Costas Array Search Technique that Maximizes Backtrack and Symmetry Exploitation

Jon C. Russo, Keith G. Erickson, *Member, IEEE*, and James K. Beard, *Life Senior Member, IEEE*

***Abstract*—Two innovations in search methodology for Costas arrays are presented here: extensive exploitation of symmetries, and look-ahead row index exclusion tables. Together, they achieve a reduction of more than a factor of four in computational requirements over conventional search methods. We examined the benefits of these innovations on Costas arrays of higher orders, and particularly on a search over order 28.**

***Index Terms*—Permutation matrices, tree searching, Costas arrays**

I. Introduction

COSTAS arrays are permutation arrays that have the additional condition – the Costas condition – that no two ones in the matrix differ in position by the same number of both rows and columns. When Costas arrays are used as frequency assignment schemes in frequency-agile waveforms, then, for any range or Doppler offset other than zero, the receiver response exhibits energy from cross-correlation of no more than one pair of pulses [1][2].

Finding Costas arrays is limited to either of two fortuitous number-theoretic generators and their extensions [3][4][5] or exhaustive search. The generators do not find all Costas arrays of a given order above about order seven [6], and Costas arrays have been found for orders as high as 27 that are not found or predicted by any of the generators [9][10]. The only known way to find all Costas arrays of orders greater than about seven is exhaustive search.

Since there are *N* factorial permutation matrices of order *N* and there is no known way of restricting the space in which Costas arrays may exist, the search itself may be of factorial complexity. The technique of backtrack programming [6][7] reduces the search complexity. We observe a factor of about five in complexity for each increase in order using backtrack programming.

Over time, the authors have developed an exhaustive search methodology that has provided the first exhaustive search over orders 24, 25, and 26 [8], and was the first to find an announced new Costas array of order 27 [9][10]. We present here two major innovations in this methodology not previously reported that provide a search with less than one-fourth the resources required by conventional search methods.

Manuscript received January 6, 2001.

J. C. Russo is with Lockheed Martin Advanced Technology Laboratories, Cherry Hill, NJ 08002 USA (e-mail: jon.c.russo@lmco.com).

K. G. Erickson is an independent consultant, Cherry Hill, NJ 08002 USA (e-mail: keith.george.erickson@gmail.com).

J. K. Beard is an independent consultant, Medford, NJ 08055 (609-654-6559, fax 609-654-8751, e-mail: jkbeard1@verizon.net).

II. The Base Search Methodology

A. Row Index Notation, the Difference Table, and the Costas Condition

Here we express permutation matrices by row index notation, in which a sequence of N integers represents the row indices of the ones in successive columns. From such a representation, we construct a difference matrix: row *i* and column *j* is the difference between the row indices for columns *j* and *j*+*i*. An entry in the difference matrix is, given a shift of *i* columns, the number of rows to shift to overlap the dots for that column. Thus if there are no duplicated differences on any given row, then no two ones differ in position by the same number of rows and columns, thus the Costas condition is met.

B. An Example of the Difference Table

We build the difference table by beginning with the Costas array in row index form. For the second row, we subtract the row indices for each column from the row index of the preceding column. We label this row D1 for differences in row indices for columns that are adjacent. For the next row of the difference table, which we label D2, we enter the differences between the row indices for columns with indices that differ by two. The row of the difference matrix for *k* columns apart has *n-k* entries, so we are done with n-1 rows. The difference table for the Costas array {4,2,5,1,3} is shown below as Table 1.

Table 1. Difference Table for {4,2,5,1,3}

Col	1	2	3	4	5
CA	4	2	5	1	3
D1	-2	3	-4	2	
D2	1	-1	-2		
D3	-3	1			
D4	-1				

By using the difference table, verification that a given permutation matrix satisfies the Costas condition can be accomplished in polynomial time. Use of the difference matrix to preclude row indices in the process of the search is backtrack programming with preclusion [7], which underlies all known practical search mythologies to date [11]. We begin here the most basic form of this process as a elementary foundation for our method.

C. *Logic and Data Flow of the Elementary Search Method*

Here we describe one method of applying backtrack programming with preclusion to be used as the core engine for a Costas array search methodology. We begin by tying the difference matrix to the Costas condition.

We build a Costas array one row index at a time by using the difference matrix to determine which row indices are precluded for the next column, either because that row index is already taken or because its use would cause a duplication in a row of the difference matrix as available to that point. Only row indices that are not precluded are accepted by the search algorithm, which then uses the accepted row index to update the difference matrix. Thus, the method builds the difference matrix simultaneously with the set of row indices by using the preclusion matrix computed from the difference matrix to ensure that each succeeding choice of column index meets the Costas condition.

We implement the preclusion matrix as follows. We begin by precluding values already used as row indices. Entries in row D1 in Table 1 represent the differences between adjacent columns. We add these entries to the row index for the column preceding the yet-undetermined row index for the next column and the sums become forbidden row indices for the next column. We add entries in row D2 in Table 1 to the row index for the column two preceding the yet-undetermined column to obtain more precluded row indices, etc. The result is the difference-preclusion table.

1) *Illustration of the Method*

Please refer to Table 1. We begin with the row index of the first column of four from our example of {4,2,5,1,3}. The row index of the second column can be any row index except four, and following our example, we select two. The difference matrix at this point consists of the title column of Table 1, columns one and two of the CA row and column one of row D1, or all except the last four columns of each difference row.

For any number of available row indices, we build up a list of row indices precluded for the next column as the union of two sets: (1) the row indices already used, and (2) precluding 0 as the next row index to avoid repeating the row index difference of -2. We only allow row indices of 1 through 5 in any case.

As Table 2 shows, we preclude row indices either because they are already used by preceding columns, or because they would result in a repeated entry in the difference table row D1, the only row that is populated to this point. We use the notation CA(j) for entry j in row CA, or the row index for column j, and D1(k) for entry k in row D1. Note that one of the precluded row indices, the lone entry in the difference preclusion table, is outside the permissible range of columns, so we note it in Table 2, but it does not affect the choice of row index for the next column. In general, a difference in row indices can be any integer from –(n-1) to +(n-1). We see from Table 2 that the row indices one, three and five are available. In keeping with our selected example of {4,2,5,1,3} we select five.

The difference matrix at this point consists of the first three columns of row CA, the first two columns of row D1, and the first column of row D2, or, all of Table 1 except the last three columns of each difference row. We use the available values of the difference table and row indices to form a list of precluded row indices for column four. In general, we begin with the row indices used in columns one through three, and these column indices are marked as used. Then we add the D1 row of the difference table to the row index of the column one back and mark those sums as forbidden for the next column index. We add the D2 row to the row index of the row index two columns back, the row three indices to the row index from three columns back, continuing until we have no more available difference table rows. Table 3 below shows the result for the row indices for the first three columns.

At this point, the difference matrix consists of Table 1 except the last two columns of each difference row. The last row index is the only row not yet taken, if that row index meets the Costas condition. The Costas condition is linked to the difference matrix, so we repeat the process of adding the available entries of the row labeled Dx to the Costas array row index for x columns back from the next column. The result in this case is shown as Table 4. Note that row index three is available in Table 4 so we have a Costas array.

D. *Summary of Backtrack Programming with Preclusion in Search*

We have defined a method of backtrack programming with preclusion for a Costas array search methodology. We perform the search in recursion levels, the current level number being the number of columns for which row indices have been defined to a given point in the search.

We initialize the search at level one by defining the row index for the first column; these row indices are stepped from one to N (or, zero to N-1). When a new row index is not precluded, it is accepted and the search moves to the next level. Then the search over the available row indices is completed, the search moves back to the previous level. When we reach level N and a row index is not precluded, we have a Costas array and we declare it and drop back to level N-1. The entire search terminates when the row index search for level one is completed. Thus, the search methodology flow is a continuing change of recursion level.

E. *Use of Bitmasks for Rows of the Difference Table and Preclusion Mask*

The difference table consists of numbers that may be anywhere from –(N-1) to (N-1). Since the Costas condition is that no entry is ever allowed to repeat, then bit positions in a 64-bit register can be used to represent a row of a difference matrix for searches up to order 33.

The preclusion table is shown for all values that may be defined in the algorithm in the examples. However, only row indices from zero to N-1 are significant in the implementation, so a 32-bit mask is sufficient. This bitmask is initialized with the rows already used up to that point in the search, and the difference table is shifted by the amount of row indices and a logical OR updates the necessary portion of the preclusion mask.

Table 2. Precluded row indices for the third column from two row indices

Row Index	Reason
0	D1(1)+CA(2)
1	
2	Taken
3	
4	Taken
5	

Table 3. Precluded row indices for the fourth column from three row indices

Row Index	Reason
2	Taken
3	D1(1)+CA(3), D2(1)+CA(2)
4	Taken
5	Taken
6	
7	
8	D1(2)+CA(3)

Table 4. Precluded row indices for the fifth column from four row indices

Row Index	Reason
-3	D1(3)+CA(4)
-2	
-1	D1(1)+CA(4), D3(1)+CA(2)
0	
1	Taken
2	Taken
3	
4	Taken, D1(2)+CA(4), D2(2)+CA(3)
5	Taken
6	D2(1)+CA(3)

F. Allocation of Tasks to Resources

A "case" or task block is defined as a search beginning with a particular set of several row indices – three, four or five, depending on order and the desired task size. We perform task allocation by assigning blocks of cases to a particular computer core. An executive front end runs the search engine, reads the allocation files and writes logs of cases completed. These logs are compiled by an automated bookkeeping program that begins with the output of a comprehensive extended Costas array generator [8][9][12] so that new Costas arrays are apparent when the count changes. The output of every run is a complete set of database files as provided by the authors at CISS 2006 and 2008, and is available on request [9][12].

G. Sets of Four and Eight Costas Arrays Related by Transposition and Rotation

Given any Costas array, seven other permutation matrices can be defined that meet the Costas condition by a combination of transpositions and rotations, or, equivalently, by reversing the order of rows and columns and exchanging rows for columns; the former is more convenient for visualization but the latter is more convenient for implementation by computer. If the Costas array or any rotated version of it is symmetrical there will be duplications and only three different Costas arrays will be found by rotation. Thus asymmetrical Costas arrays come in sets of eight and symmetrical Costas arrays come in sets of four. We refer to distinct Costas arrays as not being related to each other by any combination of rotations and translation. We refer to the sets of four or eight as polymorphs of any one of the Costas arrays is not distinct from any other.

Our example {4,2,5,1,3} is symmetrical so its set of polymorphs has four members. These are shown in Table 5 below.

Table 5. Polymorphs of {4,2,5,1,3}

2	4	1	5	3
3	1	5	2	4
3	5	1	4	2
4	2	5	1	3

For our purposes of examining symmetries, we will use the full generality provided by an asymmetrical Costas array {4,2,3,5,1}, whose polymorphs, ordered by increasing row index, are shown in Table 6 below. This Costas array can be found by using the methods of the above paragraph, "Illustration of the method" or verified by looking at its difference table.

Table 6. Polymorphs of {4,2,3,5,1}

1	4	3	5	2
1	5	3	2	4
2	4	3	1	5
2	5	3	4	1
4	1	3	2	5
4	2	3	5	1
5	1	3	4	2
5	2	3	1	4

III. Innovations in the Search Methodology

A. *Symmetries Used for Efficiency in Costas Array Search Methodology*

1) *Noting Space Searched Provides a Factor of Two*

In Table 5 and Table 6, adding the row indices from the first and last Costas array always results in six or N+1. Thus when stepping through the cases, a row index of k will always have a polymorph with row index N-k+1, so stepping k from 0 to [(N-1)/2]-1, rounded down for even N, will find all the Costas arrays for a particular n if the polymorphs are added to the list. Thus, we reduce the required search space by a factor of about two. And, when the order N is odd and the column number is (N-1)/2, only row indices for column two up to (N-1)/2-1 need be searched for the same reason.

The fact that each Costas array found has four or eight polymorphs opens the possibility that other symmetries, or proper posing of the symmetries with an appropriately structured search methodology, may reduce the required search space by a factor of four, or an additional factor of two. Search methodologies specialized for symmetrical Costas arrays are far more efficient than more general search methodologies and have been used to show that no symmetrical Costas arrays for orders 32 or 33 exist [13]. Thus, with this as an example of possible gains, we look at Table 6 for further symmetries.

One example, the “center dot” invariance is apparent from Table 6. If a Costas array has a one at location ((N-1)/2,(N-1)/2), all Costas arrays obtained by transposition and rotation will also have a center dot.

One economy of search is at the beginning of the search space. Costas arrays beginning with a row index of one in the first column can be found from Costas arrays of one order smaller by “adding a dot” so this part of the search space can be omitted.

2) *Innovations in Symmetry Exploitation*

In addition to eliminating the second half of the first row, the search employs a technique referred to as "progressive exclusion" [14][8] in order to leverage the eight-fold symmetry of the square. We call it progressive exclusion, because as the first column index increases toward the middle, a larger number of possible dots on the edges of the square are marked as not being needed to include in the search. This is possible because exhaustively searching the previous top row dots guarantees that symmetrically corresponding edge dots have been covered. In other words, if the top row has fully explored the first d dots, then the nearest d-1 dots to any corner, in addition to each corner, have been covered through symmetry. It is important to note that the current position being explored is not included in this count since it hasn't been exhausted.

As an example, consider an exhaustive search of Costas arrays of order seven. In the following discussion, dot locations are denoted by their row and column coordinates. When the search first starts out and we are exploring possibilities for the very first dot in the upper-left corner, we may not eliminate any dots due to progressive exclusion (although we may eliminate some due to the normal Costas constraints like row/column orthogonality). That is, the dot in position (7,7), must be considered as a possibility. Once position (1,1) has been fully exhausted, we may eliminate all corner dots from further search efforts, because any Costas array with a dot in any corner would have already been uncovered by exhausting the (1,1) case. This concept is advanced for each column index for which the search is exhausted. Once dots in positions (1,1) and (1,2) have been fully searched, subsequent efforts may omit both the corner dot, and dots one step from any corner. The number of dots excluded progresses as the starting row position approaches the middle, as shown in Figure 1 below:

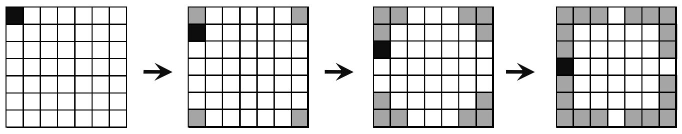
Figure 1. Innovative Symmetry logic

In Figure 1, a black square indicates the current dot position under evaluation in the search. A gray square indicates dot positions that are eliminated from consideration in the search due to progressive exclusion. As a side note, the first step in Figure 1, evaluating dot position (1,1), is not necessary because all arrays of a given order with a dot in position (1,1) may be found by attempting corner dot extensions to the previous order.

We cannot apply the logic directly to the second row and column while searching with the corner dot. The search is directed by the dot on the first row, so that dot is guaranteed to be covered in all symmetry transformations. However, the concept could be extended as follows: If your searched had completed all possibilities with fixed starting dots (1,1) and (2,3), you could eliminate position (3,2) in all subsequent searches under (1,1).

Also, for example, if you searched completely all possibilities under (1,2) and (2,3), then you should be able to eliminate all symmetric representations of that pair of dots from future searches, but the overhead for this would outweigh gains obtained through the reduction in search space.

Note that the last dot in Figure 1 need not be searched because there is no place for the last column index; this is the rational for the maximum row index for column 1.

B. *More than One Level with the Preclusion Table*

An innovation for search methodologies first presented here is the use of preclusion tables for more than one column. We use a count of recursion levels entered as an architecture-independent measure of resource requirements for a search methodology. Figure 2 shows plots orders of recursion entered, with no look-ahead and with full look-ahead, for order 28 with symmetry exploitation, for a search over row indices in a block that begins with row indices {6,26,23,13,15…}.

Note that, with look-ahead, (the dotted line), the vast majority of backtrack programming with preclusion executes at recursion levels 14 through 19, and the only recursion levels above 23 were those that led to the lone Costas array. This supports a conjecture that for column number (equivalent to recursion depth) of 18 and above, that preclusion tables one or two recursion levels higher than the current recursion search will show no available row indices and thus allow backtracking more often. This is indeed the case.

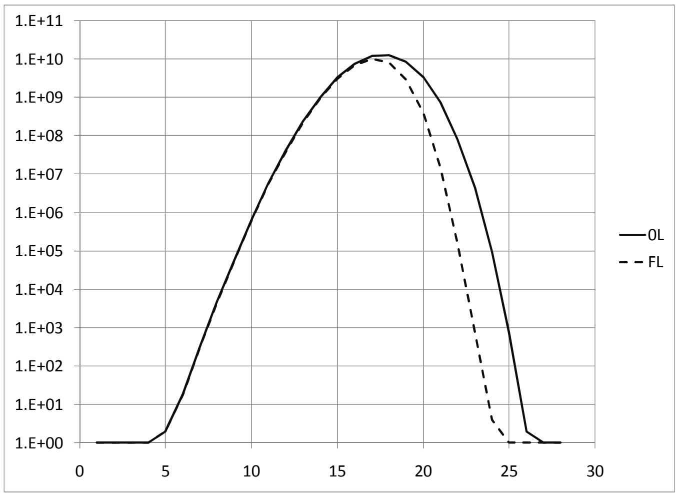

Figure 2. Orders of recursion tabulated over a limited range of search with and without look-ahead

IV. Methods of Generation

A. Concatenation and deletion

1) Taylor and Golomb extensions

Taylor and Golomb have extended the number-theoretic Welch and Lempel-Golomb generators by deleting corner dots and by augmenting existing Costas arrays with "Costas arrays" of order one and two [3][4][5]. Other possibilities exist, such as the example of augmenting two order three Costas arrays to form an order six Costas array.

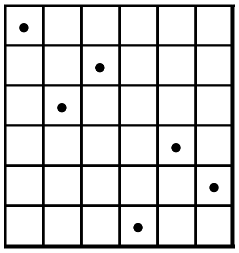

Figure 3. Augmenting two order three Costas arrays

Unfortunately, this interesting example is a rare occurrence that is not seen at higher orders. This example is very interesting because it has two zero quadrants, and the two center antidiagonals are both all zeros.

However interesting this example may be, an examination of the difference matrix reveals that the likelihood of examples for larger orders is vanishingly small. The first half of each row is identical to those of each Costas array, but the number of entries in the difference matrix from dots from both Costas arrays exceeds the number of dots from within each one by about a factor of two, and the likelihood that there will be no duplications becomes vanishingly small for relatively low orders. This is less true when Costas arrays of small orders are used to augment larger Costas arrays, as Taylor's success by adding one and two dots attests.

2) By the Authors

The generators and extensions used by the authors for many years [6][12] uses a form of augmentation that we call *spinning* in which a Costas array of order *N-1* is rotated end-around and single dot is inserted at all possible points in each rotated matrix, and the resulting permutation matrix checked for the Costas condition. Although a number of Costas arrays are found by this method, the number of Costas arrays decreases with order, and none have been found by this method for order greater than 100.

The authors generalized the Lempel-Golomb generator by offsetting the exponents in the powers to which the Galois field elements were taken. Zero is not allowed as the power to which a Galois field element is taken because that means that the other power of a Galois field element must be zero, which is not possible [8]. By adding a dot at the row and column in the resulting permutation matrix where both exponents would be zero, a permutation matrix results in which all dots other than the inserted dot are guaranteed to meet the Costas condition. Occasionally this results in a Costas array not found by any other method.

B. Interleaving

Two permutation matrices of order differing by at most one can be interleaved in a checkerboard to form another permutation matrix. Entries in even numbered rows of the difference matrix are double the entries in the difference matrices of each of the two Costas arrays; this amounts to the condition that the difference matrices of the two Costas arrays not have duplications between them. The odd numbered rows of the difference matrix for the augmented matrix, however, represents counts of the row shifts between the two interleaved Costas arrays, and amounts to a new Costas condition of complexity about twice that of each separately; thus the likelihood of checkerboard augmented Costas arrays for large orders is vanishingly small.

V. Results

A. Effect of Innovations on Costas Array Space

The occurrence of al known Costas arrays from order three through 400, plotted as occurrence as a function of the first row index *a[0]*, is shown as Figure 4. The occurrence is relatively flat across values of *a[0]* because Welch arrays dominate the numbers for large orders, and the singly-periodic quality of these arrays means that whenever one exists for a given value of *a[0]* then another exists for every other value of *a[0]*.

When the symmetry conditions of II.A.2 are applied, the surviving Costas arrays are shown in Figure 5. Note that limiting the search to *a[j]* less than *[(N-1)/2]* provides a factor of two, but that the techniques presented in III.A.2)III.A.2 dramatically reduce the numbers of Costas arrays even in that space.

The searchable space of interest in 2010 is orders through 28. The occurrence of all Costas arrays through order 28 is shown as Figure 6. The effect of the symmetry condition on this distribution is shown as Figure 7.

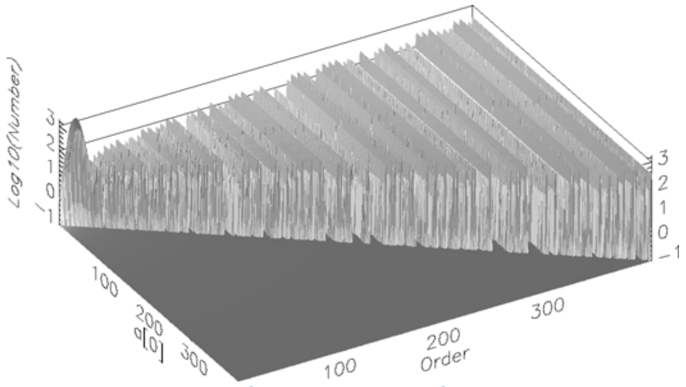

Figure 4. Occurrence of all Costas arrays from orders three through 400 as a function of *a[0]*

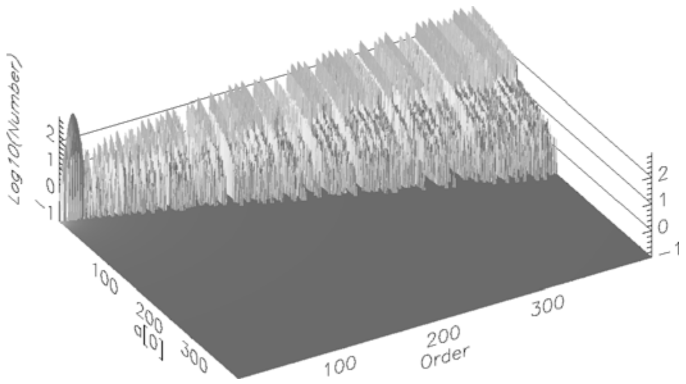

Figure 5. Costas arrays through order 400 that satisfy symmetry exclusion conditions

B. *Discovery of Last Costas Array of Order 27*

One of us (Erickson), using the idle resources of a 38-core processing farm dedicated to compiler development proved pivotal in discovering the final Costas array of order 27. We developed a distribution method that allowed spawning remote jobs on the farm to process cases at an idle priority. We then collected these results into our pre-existing log format for cataloging. To comply with the usage requirements of the farm, we reduced each job to a single case at level five. Each job at this granularity ran for about two minutes, and the farm processed 1,965,582 cases. One of these cases was the newly discovered Costas array shown below with its polymorphs as Table 7. This Costas array was found about the same time as part of another effort using a supercomputer [9][10].

C. *Metric for Representing efficiency of Methodologies*

An implementation-independent measure of the computational resources used in a search is the number of times that a recursion level is increased. Figure 2 above shows a profile of recursion levels entered for a particular limited search. Total recursion-level-entered counts are a measure of total resources used in a run.

D. *Quantification of Gains from the Innovative Methods*

Total recursions-entered counts for complete searches over orders 14 through 20 for no extra preclusion tables and for all available are shown below as Figure 8, both with full symmetry exploitations; a plot with all available look-ahead tables and only conventional symmetry exploitation is included to illustrate that effect. Close analysis of the raw data shows that the extra tables provide just over 50 percent improvement over simply limiting the first row index to [N/2]. The increase in required resources as order increases is about a factor of 4.86.

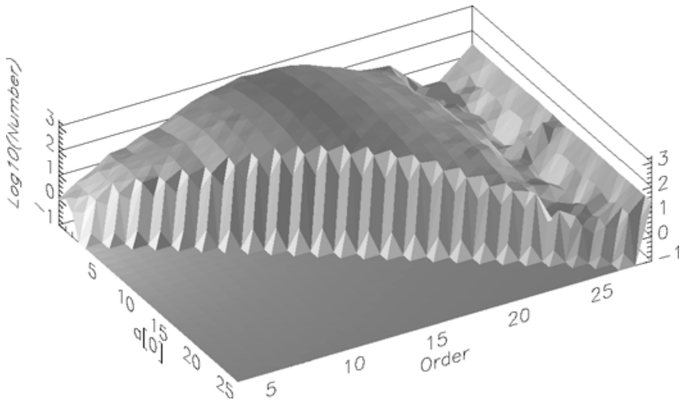

Figure 6. Costas arrays from order three through 28

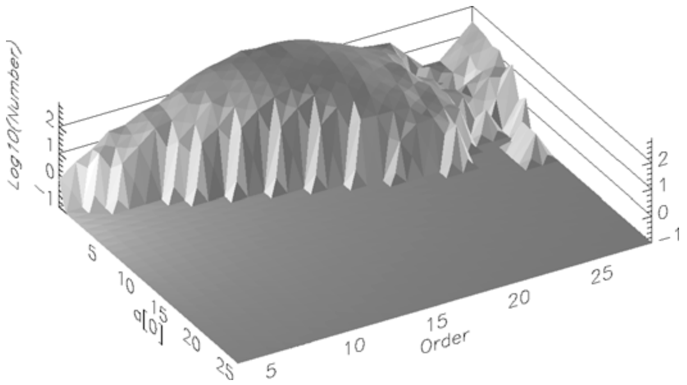

Figure 7. Costas arrays through order 28 that satisfy symmetry exclusion conditions

Because Figure 8 covers several orders of magnitude, the ratios gained by symmetry exploitation and look-ahead preclusion tables are not shown with good accuracy. Figure 9 below shows the ratios of recursions entered with and without each technique. Separate curves are shown for odd and even orders because slightly different properties of the search.

A simple least-squares fit of the form $A + B / N$ to the curves of Figure 9 gives asymptotes of factors of 0.98 for omitting the corner dot, 0.53 for full symmetry exploitation and 0.32 for look-ahead. These curves, extrapolated to orders 27 and 28, predict factors of 0.872 for omitting the corner dot, 0.60 for full symmetry exploitation and 0.43 for look-ahead. These figures are borne out by searches over small ranges of order 28.

The effect of symmetry exploitation is illustrated by a plot of the number of iterations entered, with and without symmetry exploitation; such is shown below as Figure 10.

Note that the number of cases for the searches for the corner dot and the next-to-corner dot is large compared to other row indices for the first column.

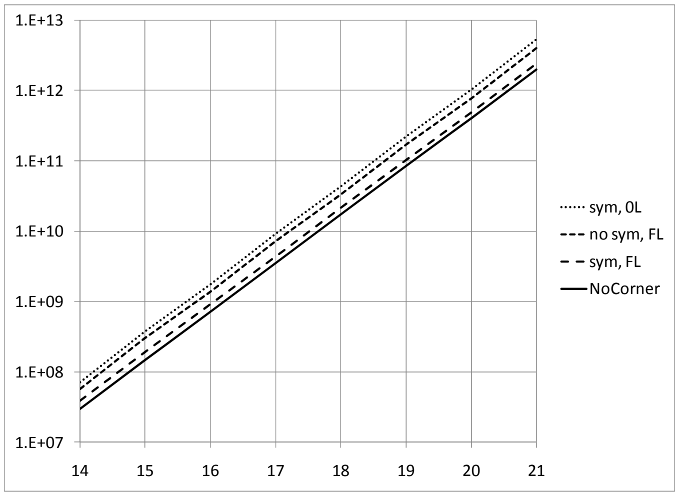

Figure 8. Number of Iterations Entered vs. Order: Effect of Symmetry Exploitation and Multiple Preclusion Tables versus Order of Search

E. *Innovations in Symmetry Exploitation*

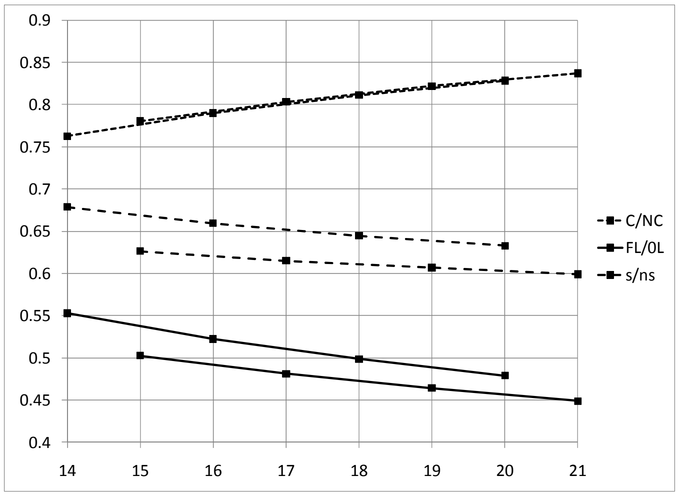

Figure 9. Ratios of Recursions-Entered Gains for Look-Ahead and Symmetry Exploitation for Orders 14 through 20

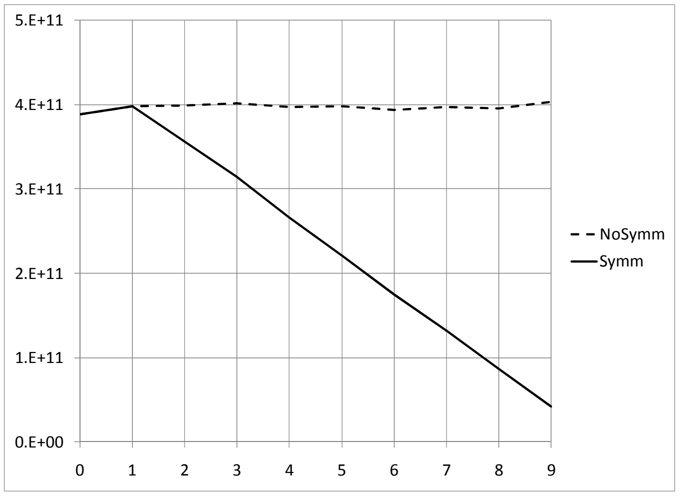

Figure 10. Number of Iterations Entered versus First Column Index for Order 21, With and Without Symmetry Exploitation

The total advantage for a search over order 28, the current lowest un-searched order, is

$$0.60 \cdot 0.43 \cdot 0.872 = 0.225$$

which does *not* include the factor of two obtained by stepping the first row index from 0 to [(N-1)/2]-1.

VI. Conclusions and Conjectures

A. *Conclusions*

We have demonstrated a search methodology with two new innovations: full symmetry exploitation and extended look-ahead for the null set in available row indices. We further point out that the Taylor extension method of adding a corner dot makes search for *a[0]* at zero unnecessary. The combined effect of these is better than a factor of four better than conventional methods that simply limit the search for values of the first row index that stop short of the center point.

B. *Conjectures and Predictions*

An engaging convention that began with Solomon W. Golomb's landmark 1984 paper with Herbert Taylor is presentation of unproven results or simply challenges to the community as conjectures to be proven or disproven. Here we present a few selected conjectures as areas for further work and predictions of future results.

- No Costas arrays above order six exist that have two empty quadrants.
- Orders 32 and 33 will be searched within the next 15 years. None will be found.
- No Costas arrays that are not disclosed in the current literature [6][9][10][12] exist, other than those that may be produced by the existing generators and their extensions as reported on in existing literature [6][13].
- The number of Costas arrays for a given order $N>23$ does not exceed N^2.
- The number of consecutive orders K for which no Costas arrays exist has no upper bound. However, for any K an order N exists for which Costas arrays do exist and $|K/N-1|$ has no lower bound.
- Costas arrays will be used in communications waveforms as part of a shared bandwidth strategy for essentially all digital communications systems. Similarly, Costas arrays will be used in essentially all digital radar waveforms designs as part of a strategy for mitigation of band sharing, mutual interference, interference with other systems, and similar purposes.
- Costas arrays will be used as part or all of designs of digital fingerprinting systems involving imagery and other multidimensional data.

VII. Acknowledgment

Keith G Erickson is responsible for finding the Costas array of order 27 first reported in an email by James K Beard on May 29, 2008 [9][10].

References

[1] John P. Costas, Project Medior – A medium-oriented approach to sonar signal processing, *HMED Technical Publication R66EMH12*, GE Syracuse NY (now Lockheed Martin Marine Systems and Sensors, Syracuse), January 1966.

[2] John P. Costas, “A study of a Class of Detection Waveforms Having Nearly Ideal Range-Doppler Ambiguity Properties,” *Proceedings of the IEEE*, Vol. 72, No. 8, August 1984.

[3] S. W. Golomb and H. Taylor, “Constructions and Properties of Costas Arrays,” *Proceedings of the IEEE* 72(9) pp 1143-2263, September 1984.

[4] Solomon Golomb and Herbert Taylor, “The T-4 and G-4 Constructions for Costas Arrays,” *IEEE Transactions on Information Theory*, vol. IT-38, no. 4, July 1992, pp. 1404-1406.

[5] Oscar Moreno, “Survey on Costas Arrays and their Generalizations,” in *Mathematical Properties of Sequences and Other Combinatorial Structures*, Jong-Seon No, Hong-Yeop Song, Tor Helleseth, and P. Vijay Kumar, Eds., Springer (Kluwer), 2003, ISBN 1-4020-7403-4.

[6] James K Beard, “Generating Costas Arrays to Order 200,” *Conference on Information Sciences and Systems (CISS) 2006* (IEEE IT Society and Princeton University).

[7] Solomon Golomb and Leonard Baumert, “Backtrack Programming,” *JACM*, October 1965, pp. 516-524.

[8] Beard, James K., Russo, Jon C., Keith G. Erickson., Michael Monteleone, and Michael Wright, “Costas Array Generation and Search Methodology,” *IEEE Transactions on Aerospace and Electronic Systems*, 43, 2 (April 2007), 522-538.

[9] Konstantinos Drakakis, Scott Rickard, James K Beard, Rodrigo Caballero, Francesco Iorio, Gareth O'Brien and John Walsh, Results of the enumeration of Costas arrays of order 27, IEEE Transactions on Information Theory 54 10 (October 2008) pp 4684-4687.

[10] James K Beard, announcement of new Costas array of order 27 on personal web site, http://jameskbeard.com/jameskbeard/Costas_Arrays.html#NewCA27.

[11] *MacTech*, Programmer’s Challenge, available on web page http://www.mactech.com/progchallenge/, click on “Costas Arrays (December 1999).”

[12] James K Beard, “Costas array generator polynomials in finite fields,” *CISS 2008*, March 21 2008, Princeton University, Session TP 03, Paper 5; Database of Costas arrays from orders 2 to 200 on a CD-ROM given out at CISS 2006 was extended to order 400.

[13] Oscar Moreno, John Rameirez, Dorothy Bollman, and Edusmildo Orozco, “Faster backtracking algorithms for the generation of symmetry-invariant permutations,” *Journal of Applied Mathematics* 2:6 (2002) pp. 277-287.

[14] James K Beard, Jon C Russo, Keith Erickson, Michael Monteleone, and Mike Wright, “Combinatoric collaboration on Costas arrays and radar applications,” *Proceedings of the IEEE 2004 Radar Conference*, April 26-29 2004, ISBN 0-7803-8234-X, pp. 260-265.

Jon C Russo was born in Geneva, NY in 1969. In 1992, Jon graduated with distinction from the Cornell University School of Electrical Engineering in Ithaca, NY. He stayed at Cornell to complete a Masters in Engineering in 1993, in the field of digital signal processing.

While at Cornell, he was a teaching assistant for electronic design lab and other classes. He went on to join the research team at Lockheed Martin Advanced Technology Labs, working in signal processing, radar, hardware design, and reconfigurable computing and compiler technologies. He has co-authored papers in high performance computing and information technologies. Research interests include cognitive architectures, communications processing, and speech and image analysis. He was a co-author of [8].

Keith G. Erickson (SM’03–M04) became a Student Member in 2003 and a Member of IEEE in 2004. He received his BS in computer engineering (electrical engineering with an emphasis on computer science) from the New Jersey Institute of Technology, Newark NJ, in 2004. He was born in Cherry Hill, NJ in 1982.

He was on the New Jersey Institute of Technology Dean’s List every semester, and was President of the SGA and of the Albert Dorman Honors College. He also was President of Team Corona, a joint Burlington (NJ) County College and New Jersey Institute of Technology team that built an electric car. He was a co-author of [8].

James K Beard (M’64–LM’04) became a Member (M) of IEEE in 1964, Life Member in 2004. He was born in Austin, TX in 1939. He receive a BS degree from the University of Texas at Austin in 1962, an MS from the University of Pittsburgh in 1963, and the Ph. D. from the University of Texas at Austin in 1968, all in electrical engineering.

Between 1959 and 2009, he worked in Government laboratories, industry, as an Adjunct Professor and as an individual consultant. He is the author of [8] and a number of symposia papers and a book, “The FFT in the 21st Century” (Springer, 2003). Current research interests include system engineering solutions to homeland defense issues, estimation and decision theory, radar and communications concept and waveform design, and digital radar concepts.

Dr. Beard is a member of AIAA, AOC, SPIE, and ASA. He is a member of Phi Eta Sigma, Eta Kappa Nu, Tau Beta Pi, and Sigma Xi. He studied for his Ph. D. under a GSRF Fellowship (matched U. Texas Austin and Ford Foundation funding, administered by U. Texas Austin) and a NSF Fellowship.

Table 7. The last Costas array?

11	10	4	24	7	23	3	18	21	9	26	16	5	1	15	27	2	25	17	22	19	6	8	12	20	13	14
12	17	10	24	22	8	19	3	7	20	9	16	13	1	2	4	27	26	18	5	23	6	15	25	21	11	14
14	11	21	25	15	6	23	5	18	26	27	4	2	1	13	16	9	20	7	3	19	8	22	24	10	17	12
14	13	20	12	8	6	19	22	17	25	2	27	15	1	5	16	26	9	21	18	3	23	7	24	4	10	11
14	15	8	16	20	22	9	6	11	3	26	1	13	27	23	12	2	19	7	10	25	5	21	4	24	18	17
14	17	7	3	13	22	5	23	10	2	1	24	26	27	15	12	19	8	21	25	9	20	6	4	18	11	16
16	11	18	4	6	20	9	25	21	8	19	12	15	27	26	24	1	2	10	23	5	22	13	3	7	17	14
17	18	24	4	21	5	25	10	7	19	2	12	23	27	13	1	26	3	11	6	9	22	20	16	8	15	14