -1, -1]
\end{align*}
\]

\[
\begin{align*}
&x_{21} = [-1, -1, -1, -1, -1, 1, 1, -1] \\
&x_{22} = [-1, -1, 1, 1, -1, -1, 1, -1]
\end{align*}
\]

and

\[
\begin{align*}
&x_{31} = [-1, -1, -1, 1, -1, -1, 1, 1] \\
&x_{32} = [-1, -1, 1, 1, 1, -1, 1, -1]
\end{align*}
\]

Recall notation introduced earlier:

- \( \pi \) means reversal of vector \( x \)
- \( A_N \) is the \( N \times N \) diagonal matrix

\[
A_N = \begin{bmatrix}
-1 & 0 & \cdots & 0 \\
0 & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & (-1)^N
\end{bmatrix}.
\]

Note that when \( A_N \) operates on an \( N \)-length binary code \( x \) on the right (that is, \( xA_N \)), it switches the sign of every other element, beginning with the first element. Then, observe that the following relationships hold between the second and third code Golay pairs returned for the \( 10 \times 2 \) case:

\[
\begin{align*}
x_{32} &= -x_{21}A_{10} \\
x_{31} &= x_{22}A_{10}
\end{align*}
\]

Once this symmetry is recognized, the yield from the algorithm can be viewed as two “distinct” sets, achieving agreement with the tally by Borwein and Ferguson for the \( 10 \times 2 \) case.

For the \( 20 \times 2 \) case, the algorithm returned 28 CCMs, while the Borwein and Ferguson tally for this case is 25. Examining the sets, it was found that six of the pairs can be grouped into three pairs of Golay pairs related by the symmetry observed in the \( 10 \times 2 \) case. The remaining twenty-two Golay pairs were found to be “self-equivalent”, in the sense that if one of these sets is represented by \( \{x_{11}, x_{12}\} \), then either

\[
\begin{align*}
x_{12} &= -x_{11}A_{20} \\
x_{11} &= x_{12}A_{20}
\end{align*}
\]

or

\[
\begin{align*}
x_{12} &= -x_{11}A_{20} \\
x_{11} &= x_{12}A_{20}
\end{align*}
\]

Finally, consider the following illustration using sets returned by the algorithm for the \( 10 \times 10 \) run (before it was discontinued for insufficient allocated memory). This is intended to show how integer sequences returned by the algorithm are translated into binary codes, and to establish the complementarity of a represented set resulting from the search.

Example \( 10 \times 10 \) code sets returned by the algorithm are listed below:

\[
\begin{align*}
&\begin{bmatrix} 12 & 28 & 44 & 45 & 45 & 69 & 101 & 103 & 306 & 342 \\
12 & 28 & 44 & 45 & 45 & 85 & 103 & 103 & 274 & 358 \\
12 & 28 & 44 & 61 & 69 & 101 & 101 & 111 & 298 & 358 \\
44 & 60 & 68 & 69 & 101 & 109 & 111 & 266 & 326 & 360
\end{bmatrix}
\end{align*}
\]

Each row is a list of 10 decimal numbers whose binary equivalents may be translated into a sequence of ones and minus ones using the mapping \( \{1, 0\} \rightarrow \{1, -1\} \).

For example, consider another set returned by the algorithm:

\[
\begin{bmatrix} 28 & 44 & 68 & 77 & 109 & 125 & 181 & 239 & 326 & 430
\end{bmatrix}
\]

The successive elements may be converted to binary strings of zeros and ones, giving the following sequence of 10 binary codes:

\[
\begin{align*}
&1) \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \\
&2) \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \\
&3) \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \\
&4) \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \\
&5) \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \\
&6) \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \\
&7) \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \\
&8) \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \\
&9) \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \\
&10) \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \\
\end{align*}
\]

Applying the mapping results in the set of binary \( \pm 1 \) codes below:

\[
\begin{align*}
&1) \quad -1 \quad -1 \quad -1 \quad -1 \quad -1 \quad 1 \quad 1 \quad -1 \\
&2) \quad -1 \quad -1 \quad -1 \quad -1 \quad 1 \quad 1 \quad 1 \quad -1 \\
&3) \quad -1 \quad -1 \quad -1 \quad 1 \quad -1 \quad -1 \quad 1 \quad -1 \\
&4) \quad -1 \quad -1 \quad -1 \quad 1 \quad -1 \quad 1 \quad 1 \quad -1 \\
&5) \quad -1 \quad -1 \quad -1 \quad 1 \quad 1 \quad 1 \quad 1 \quad -1 \\
&6) \quad -1 \quad -1 \quad 1 \quad -1 \quad 1 \quad 1 \quad 1 \quad -1 \\
&7) \quad -1 \quad -1 \quad 1 \quad -1 \quad 1 \quad -1 \quad 1 \quad -1 \\
&8) \quad -1 \quad -1 \quad 1 \quad 1 \quad -1 \quad -1 \quad 1 \quad -1 \\
&9) \quad -1 \quad -1 \quad -1 \quad 1 \quad -1 \quad -1 \quad 1 \quad -1 \\
&10) \quad -1 \quad -1 \quad -1 \quad -1 \quad -1 \quad 1 \quad 1 \quad -1 \\
\end{align*}
\]

Computing the autocorrelation sequences for each code yields the following, representing the left-hand sidelobes and the peak (since the autocorrelation sequence is symmetric, no information is added by including the right-hand sidelobes):

\[
\begin{align*}
&1) \quad [1, 2, 1, 0, -1, -2, -3, 0, 5, 10] \\
&2) \quad [1, 2, 1, 0, -1, -2, 1, 0, 1, 10] \\
&3) \quad [1, 2, 1, 0, 1, 6, 1, 0, 1, 10] \\
&4) \quad [-1, 0, -1, 0, -3, 2, 3, -2, -1, 10]
\end{align*}
\]
6) $\begin{bmatrix} 1 & 0 & -1 & 0 & -1 & -2 & 3 & -2 & -1 & 10 \\
-1 & 0 & -1 & 0 & -3 & -2 & -1 & 2 & 3 & 10 \\
-1 & 0 & 1 & -2 & 3 & -4 & 1 & 2 & -5 & 10 \\
-1 & -2 & -1 & 0 & 3 & 2 & -1 & 0 & 3 & 10 \\
1 & -2 & 1 & 0 & 1 & 2 & -3 & 0 & -3 & 10 \\
1 & -2 & -1 & 2 & 1 & 0 & -1 & 0 & -3 & 10 \end{bmatrix}$

Finally, the autocorrelation sequences are summed element-by-element to give the result:

$$[0 0 0 0 0 0 0 0 0 100].$$

As expected, the peak sum is $(N)(K) = 10^2 = 100$ and the sidelobe sums all came to 0, so the set is indeed complementary.

VI. CONCLUSIONS

A method is introduced for efficient exhaustive search of all complementary sets of $K$ binary codes of length $N$, building upon a formulation involving “complementary code matrices”, or CCMs. Similarities between the “diagonal regularity” property of row Gramians of CCMs and the structure of autocorrelation sidelobes results in the use of a branch-and-bound approach proven effective for exhaustive searches for low-peak-sidelobe binary codes. Efficiencies are built into the algorithm design to account for CCM symmetries. The performance of the algorithm is described for several square cases ($6 \times 6$, $8 \times 8$, $10 \times 10$), as well as several cases of Golay complementary pairs ($8 \times 2$, $10 \times 2$, $16 \times 2$, $20 \times 2$, $32 \times 2$ and $34 \times 2$).

VII. REFERENCES


