Combinatoric Collaboration on Costas Arrays and Radar Applications

James K Beard, Member
Lockheed Martin Marine Systems and Sensors
199 Borton Landing Road, P.O. Box 1027; Mail Station 137-233
Moorestown, NJ 08057-0927 USA

Jon C Russo
Lockheed Martin Advanced Technology Laboratories
3 Executive Campus, 6th Floor
Cherry Hill, NJ 08002 USA

Keith Erickson, Student Member
Michael Monteleone, Student Member
Mike Wright, Student Member
New Jersey Institute of Technology
3331 Rt. 38
Mt. Laurel, NJ 08054 USA

***Abstract*-Costas arrays are permutation matrices that also provide a frequency indexing sequence that permits at most one coincident tone in cross-correlations of FSK waveforms. As such, they have obvious application as frequency indexing sequences in radar and communications when long codes with bounded autocorrelation are required or when Doppler is a significant portion of the transmitted bandwidth. All Costas arrays for orders less than 25 are known, with those for N=24 disclosed here. Higher orders are found through number-theoretic generators and partial searches.**

I. Introduction

Permutation matrices and Costas arrays are denoted here using row-index notation. A key accessory to such matrices is a triangular matrix we call the check matrix.

Definition 1:

Row-index notation expresses an N by N matrix that has a one in each row with zeros elsewhere as an ordered set of N integers, each representing the index of the column with the one, as $(c_1, c_2, c_3, \ldots c_N)$.

Definition 2:

A check matrix is defined from the row-index notation of a permutation matrix. Each $d_{i,j}$ element in row i and column j is

$$d_{i,j} = c_{i+j} - c_j, i \in [1, N],\ j \in [1, N-i]. \quad (1.1)$$

We use the check matrix to determine the key properties of a permutation matrix.

Theorem 1: The N by N matrix given in row-index notation by $(c_1, c_2, c_3, \ldots c_N)$ is a permutation matrix if, and only if,

(a) $1 \le c_i \le N, 1 \le i \le N$,

(b) $d_{i,j} \ne 0, 1 \le i \le N, 1 \le j \le N - i$.

Proof:

(1) Condition (b) applies if and only if no two c_i are the same.

(2) Conditions (a) and (b) together apply if and only if all integers from 1 to N are represented in the column indices exactly once. This completes the proof.

A Costas array is a permutation matrix with a very special property: overlaying a matrix with a shifted replica of itself results in at most one coincident 1.

Definition 3:

A Costas Array is a permutation matrix such that, when shifted up i rows and left k columns, end-off, and overlaid with the un-shifted matrix, there is no more than one 1 in the shifted matrix coincident with a 1 in the un-shifted matrix – unless both i and k are both zero, in which case all N 1's are coincident. This geometry is captured in the check matrix by noting that j is the row number of the unshifted matrix where the coincident 1 appears, and that the number of columns shifted is given by the check matrix entry $d_{i,j}$ as given in (1.1) . Thus, we can use the check matrix to determine whether or not a permutation matrix is a Costas array.

Another way that a Costas array is sometimes defined is a permutation matrix in which no two vectors between 1's are alike. Two such vectors can have the same direction, or the same length, but not both, for if they did, the two pairs of 1's connected by identical vectors could be overlaid by with a properly shifted matrix.

Costas Arrays Come in Sets of Four or Eight

An important property of Costas arrays is that they come is sets of four or eight. This is apparent when one considers that a set of eight Costas arrays can be constructed from a single Costas array:

1) Any Costas array can begin a set,

2) Reversing the order of the rows of a Costas array produces another Costas array,

3) Reversing the order of the columns of a Costas array produces another Costas array,

4) Reversing the orders of both the rows and columns of a Costas array produces another Costas array, and

5) Transposing a Costas array provides a basis for another four Costas arrays.

If a Costas array is symmetrical about the main diagonal or antidiagonal (the "attacking bishops" or "attacking queens" case) then the transpose will be a duplicate of one of the other cases, and this Costas array will be part of a set of four, not eight.

Theorem 2: A permutation matrix is a Costas array if, and only if, no two elements in a given row of the check matrix are equal.

Proof:

Consider a permutation matrix and a shifted permutation matrix, shifted i rows down and a number of columns to the right taken from row i of the check matrix $d_{i,j}$, and overlaid on the unshifted permutation matrix.

(a) Each row i of the check matrix $d_{i,j}$, $j \in [1, N-i]$ represents a shift of i rows of the permutation matrix.

(b) Each entry $d_{i,j}$ in the check matrix represents the number of columns to shift that make the 1 at c_{i+j} coincide with the 1 at c_j in the unshifted matrix. The one is in column c_j in the unshifted matrix and column c_{i+j} in the shifted matrix.

Thus, when no two elements in any row of the check matrix are equal, the permutation matrix has the Costas property. This completes the proof.

Examples

A Costas array of order 5 is $(5,4,6,2,3,1)$. This Costas array and its check matrix are shown below in Table I.

Table I Costas Array and Check Matrix Example

5	**4**	**6**	**2**	**3**	**1**
-1	2	-4	1	-2	
1	-2	-3	-1		
-3	-1	-5			
-2	-3				
-4					

The matrix expressed using row-index notation as the sequence of positive integers from one to N is the identity matrix, as $(1,2,3,\ldots N)$. In the check matrix of the identity matrix the first row is all ones, the second row is all twos, etc. This is an example of a corollary that the number of equal elements in row i of a check matrix is the maximum number of coincident 1's when a matrix is shifted by i rows is overlaid on an un-shifted matrix, for any number of columns shifted. The number of columns shifted is equal to the repeated value. And, the set of absolute values of a check matrix for any permutation matrix will include exactly N-1 1's, N-2 2's, etc. The converse is not true – an arbitrary set of N-1 1's, N-2 2's, etc. distributed through a triangular matrix will not necessarily define a valid permutation matrix. We use simultaneous definition of a Costas array and the check matrix in fast methods for exhaustive searches, but the methods are very structured.

A permutation matrix, used as an operator on a column vector whose elements are equal to the row number, produces a column vector whose elements are the row-index notation of the permutation matrix.

Difficulty in Finding Costas Arrays

The definition of a Costas array as a permutation matrix with special restrictions does not lead to a simple method of finding them because the Costas condition is not easily posed in a clearly simple way such as a simple set of constraint equations. The only known way to obtain all Costas arrays for a given order is an exhaustive search. The number of permutations of order N is $N!$, while the number of Costas arrays of order N increases to a maximum of 21,104 for $N = 16$ after which the number drops rapidly. Figure 1 below shows a curve for orders 1 through 24. Results presented for the first time here include that there are 200 Costas arrays of order 24.

Exhaustive search, such as sequential generation of all $N!$ permutation matrices and examining the check matrix to determine which have the Costas property, is prohibitively slow for large N. MacTech, a journal for serious Apple Macintosh developers and users, had generation of all

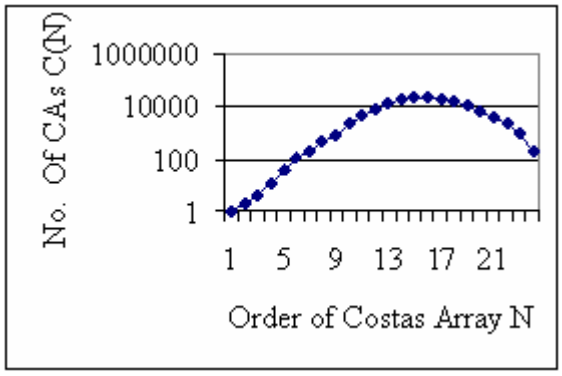

Figure 1 Number of Costas Arrays for Orders to 24

Costas arrays of order 24 as its contest for December 1999 [1]. The winner produced a fast method of doing a search, but it was not fast enough to produce a search for order 24. The article was not clear on how high they did go. Until the disclosure in this paper, an exhaustive search had not been completed for order greater than 23 [2] [3].

Conventional Sieve Search Method

The method most often used up to the present, including in the Mac Challenge solution, sequentially builds up the column index representation. This method uses the principle of the check matrix to build a mask of allowed values for the next row using the portion of the check matrix that exists at that point, taking the next available value in the active row, and proceeding to the next row. When an available value is found for the last row, a Costas array has been found. When a row is found that has no allowable values, the previous row is searched for the next available column index. Denoting this as a sieve method derives from an analogy between the mask of allowed values for the next column index and the sieve of Eratosthenes.

A New Exhaustive Search Method

Here we disclose an improved search algorithm by JCR that is based on computing and storing the necessary elements of the difference table as the recursion progresses. The disallowed column index values of the next row are determined by combining the vector difference information of this table with the previously used column data.

The method begins with the first three row indices to allow breaking up the problem for collaboration (see the next Section). For a given predefined 3 row array, the remainder of the branch was covered recursively according to the following:

Let ***costas***[0..n-1] be a vector containing the current Costas array in column index representation.

Let ***used*** be a bit field flagging the column positions currently used in the Costas array.

Let ***diff***[0..n/2-1] be a vector of bit fields flagging the current used differences for the $(k+1)^{th}$ order of separation.

The variable ***costas*** can be a static array, whereas ***used*** and ***diff*** are "stack" variables in that their old values are stored and recovered as the recursive algorithm "retreats".

Note that bit field ***diff*** must be large enough to contain all the possible differences. For the case of n=24, the differences can range from -23 to +23.

We use length 16 bit fields in the following example for brevity; 32-bit fields are used in implementation to allow the method to be used for larger orders. Consider the starting 3 rows: ***costas*** = [0 1 3]. With the convention that the Least Significant Bit is the right-most bit,

$$\begin{aligned} used &= 0000000000001011 \\ diff[0] &= 0000000000000110\ \ 0000000000000000 \end{aligned} \quad (1.2)$$

Note that ***diff***[0] is shown built out of two 16-bit fields (32-bit fields in implementation), and that in this representation, the bit immediately left of the space corresponds to a difference of zero, which never occurs. The next two bits left of this correspond to the differences 1-0=1, and 3-1=2, respectively. The bit immediately right of the space corresponds to a difference of -1, and becomes increasingly negative with distance to the right. The exact construction of the bit fields is a convention which may be optimized for different implementations.

From this point, the possible positions for the next row are determined by shifting the ***diff***[0] bit field left by the current row's value and combining this with the ***used*** mask via bitwise OR:

$$\begin{aligned} used &= \quad 0000000000001011 \\ diff[0] &= 00000000000001100000000000000000 \\ OR &= \quad 0000000000111011 \end{aligned} \quad (1.3)$$

So in row index 3 (counting from 0), we know that the values 0,1,3,4, and 5 are not possible and prune those branches from the rest of the search. The algorithm recurs over the determined possible values. In this example the value tried for row 3 would be "2", so, ***costas*** = [0 1 3 2]. The 1-separation difference field is updated to include the current value for row 3, and shifted left by 2. We must also add a new 2-separation difference field, with a bit set corresponding to ***costas***[2]-***costas***[0]= 3-0 = 3, which is subsequently shifted left by 3 (***costas***[2]) as shown below:

$$\begin{aligned} used &= \quad 0000000000001111 \\ diff[0] &= \ 00000000000001101000000000000000 \\ diff[1] &= 00000000000010000000000000000000 \\ OR &= \quad 0000000001011111 \end{aligned} \quad (1.4)$$

An important note is that it is not necessary to form ***diff***[1] and apply until this level. Although it is possible to form and apply a 2-separation difference field with an array of only 3 elements, it is not necessary because a redundant operation with a level-1 difference is performed. In the above example, ***costas***[2]-***costas***[0]+***costas***[1] = ***costas***[1]-***costas***[0]+***costas***[3]. Making use of this symmetry saves half of the difference table computations. In general, at the k^{th} level of recursion, it is only necessary to have floor(k/2) difference fields to determine how to proceed on to the next level.

The recursion progresses until there are no possible branches to continue, or until a full Costas array has been determined. If either of these conditions occurs, the algorithm retreats until there is a possible unchecked position, and continues from there.

II. COMBINATORIC COLLABORATION

Overview

Our combinatoric collaboration consisted of three elements:

1) The algorithm by JCR, with an initialization shell that allows the algorithm to be started for three column indices that are set as inputs,

2) Available, idle computational resources including Suns owned by Lockheed Martin Advanced Technology Laboratories on nights and weekends, personal computers belonging to all authors, a Beowulf cluster of castoff machines provided by KE, MM, and MW, and the routine work computers left idle on weekends, and

3) A bookkeeping and dynamic problem parsing and allocation that supported collaboration of separate facilities, automated and operated by JKB.

Collaborative methods

The collaboration operated by allocation of different parts of the problem to the various resources. The parts of the problem are denoted by the first three column indices that initialize the algorithm. The algorithm can be stopped at any time on any resource, leaving a truncated log file. These log files were sent to JKB, who maintained an automated bookkeeping scheme that kept track of the cases run and the Costas arrays found. In this way, resources of diverse types were coordinated with an absolute minimum of duplication.

Beowulf Clustering Techniques

LTSP, the Linux Terminal Server Project, was used on the server, which allows remote booting of diskless clients. Diskless clients boot up and telnet into the application server seeking either a command line or a graphical interface. The "users" can all run programs off a single server. The local system is rarely used, and the kernel is minimalized.

We boot the clients and they wait for the server to send them jobs. Although LTSP is geared for multiple users on multiple machines running programs off of one machine, a single user on a single machine runs programs on multiple machines.

We use Condor, software developed at The University of Wisconsin for the purpose of cross-platform computing, to spawn processes over a network of heterogeneous machines. We also used industry standard implementations of MPI(Message Passing Interface) and PVM (Parallel Virtual Machine) for comparison purposes, and when using temporarily idle machines in off-hours.

Resources

Table II below lists the resources used in our search.

Table II Resources used in search

Manufacturer	Model or Motherboard	Processor
Sun	Blade 1000 Model 1750	UlstraSparc-III 750 MHz
Sun	Ultra 10 Model 360	UltraSparc-II 360 MHz
Parts Built	ASUS A7M266-D	AMD Athalon MP2200 1.8 GHz (dual)
Parts Built	ASUS A7V8X	AMD 2600+
Parts Built	ASUS A7V8X	AMD 2200+
Parts Built	ABIT AT7-MAX2	AMD 2600+
Parts Built	Creative Blaster Board	Intel Celeron 700
Parts Built	ASUS A7V8X-X	AMD 2200+
Parts Built	ASUS A7N8X Deluxe	AMD 2600+
Parts Built	ASUS A7N8X Deluxe	AMD 2400+
Parts Built	ABIT KT7A	800 Duron
Parts Built	ABIT KR7A	AMD 1800+
Parts Built	ASUS A7N8X	AMD 2200+
Parts Built	ABIT KG7-Raid	AMD 1500+
Parts Built	Tyan Trinity 400 S1854	Pentium III 1000
Aspect	VIA 133A	Pentium III 866
Hewlett Packard	Pavilion	AMD 1.0 GHz
Parts Built	ASUS K7A	AMD K7 700
Parts Built	Intel i440BX	Pentium II 450
Parts Built	Intel i845	Pentium IV 2.4GHz
Compaq	Deskpro EN	Pentium III 1000 (x30)
Compaq	Deskpro EN	Pentium III 800 (x30)
Compaq	Evo N600	Pentium III 866 (x45)
Parts Built	Beowulf Cluster	Pentium I 100 (x30)

III. RADAR APPLICATIONS

Costas arrays were originally used as frequency shift schemes in FSK waveforms for sonar applications [4] [5]. Software defined radio (SDR) is an enabling technology for evolving digital wireless communications infrastructure such as cell phones [6]. An enabling technology for next-generation digital communications is the Costas loop [7] [8] that allows phase-locking on a suppressed carrier signal without frequency doubling. An apparently unrelated application is digital watermarking, in which an embedded FSK signal defined as a set of frequency hops defined by a Costas array provides the hook for synchronization and detection of the codes [9]. All of these applications accrue because of the fundamental property that J. P. Costas needed for sonar signals [4] [5]: an ambiguity function with ideal properties. An ambiguity function is a property of a waveform, and is the response of a signal processor to the waveform versus time, with a second variable of frequency offset (usually radar or sonar Doppler shift or communications frequency mismatch) providing a surface. A frequency-shift waveform in which the frequency shifts are determined according to a Costas array pattern can have an ambiguity function in which the highest sidelobe is down by an amplitude factor of N, the order of the Costas array. An example for an order 24 Costas array is shown below as Figure 2. The ideal bed-of-nails sidelobe structure is apparent in this three dimensional plot, but the 27.6 dB sidelobe performance is shown more clearly in the zero Doppler slice shown in Figure 3.

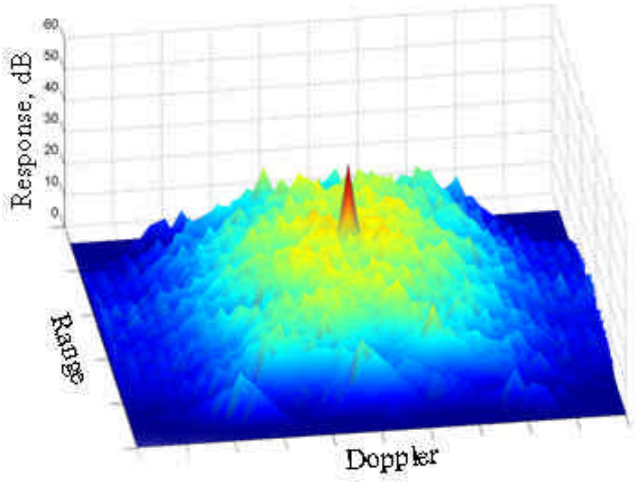

Figure 2 Ambiguity Function for FSK Waveform Based on Order 24 Costas Array

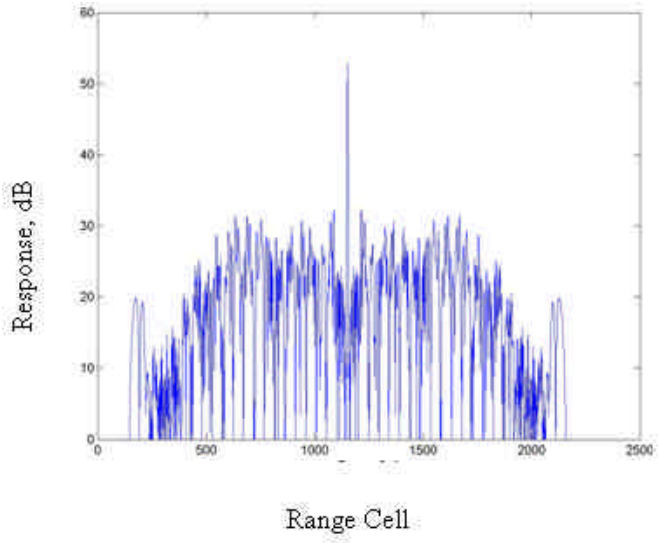

Figure 3 Zero-Doppler Slice of Ambiguity Function

Studies have been made of cross-correlation of FSK waveforms based on Costas arrays with results similar to those of BPSK radar and communications waveforms [10].

IV. RESULTS AND CONCLUSIONS

Costas Arrays

There are 200 Costas arrays of order 24. None of them are symmetrical. Table III below lists 25 Costas arrays of order 24. Each of them is a basis for a set of eight, for a total of 200

Number-theoretic generators provide Costas arrays for a wide variety of orders, and there are some extensions based on augmenting or decrementing rows and columns of existing Costas arrays [11]. JKB has extended upon these. These generated Costas arrays provide an existence proof of Costas arrays for most orders, and prove certain early conjectures in [11] such as no upper bound on the number of Costas arrays or the orders for which Costas arrays exist. Figure 4 below shows the total number of Costas arrays for orders up to 24 against a background of numbers of generated Costas arrays of order up to 200. With this work, all existing Costas arrays up to order 24 are available, and number-theoretic generators with extensions provide plentiful Costas arrays of larger orders.

Radar Applications

With emerging technologies for exhaustive search as presented here, sufficient palettes of moderate order Costas arrays are available to produce effectively ideal autocorrelation performance with cross-correlation performance similar to that of BPSK codes. These waveforms are particularly useful in applications where the target Doppler is a significant portion of the radar bandwidth, such as radars designed to track extra-atmospheric objects, particularly in highly eccentric orbits such as those with Molnyia orbits or ballistic objects such as launch vehicles. Other radar types that can use FSK waveforms based on Costas arrays include CW or quasi-CW bistatic radars and high duty cycle radars with broadband chips that are transmitted in an FSK pattern based on Costas arrays.

ACKNOWLEDGMENT

The authors thank Lockheed Martin Advanced Technology Laboratories for allowing use of idle computers.

REFERENCES

[1] MacTech, December 1999 Programmer's Challenge, available on web page http://www.mactech.com/progchallenge/9912Challenge.html
[2] S. Rickard, "The quest for Costas arrays," IEEE AES Regional Meeting presentation, Rowan University, October 8, 2002.
[3] J. K. Beard, "Costas arrays, properties and generators," IEEE AES Regional Meeting presentation, Rowan University, October 8, 2002.
[4] J. 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.
[5] J. P. Costas, "Medium constrains on sonar design and performance," in *FASCON Conv. Rec.*, pp 68A-68L, 1975.
[6] C. Dick and F. Harris, "FPGA DSPs – the platform for next-generation wireless communications." RF Design, October 2000, pp 56-66.
[7] J. P. Costas, "Synchronous Communications," Proc. IRE, Vol 44, pp 1713-1718, December 1956.
[8] B. Sklar, "Digital Communications," Prentice-Hall (1988), pp 447-448.
[9] M. F. Bocko, "Data hiding in digital audio files," IEEE Signal Processing Society, Rochester chapter, March 5, 2003.
[10] Avi Freedman, "Trains of Costas bursts and their ambiguity function," M.S. Thesis, Tel Aviv University, October 1985.
[11] S. W. Golomb and H. Taylor, "Constructions and Properties of Costas Arrays," Proc. IEEE 72(9) pp 1143-2263, September 1984.

Table III Fundamental Costas arrays of order 24

0	2	17	18	23	10	3	12	1	15	22	5	21	6	14	11	9	19	13	8	7	20	16	4
0	3	20	7	5	12	18	4	15	14	22	23	11	1	13	17	6	8	21	16	9	19	10	2
0	4	23	15	11	5	7	2	1	10	3	17	20	8	19	9	6	14	21	12	13	18	16	22
0	11	15	21	17	7	8	6	20	12	9	19	14	2	4	23	22	1	18	5	13	16	10	3
1	15	19	12	7	10	20	0	11	17	3	21	23	22	5	6	4	9	18	2	14	8	16	13
2	13	8	23	11	15	5	14	0	7	1	19	12	10	9	6	16	21	3	4	17	20	22	18
3	2	15	12	21	14	10	4	6	23	9	20	7	19	22	13	5	0	16	1	11	17	18	8
3	14	10	20	13	11	6	23	22	19	1	16	2	21	0	8	17	7	12	15	4	5	18	9
3	21	8	10	16	7	2	18	11	19	4	17	0	1	13	20	23	22	14	12	6	15	5	9
4	3	9	19	21	17	15	0	11	14	22	1	23	10	2	20	8	5	12	13	7	16	6	18
4	7	9	20	19	14	8	22	12	17	5	1	21	2	11	23	10	3	16	0	18	15	6	13
4	9	15	2	13	22	21	5	8	1	23	3	18	12	0	17	14	16	20	10	6	7	19	11
4	13	7	8	21	17	5	15	22	1	18	23	0	20	12	11	2	10	3	14	16	19	9	6
4	14	15	6	10	9	1	13	22	19	21	2	23	7	0	20	16	5	3	17	12	18	8	11
4	19	14	16	8	21	7	23	2	11	15	3	13	10	0	22	9	5	12	20	1	6	18	17
5	2	11	14	16	12	13	23	9	7	18	1	17	8	3	22	21	15	0	4	19	6	20	10
5	14	3	21	13	15	0	19	1	9	20	4	16	12	2	18	22	23	6	11	17	10	8	7
5	14	8	18	20	3	4	16	15	12	17	7	23	0	21	10	2	19	1	9	22	6	13	11
5	19	3	15	11	21	13	2	22	20	6	10	7	1	0	23	16	17	4	12	14	9	18	8
6	9	16	14	7	11	0	17	2	21	20	15	23	13	5	1	22	10	12	18	4	19	3	8
6	21	14	9	13	18	19	7	3	5	4	23	20	0	12	15	2	11	22	8	16	10	1	17
6	21	15	16	12	2	13	4	20	17	5	0	10	8	14	23	7	19	18	22	9	1	3	11
7	12	18	13	20	10	9	17	1	3	19	23	6	2	5	15	0	21	14	11	22	4	16	8
7	14	2	11	5	21	18	20	19	10	0	4	23	6	12	22	3	1	15	8	13	16	17	9
8	17	1	5	13	19	20	6	23	0	18	15	10	21	14	4	16	3	2	22	11	7	9	12

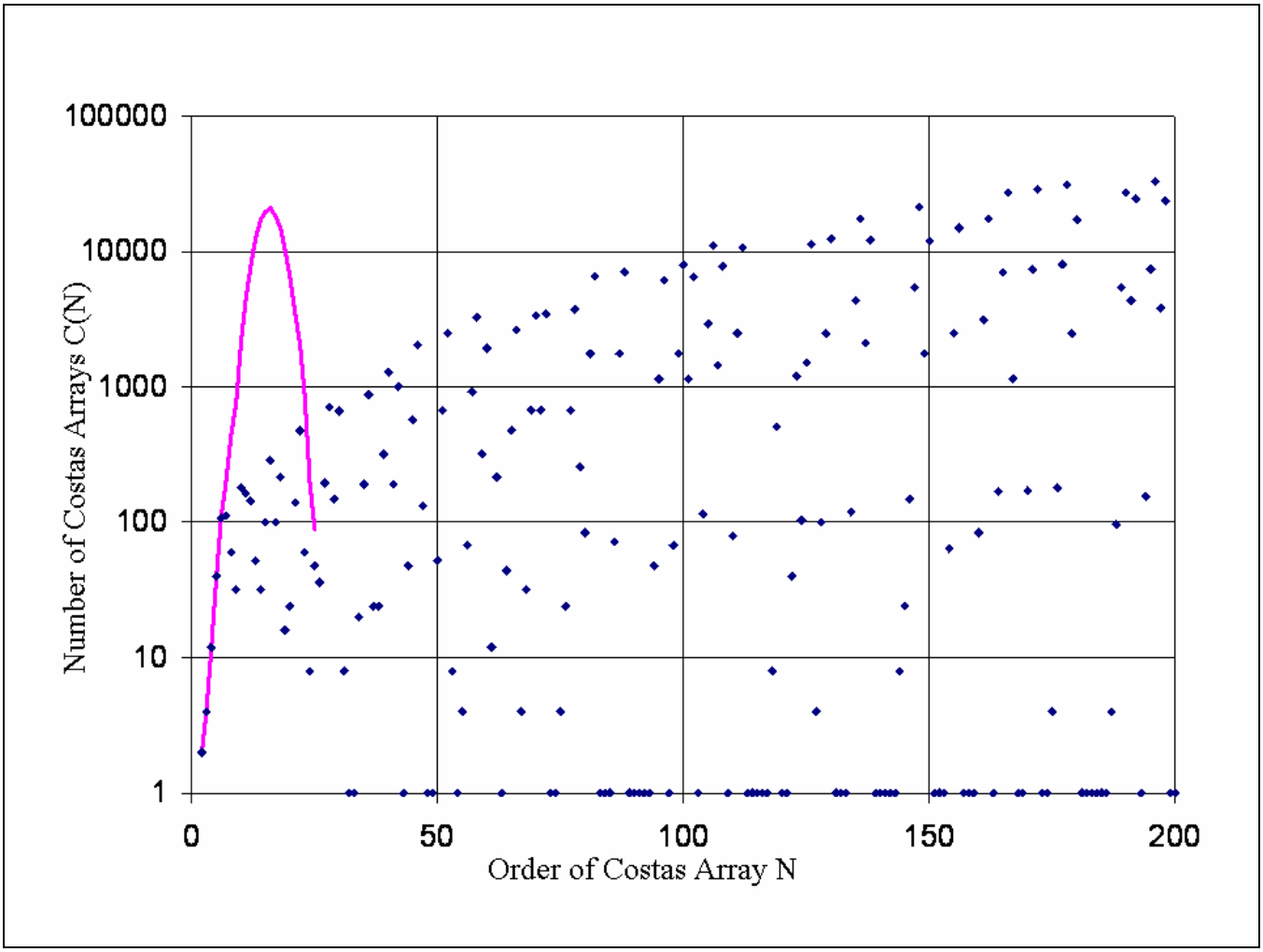

Figure 4 Orders of Costas Arrays, All to Order 24, Generated to Order 200