

De Bruijn Sequences of Higher Dimension

Professor Karel Casteels, Ted Tinker

Abstract

De Bruijn sequences are objects from discrete math relating to combinatorics, especially graph theory. This paper briefly reviews an algorithm for generating De Bruijn sequences before discussing De Bruijn torii, which extend the properties of De Bruijn sequences to two dimensions. De Bruijn sequences and torii have applications in fields as diverse as “robotic vision, location detection, and projective touch-screen displays” [3], “pseudo-random arrays, and the design of mask configurations for spectrometers” [4], magic tricks, ATM cracking, DNA sequencing, and digital paper [6].

This paper explores the complexity of De Bruijn torii through an example. We then outline a construction method for producing objects exhibiting the De Bruijn property in three dimensions. Then we extend the construction inductively to prove that analogues to De Bruijn sequences exist for every finite number of dimensions.

Finally we speculate at some implications of this finding on applications of De Bruijn sequences and torii.

Contents

1	Introduction	3
2	Background	4
2.1	De Bruijn Sequences	4
2.2	The Toroidal Array	6
2.3	De Bruijn Arrays	6
2.3.1	Sub-Matrix Mapping	7
2.3.2	Hexadecimal Tiling	8
3	Uniqueness of the Clockwise Array	9
4	Transitioning to Three Dimensions	18
4.1	Notation for $3D$ Arrays	18
4.2	The Hyper-Toroidal Array	19
4.3	Defining De Bruijn Arrays in Three Dimensions	19
5	Constructing $3D$ De Bruijn Arrays	20
5.1	A Layer-By-Layer Construction	20
5.2	Proving $[F]$ is a $16 \times 4 \times 4$ De Bruijn Array of b -cubes	21
5.3	Tools for Possible Alternative Constructions	29
6	Higher Dimensions	31
6.1	Constructing an $(n + 1)D$ De Bruijn Array	32
6.2	Proving that $[M]$ satisfies Definition 13	33
7	Applications of De Bruijn Sequences and Torii	35
7.1	Mapping of Space and $3D$ Position Tracking	35
7.2	DNA Sequencing and Efficient Packing	35
7.3	Error Detection and Correction	36
8	Further Questions	38
9	Appendix	39
9.1	Encoding and Decoding messages in De Bruijn Arrays (Section 7.4.1)	39
	References	41

1 Introduction

De Bruijn sequences are objects from discrete math relating to combinatorics, especially graph theory. An example of a De Bruijn sequence is the cycle (000111212220101200202102211) which exhibits the property of containing every 3-length combination of the elements $\{0, 1, 2\}$ precisely once, and nothing else:



Figure 1: A visual example of a De Bruijn sequence

De Bruijn torii extend the properties of De Bruijn sequences to two dimensions. De Bruijn sequences and torii have applications in fields as diverse as “robotic vision, location detection, and projective touch-screen displays” [3], “pseudo-random arrays, and the design of mask configurations for spectrometers” [4], magic tricks, ATM cracking, DNA sequencing, and digital paper [6].

We first propose to explore the complexity of De Bruijn torii through example. Just as De Bruijn sequences are cyclic, De Bruijn torii will ‘loop’ horizontally (first column to last) and vertically (first row to last); we will formalize the notion of a matrix shaped like a torus to facilitate discussion of these objects.

Then we will outline a construction method for producing objects exhibiting the De Bruijn property in three dimensions. The construction method will suggest a lower bound for the quantity of these objects. Then we will extend the construction inductively to prove that analogues to De Bruijn sequences exist for every finite number of dimensions.

Finally we speculate at some implications of this finding on applications of De Bruijn sequences and torii.

2 Background

2.1 De Bruijn Sequences

There are several conceptually equivalent notations for De Bruijn sequences, so we here declare our own preferring clarity over compactness for consistency in higher dimensions.

Definition 1. Given positive integers j and n , a *De Bruijn sequence of j -length n -sequences* is a cycle containing precisely one copy of each j -length sequence of the elements $\{0, 1, 2, \dots, n - 1\}$ as a subsequence.

(000111212220101200202102211) is a De Bruijn sequence of 3-length 3-sequences. Informally this means that when written in a circle and viewed three consecutive digits at a time, every 3-length sequence of the elements in $\{0, 1, 2\}$ will be displayed precisely once in each rotation.

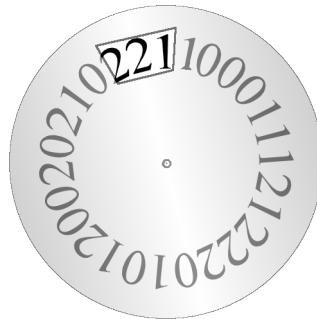


Figure 2: A visual example of a De Bruijn sequence (copied from figure 1)

On the next page, we demonstrate the method used to construct this De Bruijn sequence of 3-length 3-sequences. Though we will not prove this here, the methodology can be extended inductively to produce De Bruijn sequences for any positive integers j and n [1].

Create a graph with nine nodes representing all length-2 sequences of elements in $\{0, 1, 2\}$. For $a, b \in \{0, 1, 2\}$, connect node (ab) to nodes $(b0)$, $(b1)$, and $(b2)$ with arrows, or directed edges.

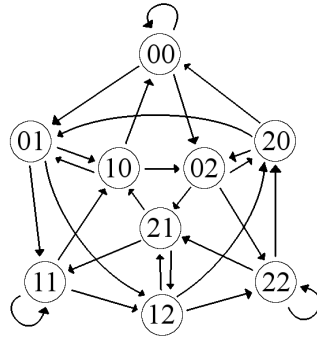


Figure 3: A graph describing possible relations between 2-length 3-sequences

The graph in figure 3 is balanced, meaning every node has exactly as many arrows entering as exiting. Balanced graphs have at least one Eulerian path, a circuit which follows each arrow precisely once. The graph's Eulerian paths represent De Bruijn sequences of 3-length 3-sequences. The red circuit in figure 4 below represents the De Bruijn sequence introduced in figure 1 and figure 2.

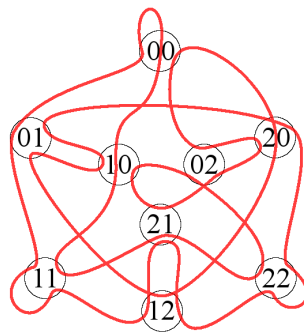


Figure 4: The Eulerian path representing (000111212220101200202102211)

Figure 3 is an example of a De Bruijn graph, and the theory behind such graphs can be inductively extended to produce De Bruijn sequences for any positive integers j and n [1].

The intuitive generalization of the cyclic De Bruijn sequence to two dimensions requires the notion of a matrix shaped like a torus. We will call such an object a toroidal array, using the word array to refer to multidimensional collections of elements with the intention to later define hyper-toroidal arrays.

2.2 The Toroidal Array

For consistency with later sections and for intuitive use of the modulus function, index rows and columns of matrices beginning from 0 rather than 1.

Definition 2. An $m \times n$ *Toroidal Array* is an equivalence class of $m \times n$ matrices under the relation $A \approx B$ if A and B are both $m \times n$ matrices and there exist integers x and y with $0 \leq x < m$ and $0 \leq y < n$ such that for all entries of A and B , $A_{i,j} = B_{k,l}$ with $k = i + x \bmod m$ and $l = j + y \bmod n$.

For example, suppose

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

$A, B,$ and C are 3×3 matrices. $A_{i,j} = B_{k,l}$ with $k = i + 2 \bmod 3$ and $l = j$, while $A_{i,j} = C_{k,l}$ with $k = i + 1 \bmod 3$ and $l = j + 1 \bmod 3$. Therefore $A, B,$ and C are equivalent as representations of the same 3×3 toroidal array. Referring to this toroidal array as $[A]$, we may write $A, B, C \in [A]$.

Definition 3. An $m \times n$ toroidal array $[A]$ *contains* a $k \times l$ matrix B (with $k \leq m, l \leq n$) if there exists a matrix $A \in [A]$ so $A_{i,j} = B_{i,j}$ $\begin{matrix} 0 \leq i < k, \\ 0 \leq j < l \end{matrix}$. If A is the only matrix satisfying this property in $[A]$, then $[A]$ contains B *precisely once*.

2.3 De Bruijn Arrays

Now that the necessary definitions are in place, we may focus on a specific, well-known case of the $2D$ De Bruijn object to demonstrate their complexity.

Definition 4. A 4×4 *De Bruijn array of 2×2 0-1 sub-matrices* is a 4×4 toroidal array which contains every 2×2 0-1 matrix precisely once.

Theorem 2 will prove the toroidal arrays represented in figure 5 are the *only* 4×4 De Bruijn arrays of 2×2 0-1 sub-matrices. For ease of reference in text denote them the ‘‘Clockwise’’ and ‘‘Counterclockwise’’ arrays, respectively.

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Figure 5: The Clockwise and Counterclockwise arrays

The challenge of proving the uniqueness of these arrays calls for new notation.

2.3.1 Sub-Matrix Mapping

Given $m \times n$ toroidal array $[A]$ and any $m \times n$ matrix $A \in [A]$, generate $2m \times 2n$ matrix B such that $B_{i,j} = A_{k,l}$, with

$$\begin{matrix} 0 \leq i < 2m, \\ 0 \leq j < 2n \end{matrix}$$

$$k = \left\lceil \frac{i}{2} \right\rceil \bmod m \quad \text{and} \quad l = \left\lceil \frac{j}{2} \right\rceil \bmod n.$$

Draw a line between every other column and every other row of B to group its elements into 2×2 boxes. The box of B 's elements in the x^{th} row and y^{th} column of B 's boxes, with $0 \leq x < m$ and $0 \leq y < n$, has the following form.

$$\begin{bmatrix} A_{x,y} & A_{x+1 \bmod m,y} \\ A_{x,y+1 \bmod n} & A_{x+1 \bmod m,y+1 \bmod n} \end{bmatrix}$$

B therefore displays which 2×2 matrices $[A]$ contains, even those not immediately apparent in A . Denote this process $A \xrightarrow[\text{map}]{} B$.

Theorem 1. *The Clockwise and Counterclockwise arrays are 4×4 De Bruijn arrays of 2×2 0-1 sub-matrices.*

Proof. Find the sub-matrix map for any representative of the Clockwise array.

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \xrightarrow[\text{map}]{} \begin{array}{|c|c|c|c|} \hline 0 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 0 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 1 & 0 \\ \hline \end{array}$$

Figure 6: A map of the sub-matrices in A

Each of the sixteen boxes of elements holds a different 2×2 combination of 0s and 1s. Therefore, the Clockwise array contains each of the sixteen possible 2×2 0-1 sub-matrices precisely once, satisfying Definition 4. The proof for the Counterclockwise array is identical. \square

Even armed with sub-matrix mapping, it may be difficult to visually ensure each of the sixteen possible 2×2 0-1 sub-matrices is contained precisely once. Hexadecimal tiling will assist in visual understanding.

2.3.2 Hexadecimal Tiling

In hexadecimal tiling, each 2×2 0-1 matrix is assigned a symbol. These symbols use black space to represent 0s and white space to represent 1s.

Matrix	Symbol	Matrix	Symbol	Matrix	Symbol	Matrix	Symbol
$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$		$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$		$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	
$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$		$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$		$\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$		$\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$	
$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$		$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$		$\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$		$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$	
$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$		$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$		$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$		$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$	

Figure 7: The Hexadecimal Tiles

Given an $m \times n$ toroidal array $[A]$ containing only 0s and 1s, generate its hexadecimal tiling C by generating the sub-matrix map B for any $A \in [A]$, then replacing each box in B with the hexadecimal tile corresponding to the 2×2 matrix it contains. Denote this process $A \xrightarrow{\text{map}} B \xrightarrow{\text{hex}} C$.

Hex tiling the Clockwise array shows it contains precisely the 2×2 0-1 matrices.

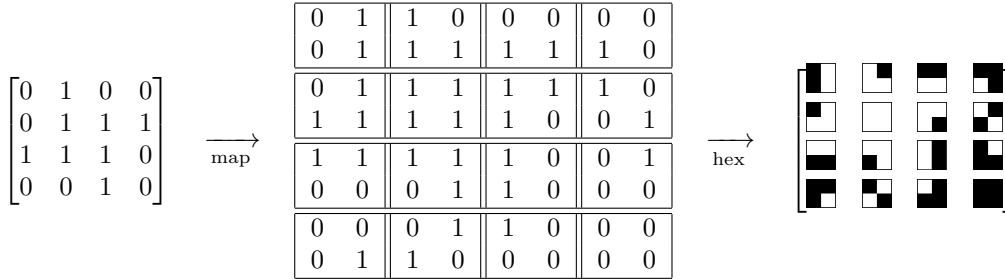


Figure 8: Tiling the Clockwise Array

Neighboring tiles must match in color along shared edges and corners to corroborate the 0s and 1s their shared edges represent in $[A]$. This intuitive domino-like structure is vital to next section's proof that the Clockwise and Counterclockwise arrays are the only 4×4 De Bruijn arrays of 2×2 0-1 sub-matrices.

This has been proven before[2], and it is not difficult for a computer to prove by exhaustively checking each of the 2^{16} possible 4×4 binary matrices for the De Bruijn property. Still, the laborious proof of Theorem 2 will demonstrate the complexity which limits the construction of De Bruijn arrays.

3 Uniqueness of the Clockwise Array

Theorem 2. *The Clockwise array and its transposition the Counterclockwise array are the only 4×4 De Bruijn arrays of 2×2 0-1 sub-matrices.*

Proof. For contradiction, suppose a 4×4 De Bruijn array of 2×2 0-1 sub-matrices exists which is *not* the Clockwise or Counterclockwise array. This new toroidal array contains $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ by Definition 4, so represent the array with $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ in its center without any loss of generality. Convert to hexadecimal tiling, using ‘-’ for unknown entries.

$$\begin{bmatrix} - & - & - & - \\ - & 1 & 1 & - \\ - & 1 & 1 & - \\ - & - & - & - \end{bmatrix} \xrightarrow[\text{map hex}]{} \begin{bmatrix} - & - & - & - \\ - & \square & - & - \\ - & - & - & - \\ - & - & - & - \end{bmatrix}$$

Figure 9: A new De Bruijn array for contradiction

The hexadecimal tile \blacksquare cannot be directly beside \square in figure 9, nor diagonally adjacent to it, because it does not have a white edge or corner. This leaves seven spaces available for \blacksquare , marked with x in figure 10 below.

$$\begin{bmatrix} - & - & - & x \\ - & \square & - & x \\ - & - & - & x \\ x & x & x & x \end{bmatrix}$$

Figure 10: Possible positions for the 2×2 zero sub-matrix

Remark 1. *Considering symmetry, there are only three unique cases for \blacksquare 's placement: Cases A, B, and C.*

$$\begin{bmatrix} - & - & - & - \\ - & \square & - & \blacksquare \\ - & - & - & - \\ - & - & - & - \end{bmatrix}, \begin{bmatrix} - & - & - & - \\ - & \square & - & - \\ - & - & - & \blacksquare \\ - & - & - & - \end{bmatrix}, \begin{bmatrix} - & - & - & - \\ - & - & - & - \\ - & \square & - & - \\ - & - & - & \blacksquare \end{bmatrix}$$

Figure 11: Case A, Case B, and Case C, respectively

Given a proof that Case A or Case B cannot lead to a new De Bruijn array, we may reflect the steps of the proof over horizontal or diagonal axes as depicted in figure 12 on the next page. Therefore all possibilities for \blacksquare 's placement are accounted for in only three cases.

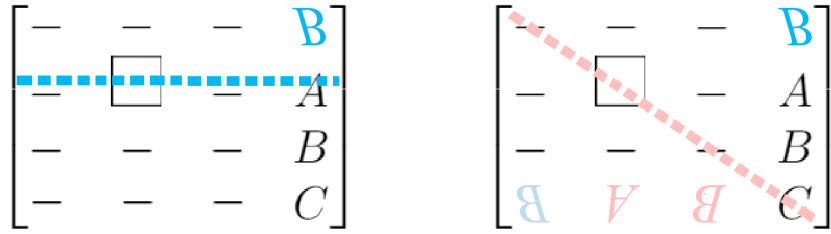


Figure 12: All seven x marks in figure 10 are accounted for in only three cases

Let us begin straightaway with Case A. Placing \blacksquare in this position necessitates placing \blacksquare and \blacksquare , the only tiles with white on one side and black on the other.

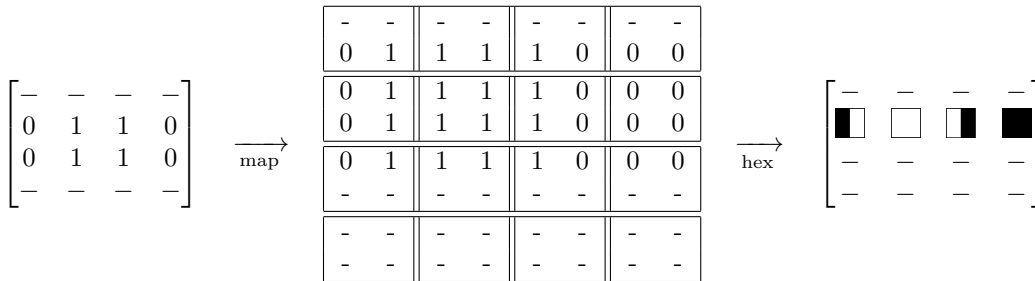


Figure 13: Case A

Only \blacksquare , \blacksquare , or \blacksquare could be directly above \square . Only \blacksquare , \blacksquare , or \blacksquare could be directly below \square . With three possibilities above and below, there should be nine more sub-cases to test. Luckily \blacksquare , \blacksquare , \blacksquare , \blacksquare , \blacksquare , and \blacksquare are reflections of one another, so we may reduce this to four sub-cases: \blacksquare and \blacksquare ; \blacksquare and \blacksquare ; \blacksquare and \blacksquare ; or \blacksquare and \blacksquare .

We may not use \blacksquare above and \blacksquare below \square , because their combination would produce another \blacksquare and this De Bruijn array must contain only one of each tile. In figure 14 below, the contradiction is highlighted in red.

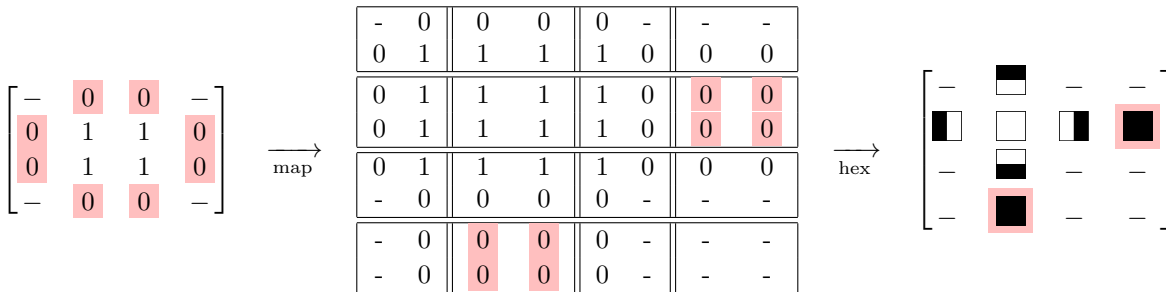


Figure 14: Case A, Sub-case 1 of 4

Similarly, we may not use \blacksquare above and \blacksquare below \square , as this produces an extra \blacksquare . The contradiction is highlighted in figure 15 below.

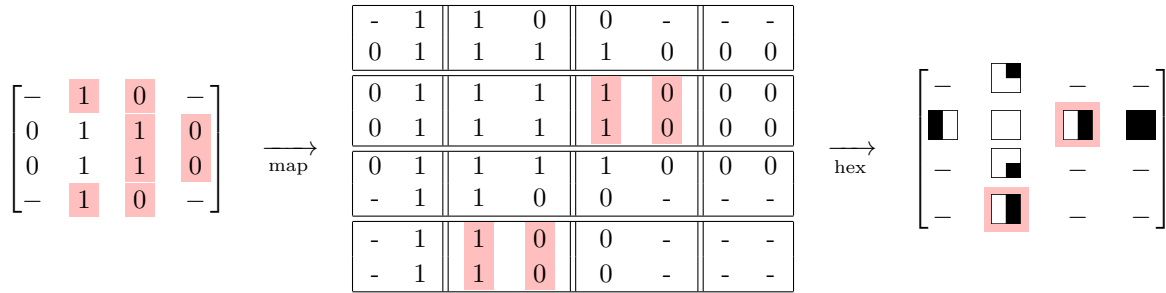


Figure 15: Case A, Sub-case 2 of 4

\blacksquare above and \blacksquare below \square produces a more complicated contradiction.

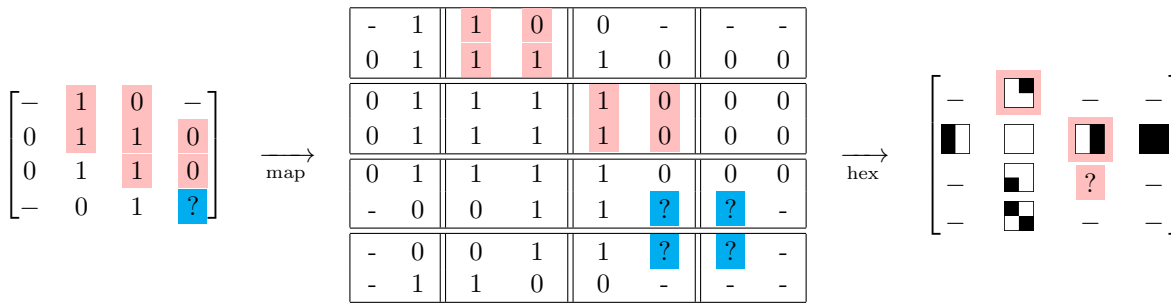


Figure 16: Case A, Sub-case 3 of 4

If the blue question mark in the leftmost matrix of figure 16 is 0, then the red question mark in the hexadecimal tiling will be a copy of \blacksquare . If the blue question mark is 1, then the red question mark will be a copy of \square . Either of these two possibilities would be a contradiction, as both those tiles are already present and highlighted red.

The final sub-case is \blacksquare above and \blacksquare below \square , pictured in figure 17 below.

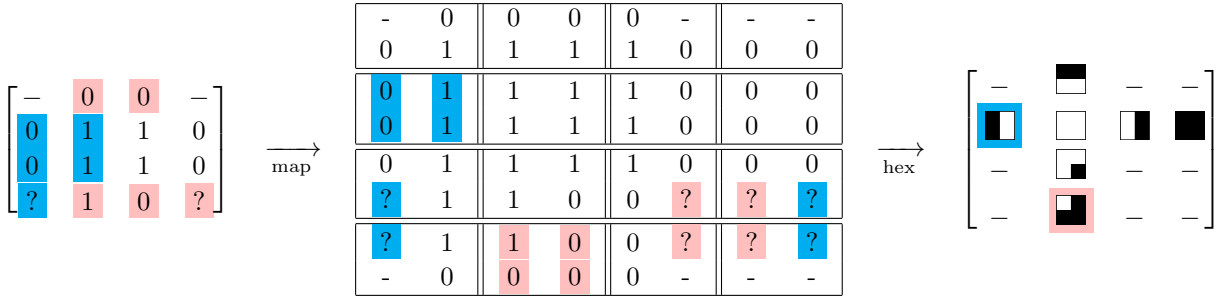


Figure 17: Case A, Sub-case 4 of 4, Step 1

Because \blacksquare is already present in figure 17 (highlighted blue), the blue question mark in the leftmost matrix must be 1 instead of 0. Because \blacksquare is already present in figure 17 (highlighted red), the red question mark in the leftmost matrix must be 1 instead of 0. These pigeonholed values create a duplicate copy of \blacksquare highlighted red in figure 18 below.

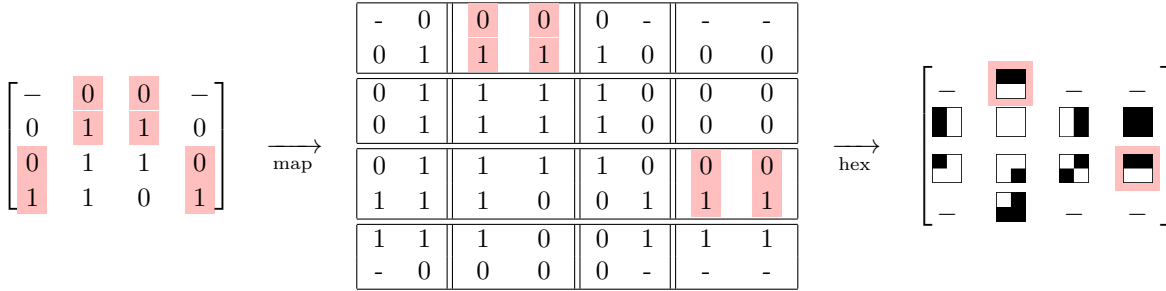


Figure 18: Case A, Sub-case 4 of 4, Step 2

Because the pigeonholed values create a second copy of \blacksquare , this cannot lead to a De Bruijn array. Therefore, after carefully culling sub-cases by symmetry, Case A has been eliminated and we move to Case B on the following page.

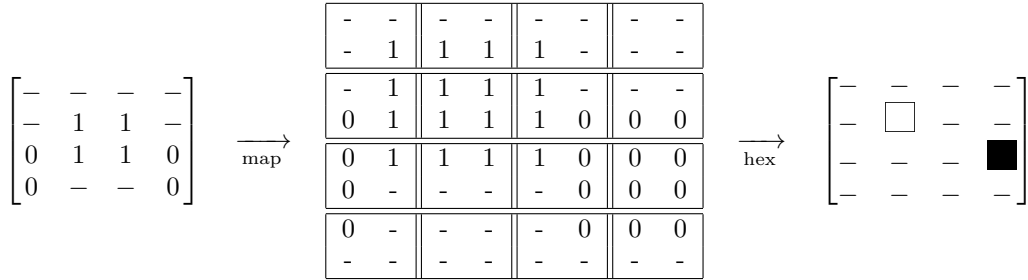


Figure 19: Case B

Only three 2×2 sub-matrices could fit underneath \square : \blacksquare or \square , which are reflections of one another, or $\begin{bmatrix} \blacksquare & \square \\ \square & \blacksquare \end{bmatrix}$. The contradiction in the $\begin{bmatrix} \blacksquare & \square \\ \square & \blacksquare \end{bmatrix}$ case is immediate.

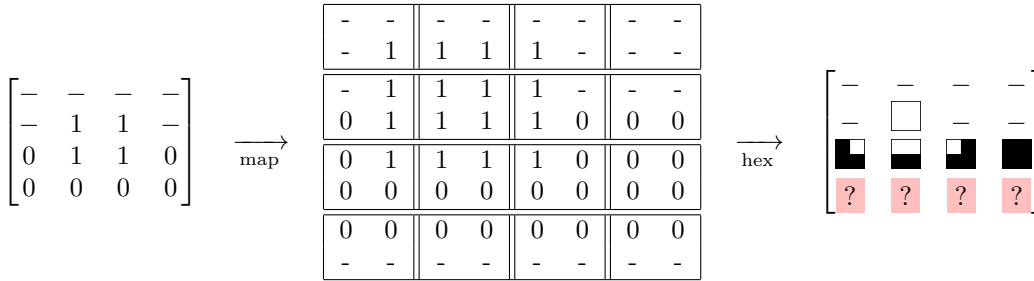


Figure 20: Case B, Sub-case 1 of 2

The bottom row of the leftmost matrix in figure 20 is entirely 0. Therefore the red question mark tiles in the hexadecimal tiling must be an ordering of $\begin{bmatrix} \blacksquare & \square \\ \square & \blacksquare \end{bmatrix}$, $\begin{bmatrix} \blacksquare & \blacksquare \\ \square & \square \end{bmatrix}$, and $\begin{bmatrix} \square & \square \\ \square & \blacksquare \end{bmatrix}$ or else one of those tiles would be repeated, and the array would fail Definition 4. Because $\begin{bmatrix} \blacksquare & \square \\ \square & \blacksquare \end{bmatrix}$ has already been placed in figure 20, this forced repetition of $\begin{bmatrix} \blacksquare & \square \\ \square & \blacksquare \end{bmatrix}$ is a contradiction.

Now we test the $\begin{bmatrix} \square & \square \\ \square & \blacksquare \end{bmatrix}$ sub-case (which symmetrically accounts for $\begin{bmatrix} \square & \square \\ \square & \blacksquare \end{bmatrix}$, as well).

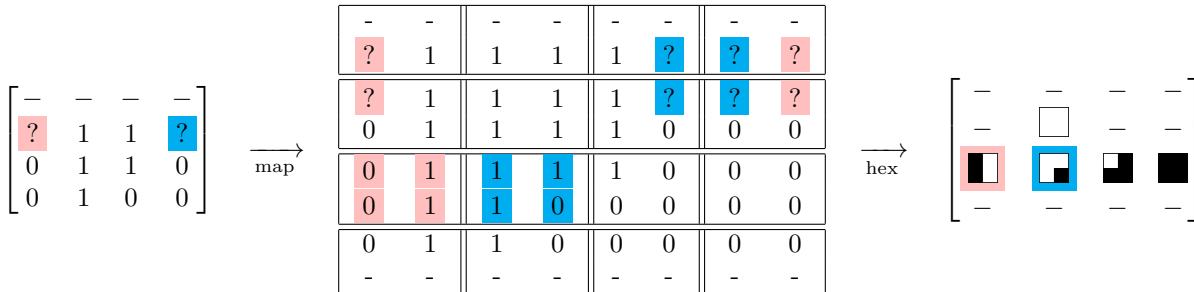


Figure 21: Case B, Sub-case 2 of 2, Step 1

Because \blacksquare is already present in figure 21 (highlighted red), the red question mark in the leftmost matrix must be 1 instead of 0. Because \blacksquare is already present in figure 21 (highlighted blue), the blue question mark in the leftmost matrix must be 0 instead of 1.

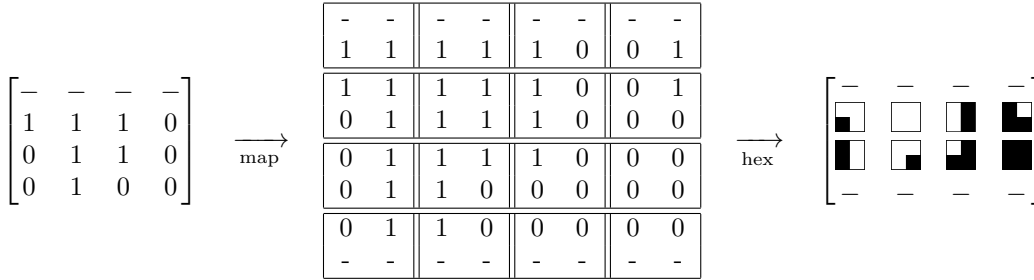


Figure 22: Case B, Sub-case 2 of 2, Step 2

Each unfinished box in the sub-matrix map of figure 22 has a 1 in the bottom row or a 0 in the top row. Therefore the tile \blacksquare , which is absent in figure 22, cannot be correctly placed anywhere, so this cannot lead to a De Bruijn array. Thus, Case *B* has been eliminated and we begin Case *C*.

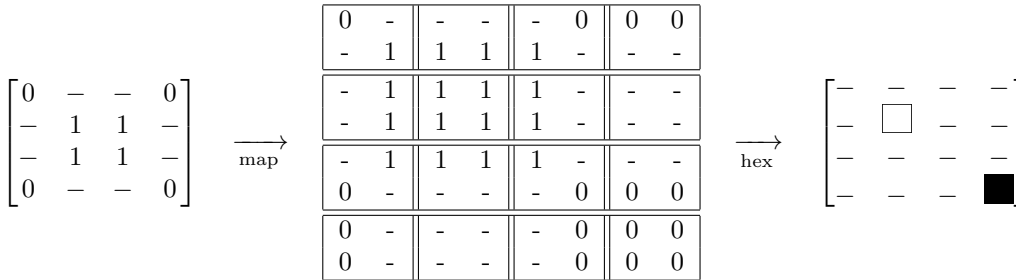


Figure 23: Case C

Just as in case *A* (see figure 13), there are four possibilities for the sub-matrices above and below \square , accounting for symmetry: \blacksquare and \blacksquare , \square and \square , \square and \square , or \square and \square . Using \blacksquare and \blacksquare results in immediate contradiction as we see in figure 24 on the following page.

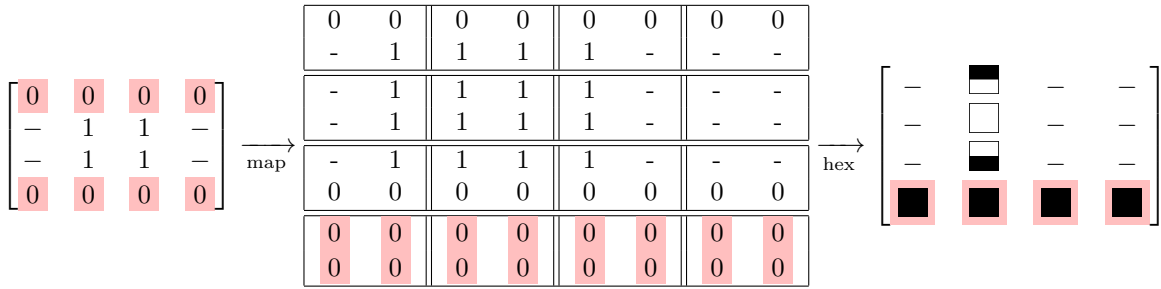


Figure 24: Case C, Sub-case 1 of 4

Figure 24 contains multiple copies of \blacksquare . A similar fault occurs with \square and \square .

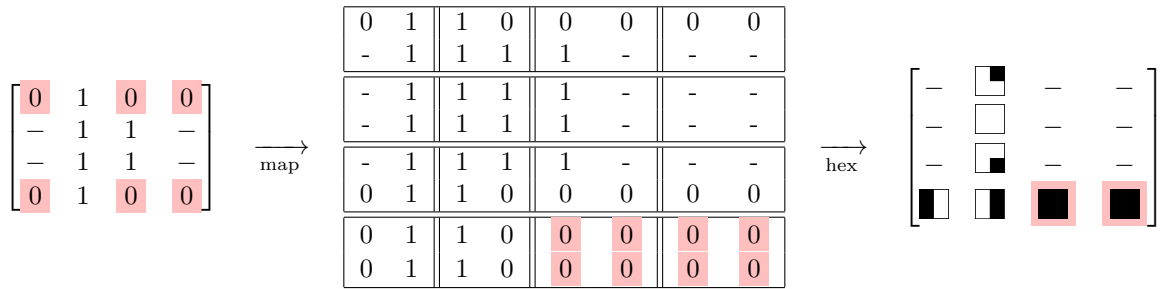


Figure 25: Case C, Sub-case 2 of 4

A similar contradiction occurs again with \blacksquare and \square .

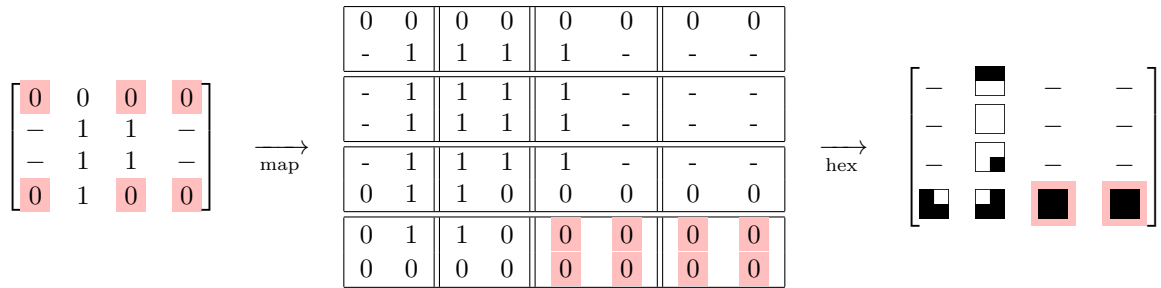


Figure 26: Case C, Sub-case 3 of 4

The only remaining case, \square and \square , will require a Lemma. We begin on the following page.

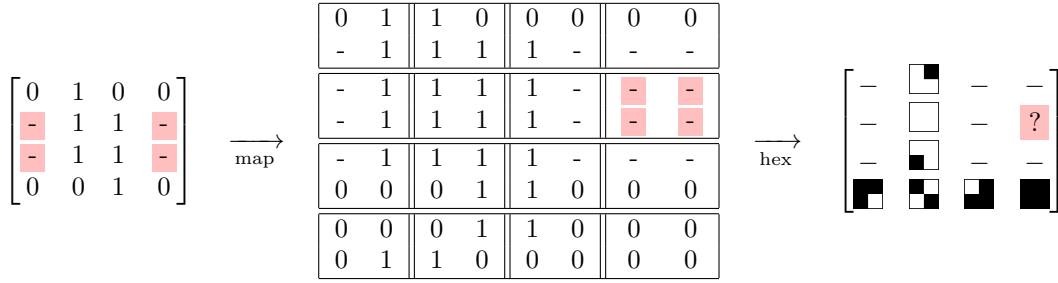


Figure 27: Case C, Sub-case 4 of 4, Step 1

The red question mark in the hexadecimal tiling of figure 27 may not repeat \square , \blacksquare , \square , \square , \square , or \blacksquare . It may not be \square , because that forms the Clockwise array as seen below in figure 28 below, contradicting our initial assumption.

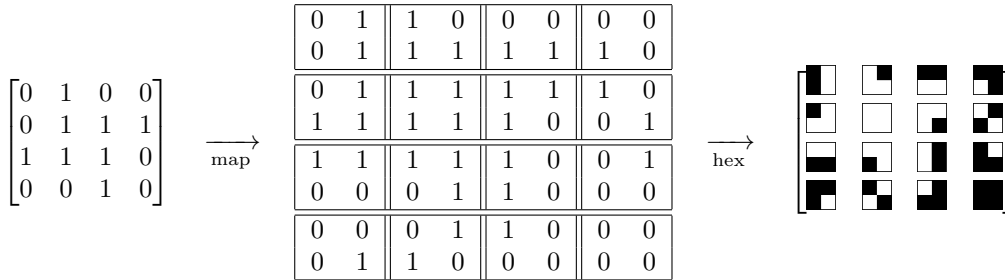


Figure 28: Case C, Sub-case 4, Step 2

A Lemma further removes \square , \square , \square , and \blacksquare from consideration.

Lemma 1. 4×4 De Bruijn arrays of 2×2 0-1 sub-matrices contain eight 0s.

Proof. Such a De Bruijn array $[A]$ contains each hexadecimal tile precisely once. \square represents no 0s in $[A]$'s sub-matrix map. \square , \square , \square , and \square represent one 0 each. \square , \square , \square , \square , \square , and \square represent two 0s each. \square , \square , \square , and \blacksquare represent three 0s each. \blacksquare represents four 0s in $[A]$'s sub-matrix map.

$(1 \times 0) + (4 \times 1) + (6 \times 2) + (4 \times 3) + (1 \times 4) = 32$ zeroes. $[A]$'s sub-matrix map features four times as many 0s as $[A]$ by construction, so $[A]$ contains $\frac{32}{4} =$ eight 0s. The remaining eight spaces are 1s. \square

The leftmost matrix in figure 27 includes six 0s and 1s. Lemma 1 implies the red question mark in its hexadecimal tiling must therefore represent two 0s and two 1s; it must be one of \square , \square , \square , or \blacksquare .

Replacing the red question mark with \blacksquare creates several contradictions, one of which is highlighted in figure 29. (Using \blacksquare produces the same array, rotated.)

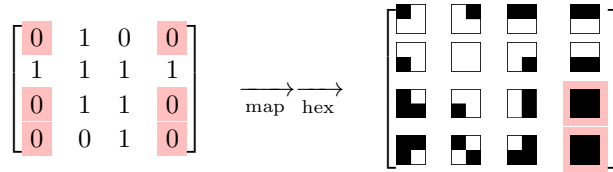


Figure 29: Case C, Sub-case 4, Step 3

Replacing the red question mark with \blacksquare creates another contradiction. (Using \blacksquare produces the same array, rotated.)

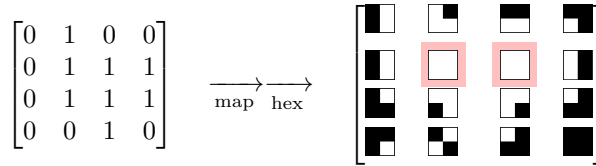


Figure 30: Case C, Sub-case 4, Step 4

Thus, Case *C* only yields a De Bruijn array if it's the Clockwise or Counterclockwise array, and the proof by contradiction is complete. Theorem 2 is proven. \square

In the first half of this capstone project, we have proven 4×4 De Bruijn arrays of 2×2 0-1 sub-matrices exist, and the uniqueness of the Clockwise and Counterclockwise arrays. This and more is already known[4].

In the second half, we will define and construct a $16 \times 4 \times 4$ De Bruijn array of $2 \times 2 \times 2$ 0-1 sub-arrays. We will provide a lower bound for the quantity of these objects, and inductively extend the construction to produce analogues to the De Bruijn sequence in higher dimensions. We will also speculate at potential applications for high-dimension De Bruijn arrays.

4 Transitioning to Three Dimensions

4.1 Notation for 3D Arrays

In this paper 3D arrays are presented as cross-sectional 2D matrices listed from ‘front’ to ‘back’ separated by crosses. For example, the $2 \times 2 \times 2$ array with front layer $\begin{bmatrix} \blacksquare & \square \\ \square & \square \end{bmatrix}$ and back layer $\begin{bmatrix} \square & \square \\ \square & \square \end{bmatrix}$ will be written as

$$\left(\begin{bmatrix} \blacksquare & \square \\ \square & \square \end{bmatrix} \times \begin{bmatrix} \square & \square \\ \square & \square \end{bmatrix} \right), \text{ or, equivalently, } \left(\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \right).$$

If built out of black and white blocks (black for 0 and white for 1), $\left(\begin{bmatrix} \blacksquare & \square \\ \square & \square \end{bmatrix} \times \begin{bmatrix} \square & \square \\ \square & \square \end{bmatrix} \right)$ would appear as in figure 31 below.

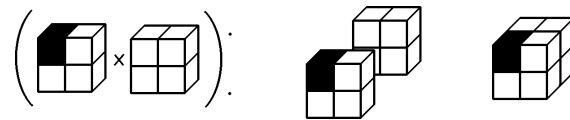


Figure 31: A 3D array expressed as layers

In discussing the position of $2 \times 2 \times 2$ sub-arrays in 3D arrays, it will be helpful to have less burdensome terminology.

Definition 5. A *b-cube* is a $2 \times 2 \times 2$ array of binary elements, 0s and 1s.

Definition 6. The front, top, left element of a *b-cube* is its *Handle*. The position of a *b-cube* in a 3D array is stated as the position of its handle in the format (Layers back, Rows down, Columns right). As stated in Section 1.2, array indexes begin at zero.

For example, the handle of the *b-cube* $\left(\begin{bmatrix} \blacksquare & \square \\ \square & \square \end{bmatrix} \times \begin{bmatrix} \square & \square \\ \square & \square \end{bmatrix} \right)$ is its 0, the black corner.

Discussion of De Bruijn arrays in three dimensions requires the notion of a 3D array shaped like a hyper-torus embedded in 4D space. The following definitions are direct extensions of those presented in sections 2.2 and 2.3 on page 6.

4.2 The Hyper-Toroidal Array

Definition 7. An $m \times n \times p$ *Hyper-toroidal Array* is an equivalence class of $m \times n \times p$ arrays under the relation $A \approx B$ if A and B are both $m \times n \times p$ arrays and there exist non-negative integers x, y , and z with $x < m$, $y < n$, $z < p$ such that for all entries of A and B , $A_{i,j,k} = B_{d,e,f}$ with $d = i + x \bmod n$, $d = j + y \bmod m$, and $f = k + z \bmod p$.

Definition 8. An $m \times n \times p$ hyper-toroidal array $[A]$ contains a $d \times e \times f$ array B (with $d \leq m$, $e \leq n$, and $f \leq p$) if there exists an array $A \in [A]$ so

$$A_{i,j,k} = B_{i,j,k}.$$

$$\begin{array}{l} 0 \leq i < d, \\ 0 \leq j < e, \\ 0 \leq k < f \end{array}$$

If A is the only array with this trait in $[A]$, then $[A]$ contains B *precisely once*.

With these definitions, we may explore De Bruijn arrays in three dimensions.

4.3 Defining De Bruijn Arrays in Three Dimensions

Definition 9. A $16 \times 4 \times 4$ *De Bruijn array of b -cubes* is a $16 \times 4 \times 4$ hyper-toroidal array which contains each b -cube precisely once.

The position of a b -cube's handle in a De Bruijn array $[A]$ should be given with respect to a stationary reference point so coordinates refer to the same location for any representative $A \in [A]$. The following definition provides such a reference point.

Definition 10. The *Origin* of a $16 \times 4 \times 4$ De Bruijn array of b -cubes is the Handle of $(\blacksquare \times \blacksquare)$, which Definition 12 guarantees is contained precisely once. The origin defines the 0^{th} layer, row, and column of the hyper-toroidal array.

Then, when a b -cube is said to be in position (x, y, z) of a $16 \times 4 \times 4$ De Bruijn array of b -cubes, we understand that the b -cube's handle is x layers behind, y rows beneath, and z columns right of the handle of $(\blacksquare \times \blacksquare)$. x, y , and z should always be non-negative integers with $x < 16$ and $y, z < 4$.

These are the tools necessary to understand section 5.

5 Constructing 3D De Bruijn Arrays

This section outlines a construction method, proves objects created with that method satisfy Definition 9, presents such an object, and enumerates its b -cubes to prove the existence of $16 \times 4 \times 4$ De Bruijn arrays of b -cubes.

5.1 A Layer-By-Layer Construction

Suppose $[F]$ is a $16 \times 4 \times 4$ hyper-toroidal array whose elements are yet unassigned. Let $F \in [F]$. For $0 \leq i \leq 15$ let the i^{th} layer of F be denoted F_i .

Over the same range let x_i and y_i be integers in the set $\{0, 1, 2, 3\}$ such that if $i \neq j$, then at least one of $x_i \neq x_j$ or $y_i \neq y_j$. Realize the set of (x_i, y_i) pairs is an ordering of $(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), \dots, (3, 3)$. Without loss of generality because of $[F]$'s hyper-toroidal shape, reorder and relabel so $(x_0, y_0) = (0, 0)$.

Let F_0 be the representative of the Clockwise array (or Counterclockwise array, symmetrically) with \blacksquare in its top, left corner. Let $F_{(1,j,k)} = F_{(0,m,n)}$ with $m = j - x_0 \bmod 4$ and $n = k - y_0 \bmod 4$. $(x_0, y_0) = (0, 0)$, so $F_1 = F_0$.

$$\left(\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \right)$$

Figure 32: $(F_0 \times F_1)$











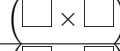





b -cube	Handle in $[F]$	b -cube	Handle	b -cube	Handle	b -cube	Handle
	$(0, 0, 0)$		$(0, 0, 1)$		$(0, 0, 2)$		$(0, 0, 3)$
	$(0, 1, 0)$		$(0, 1, 1)$		$(0, 1, 2)$		$(0, 1, 3)$
	$(0, 2, 0)$		$(0, 2, 1)$		$(0, 2, 2)$		$(0, 2, 3)$
	$(0, 3, 0)$		$(0, 3, 1)$		$(0, 3, 2)$		$(0, 3, 3)$

Figure 33: b -cubes contained in $(F_0 \times F_1)$ as layers of $F \in [F]$

For subsequent layers of F , let $F_{(i+1,j,k)} = F_{(i,m,n)}$ with $m = j - x_i \bmod 4$ and $n = k - y_i \bmod 4$. This ensures every layer of F is congruent to the Clockwise array, but each pair of adjacent layers relates to a different pair (x_i, y_i) . The construction of F is complete. The next section proves $[F]$ satisfies definition 9.

(x_{15}, y_{15}) is not used to generate a layer of F . Rather, Lemma 2 on the following page shows $F_{(0,j,k)} = F_{(15,m,n)}$ with $m = j - x_{15} \bmod 4$ and $n = k - y_{15} \bmod 4$.

5.2 Proving $[F]$ is a $16 \times 4 \times 4$ De Bruijn Array of b -cubes

Lemma 2 demonstrates the relationship between layer 0 and layer 15 of $F \in [F]$.

Lemma 2. $F_{(0,j,k)} = F_{(15,m,n)}$ with $m = j - x_{15} \pmod 4$ and $n = k - y_{15} \pmod 4$.

Proof. Since

$$F_{(1,j,k)} = F_{(0,m,n)} \text{ with } m = j - x_0 \pmod 4 \text{ and } n = k - y_0 \pmod 4, \text{ and}$$

$$F_{(2,j,k)} = F_{(1,m,n)} \text{ with } m = j - x_1 \pmod 4 \text{ and } n = k - y_1 \pmod 4,$$

it follows that

$$F_{(2,j,k)} = F_{(0,m,n)} \text{ with } m = j - (x_0 + x_1) \pmod 4, n = k - (y_0 + y_1) \pmod 4.$$

Extending this relation yields

$$(*) \quad F_{(15,j,k)} = F_{(0,m,n)} \text{ with } m = j - \sum_{i=0}^{14} x_i \pmod 4 \text{ and } n = k - \sum_{i=0}^{14} y_i \pmod 4.$$

Recall the set of all (x_i, y_i) pairs is an ordering of $(0, 0), (0, 1), (0, 2), \dots, (3, 3)$, so the sum $\sum_{i=0}^{15} x_i$ equals $4(0) + 4(1) + 4(2) + 4(3) = 24$, which is congruent to $0 \pmod 4$. Thus $\sum_{i=0}^{14} x_i = -x_{15} \pmod 4$, and similarly for y_{15} . So, $(*)$ implies $F_{(0,j,k)} = F_{(15,m,n)}$ with $m = j - x_{15} \pmod 4$ and $n = k - y_{15} \pmod 4$. \square

Now that the relationship between any two adjacent layers of $[F]$ is guaranteed to be unique, the following Lemmas assess the uniqueness of the b -cubes they contain.

Lemma 3. *The sixteen b -cubes contained in $(F_i \times F_{i+1 \pmod{16}})$ are distinct for all $0 \leq i \leq 15$.*

Proof. Certainly sixteen b -cubes are contained, because each of the sixteen elements of the layer F_i is the handle of one b -cube (including those b -cubes ‘split’ by an edge or corner as F is a representation of hyper-toroidal array $[F]$).

For contradiction assume that for some $0 \leq i \leq 15$, $(F_i \times F_{i+1 \pmod{16}})$ contains two copies of the b -cube $(\alpha \times \beta)$, α and β being hexadecimal tiles. Then F_i contains two copies of α , the front half of $(\alpha \times \beta)$ —but F_i is congruent to the Clockwise array, a 4×4 De Bruijn array of 2×2 0-1 sub-matrices. Containing two copies of α would be a contradiction. The same goes for $F_{i+1 \pmod{16}}$ and β . Hence, $(F_i \times F_{i+1 \pmod{16}})$ cannot contain two copies of the same b -cube. \square

Lemma 4. For integers i and j with $0 \leq i \neq j \leq 15$, $(F_i \times F_{i+1 \bmod 16})$ and $(F_j \times F_{j+1 \bmod 16})$ share none of the b -cubes they contain.

Proof. Suppose $(F_i \times F_{i+1 \bmod 16})$ contains the b -cube $(\alpha \times \beta)$. Since F_j and $F_{j+1 \bmod 16}$ are congruent to the Clockwise array, F_j contains α precisely once and $F_{j+1 \bmod 16}$ contains β precisely once.

However, since $i \neq j$, Lemma 2 and the construction of F imply

$$F_{(i+1 \bmod 16, h, k)} = F_{(i, m, n)} \text{ with } m = h - x_i \bmod 4 \text{ and } n = k - y_i \bmod 4,$$

while

$$F_{(j+1 \bmod 16, h, k)} = F_{(j, m, n)} \text{ with } m = h - x_j \bmod 4 \text{ and } n = k - y_j \bmod 4.$$

As $i \neq j$ implies at least one of $x_i \neq x_j$ or $y_i \neq y_j$, the alignment of α and β in $(F_i \times F_{i+1 \bmod 16})$ *prohibits* their alignment in $(F_j \times F_{j+1 \bmod 16})$. The two distinct pairs of layers can share no b -cubes. \square

For example, let $(x_1, y_1) = (0, 1)$. This one-space offset from F_1 to F_2 ensures the 16 b -cubes in $(F_1 \times F_2)$ are different than the 16 b -cubes in $(F_0 \times F_1)$. Compare figures 34 and 35 with figures 32 and 33.

$$\left(\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \right)$$

Figure 34: $(F_0 \times F_1 \times F_2)$

b -cube	Handle in F	b -cube	Handle	b -cube	Handle	b -cube	Handle
	(1, 0, 0)		(1, 0, 1)		(1, 0, 2)		(1, 0, 3)
	(1, 1, 0)		(1, 1, 1)		(1, 1, 2)		(1, 1, 3)
	(1, 2, 0)		(1, 2, 1)		(1, 2, 2)		(1, 2, 3)
	(1, 3, 0)		(1, 3, 1)		(1, 3, 2)		(1, 3, 3)

Figure 35: b -cubes contained in $(F_1 \times F_2)$ as layers of $F \in [F]$

Theorem 3. $[F]$ satisfies definition 9.

Proof. $[F]$ is a $16 \times 4 \times 4$ hyper-toroidal array, so it has the required dimensions and shape.

Lemmas 3 and 4 guarantee 16 distinct b -cubes in each of 16 pairs of layers, so there are 256 distinct b -cubes contained in $[F]$. There are exactly $2^8 = 256$ possible b -cubes, so $[F]$ must contain each b -cube precisely once. Thus, $[F]$ satisfies Definition 9. \square

This construction began with the choice of assigning F_0 to be Clockwise or Counterclockwise. Then we permuted the set of sixteen possible pairs of elements from $\{0, 1, 2, 3\}$ with $(0, 0)$ first without loss of generality due to $[F]$'s hyper-toroidal shape. Hence, there are at least

$$2 \times 15! \approx 2.614 \times 10^{12}$$

$16 \times 4 \times 4$ De Bruijn arrays of b -cubes. Perhaps more examples could be generated using different construction techniques which did not rely on the Clockwise or Counterclockwise shape (see section 5.3).

An example of an array built with this construction is written with $\blacksquare = 0$ and $\square = 1$ on the next page in figure 36.

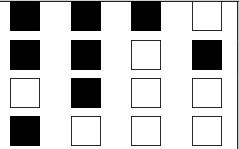
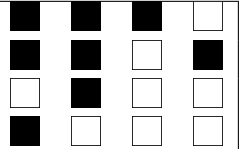
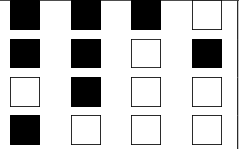
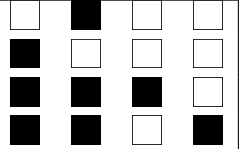
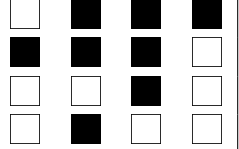
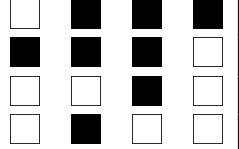
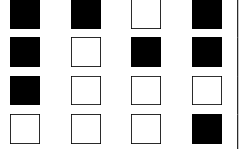
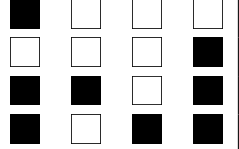
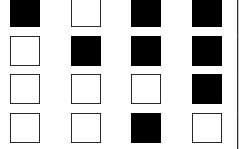
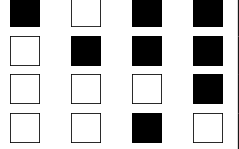
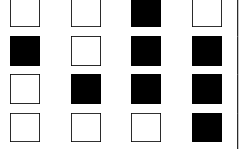
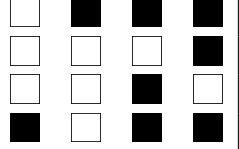
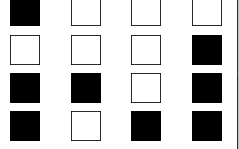
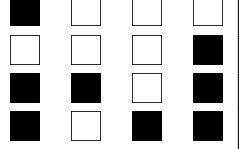
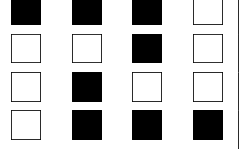
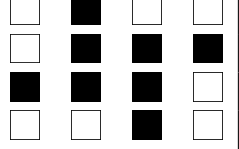
Layer 0			Layer 8
			Offset (1,3)
Layer 1			Layer 9
Offset (0,0)			Offset (2,0)
Layer 2			Layer 10
Offset (0,1)			Offset (2,1)
Layer 3			Layer 11
Offset (0,2)			Offset (2,2)
Layer 4			Layer 12
Offset (0,3)			Offset (2,3)
Layer 5			Layer 13
Offset (1,0)			Offset (3,0)
Layer 6			Layer 14
Offset (1,1)			Offset (3,1)
Layer 7			Layer 15
Offset (1,2)			Offset (3,2)

Figure 36: A $16 \times 4 \times 4$ De Bruijn array of b -cubes
Note Layer 0 is Layer 15 offset by 3 rows and 3 columns, as stated in Lemma 2

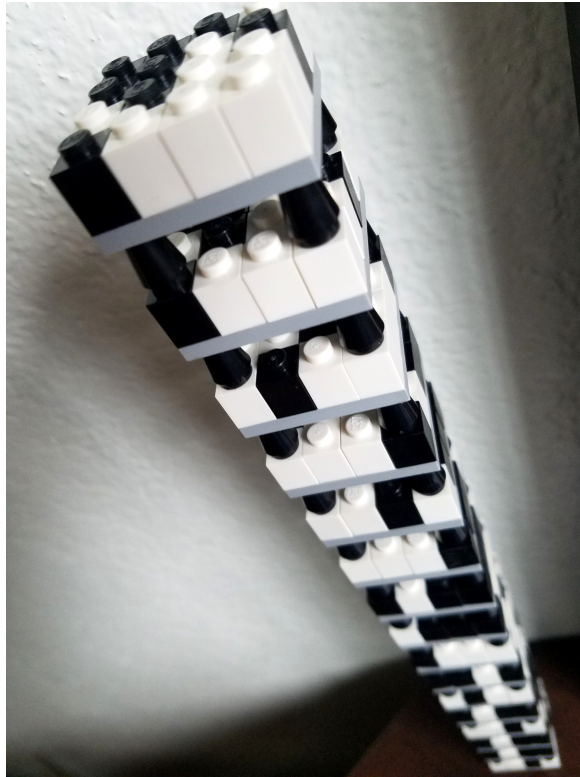


Figure 37: The same array in Lego Brick

To show every b -cube is contained precisely once in the array represented in figures 36 and 37, below find an exhaustive list of all 256 b -cubes by their handle.








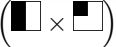
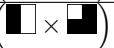
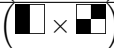

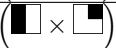
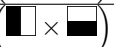

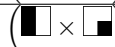
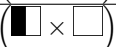
b -cube	Handle	b -cube	Handle	b -cube	Handle	b -cube	Handle
	(0, 0, 0)		(3, 0, 3)		(12, 0, 2)		(13, 3, 2)
	(4, 0, 2)		(15, 1, 1)		(2, 0, 1)		(11, 2, 3)
	(1, 0, 0)		(8, 0, 0)		(5, 1, 2)		(14, 2, 3)
	(7, 3, 1)		(6, 2, 3)		(9, 2, 0)		(10, 0, 1)



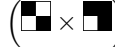
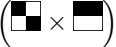





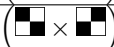

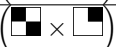

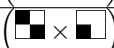

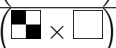
b -cube	Handle	b -cube	Handle	b -cube	Handle	b -cube	Handle
	(1, 0, 1)		(0, 0, 1)		(13, 3, 3)		(14, 2, 0)
	(5, 1, 3)		(12, 0, 3)		(3, 0, 0)		(8, 0, 1)
	(2, 0, 2)		(9, 2, 1)		(6, 2, 0)		(15, 1, 2)
	(4, 0, 3)		(7, 3, 2)		(10, 0, 2)		(11, 2, 0)



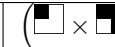
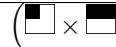
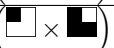
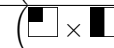
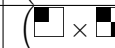
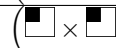
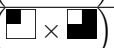

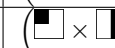
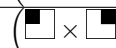
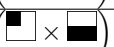


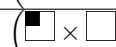
b -cube	Handle	b -cube	Handle	b -cube	Handle	b -cube	Handle
	(4, 1, 2)		(7, 0, 1)		(0, 1, 0)		(1, 1, 0)
	(8, 1, 10)		(3, 1, 3)		(6, 3, 3)		(15, 2, 1)
	(5, 2, 2)		(12, 1, 2)		(9, 3, 0)		(2, 1, 1)
	(11, 3, 3)		(10, 1, 1)		(13, 0, 2)		(14, 3, 3)

















b -cube	Handle	b -cube	Handle	b -cube	Handle	b -cube	Handle
	(7, 0, 0)		(6, 3, 2)		(3, 1, 2)		(0, 1, 3)
	(11, 3, 2)		(2, 1, 0)		(5, 2, 1)		(14, 3, 2)
	(4, 1, 1)		(15, 2, 0)		(8, 1, 3)		(1, 1, 3)
	(10, 1, 0)		(9, 3, 3)		(12, 1, 1)		(13, 0, 1)

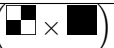
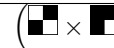

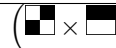
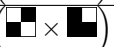
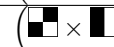
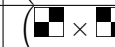
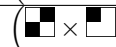

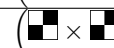

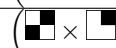



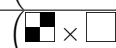
b -cube	Handle	b -cube	Handle	b -cube	Handle	b -cube	Handle
	(12, 3, 2)		(15, 0, 1)		(8, 3, 0)		(9, 1, 0)
	(0, 3, 0)		(11, 1, 3)		(14, 1, 3)		(7, 2, 1)
	(13, 2, 2)		(4, 3, 2)		(1, 3, 0)		(10, 3, 1)
	(3, 3, 3)		(2, 3, 1)		(5, 0, 2)		(6, 1, 3)




























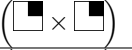























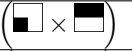











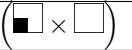





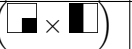

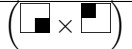
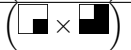


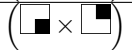
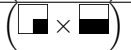
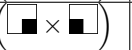


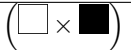


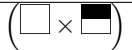
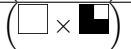
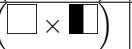

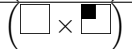

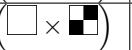

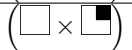



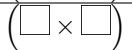
<i>b</i> -cube	Handle	<i>b</i> -cube	Handle	<i>b</i> -cube	Handle	<i>b</i> -cube	Handle
	(5, 2, 3)		(4, 1, 3)		(1, 1, 1)		(2, 1, 2)
	(9, 3, 1)		(0, 1, 1)		(7, 0, 2)		(12, 1, 3)
	(6, 3, 0)		(13, 0, 3)		(10, 1, 2)		(3, 1, 0)
	(8, 1, 1)		(11, 3, 0)		(14, 3, 0)		(15, 2, 2)

<i>b</i> -cube	Handle	<i>b</i> -cube	Handle	<i>b</i> -cube	Handle	<i>b</i> -cube	Handle
	(2, 0, 3)		(1, 0, 2)		(14, 2, 1)		(15, 1, 3)
	(6, 2, 1)		(13, 3, 0)		(0, 0, 2)		(9, 2, 2)
	(3, 0, 1)		(10, 0, 3)		(7, 3, 3)		(12, 0, 0)
	(5, 1, 0)		(4, 0, 0)		(11, 2, 1)		(8, 0, 2)

<i>b</i> -cube	Handle	<i>b</i> -cube	Handle	<i>b</i> -cube	Handle	<i>b</i> -cube	Handle
	(9, 0, 1)		(8, 2, 1)		(5, 3, 3)		(6, 0, 0)
	(13, 1, 3)		(4, 2, 3)		(11, 0, 0)		(0, 2, 1)
	(10, 2, 2)		(1, 2, 1)		(14, 0, 0)		(7, 1, 2)
	(12, 2, 3)		(15, 3, 2)		(2, 2, 2)		(3, 2, 0)

<i>b</i> -cube	Handle	<i>b</i> -cube	Handle	<i>b</i> -cube	Handle	<i>b</i> -cube	Handle
	(3, 0, 2)		(2, 0, 0)		(15, 1, 0)		(12, 0, 1)
	(7, 3, 0)		(14, 2, 2)		(1, 0, 3)		(10, 0, 0)
	(0, 0, 3)		(11, 2, 2)		(4, 0, 1)		(13, 3, 1)
	(6, 2, 2)		(5, 1, 1)		(8, 0, 3)		(9, 2, 3)

<i>b</i> -cube	Handle	<i>b</i> -cube	Handle	<i>b</i> -cube	Handle	<i>b</i> -cube	Handle
	(8, 2, 0)		(11, 0, 3)		(4, 2, 2)		(5, 3, 2)
	(12, 2, 2)		(7, 1, 1)		(10, 2, 1)		(3, 2, 3)
	(9, 0, 0)		(0, 2, 0)		(13, 1, 2)		(6, 0, 3)
	(15, 3, 1)		(14, 0, 3)		(1, 2, 0)		(2, 2, 1)

b -cube	Handle	b -cube	Handle	b -cube	Handle	b -cube	Handle
	(15, 0, 0)		(14, 1, 2)		(11, 1, 2)		(8, 3, 3)
	(3, 3, 2)		(10, 3, 0)		(13, 2, 1)		(6, 1, 2)
	(12, 3, 1)		(7, 2, 0)		(0, 3, 3)		(9, 1, 3)
	(2, 3, 0)		(1, 3, 3)		(4, 3, 1)		(5, 0, 1)
b -cube	Handle	b -cube	Handle	b -cube	Handle	b -cube	Handle
	(6, 3, 1)		(5, 2, 0)		(2, 1, 3)		(3, 1, 1)
	(10, 1, 3)		(1, 1, 2)		(4, 1, 0)		(13, 0, 0)
	(7, 0, 3)		(14, 3, 1)		(11, 3, 1)		(0, 1, 2)
	(9, 3, 2)		(8, 1, 2)		(15, 2, 3)		(12, 1, 0)
b -cube	Handle	b -cube	Handle	b -cube	Handle	b -cube	Handle
	(13, 2, 3)		(12, 3, 3)		(9, 1, 1)		(10, 3, 2)
	(1, 3, 1)		(8, 3, 1)		(15, 0, 2)		(4, 3, 3)
	(14, 1, 0)		(5, 0, 3)		(2, 3, 2)		(11, 1, 0)
	(0, 3, 1)		(3, 3, 0)		(6, 1, 0)		(7, 2, 2)
b -cube	Handle	b -cube	Handle	b -cube	Handle	b -cube	Handle
	(14, 1, 1)		(13, 2, 0)		(10, 3, 3)		(11, 1, 1)
	(2, 3, 3)		(9, 1, 2)		(12, 3, 0)		(5, 0, 0)
	(15, 0, 3)		(6, 1, 1)		(3, 3, 1)		(8, 3, 2)
	(1, 3, 2)		(0, 3, 2)		(7, 2, 3)		(4, 3, 0)
b -cube	Handle	b -cube	Handle	b -cube	Handle	b -cube	Handle
	(11, 0, 2)		(10, 2, 0)		(7, 1, 0)		(4, 2, 1)
	(15, 3, 0)		(6, 0, 2)		(9, 0, 3)		(2, 2, 0)
	(8, 2, 3)		(3, 2, 2)		(12, 2, 1)		(5, 3, 1)
	(14, 0, 2)		(13, 1, 1)		(0, 2, 3)		(1, 2, 3)
b -cube	Handle	b -cube	Handle	b -cube	Handle	b -cube	Handle
	(10, 2, 3)		(9, 0, 2)		(6, 0, 1)		(7, 1, 3)
	(14, 0, 1)		(5, 3, 0)		(8, 2, 2)		(1, 2, 2)
	(11, 0, 1)		(2, 2, 3)		(15, 3, 3)		(4, 2, 0)
	(13, 1, 0)		(12, 2, 0)		(3, 2, 1)		(0, 2, 2)

Four copies each of A , B , C , and D would contain the distribution of tiles needed to make a $16 \times 4 \times 4$ De Bruijn array of b -cubes. The apparent structure in their design inspires hope that this may be possible, though we have found no example.

An attempt to generate a $16 \times 4 \times 4$ De Bruijn array of b -cubes using A , B , C , and D might begin with the generation of a De Bruijn sequence of two-length four-sequences whose alphabet is $\{A, B, C, D\}$ rather than $\{0, 1, 2, 3\}$. ($AABBCCDDCBADB DAC$) is an example. Perhaps four copies each of A , B , C , and D could be compiled in that order with offsets in a manner similar to the construction outlined in section 5.1.

6 Higher Dimensions

With an example of a $3D$ De Bruijn array, we could extend the technique outlined in section 5.1 to generate a $4D$ De Bruijn array. We could then use that $4D$ De Bruijn array to generate a $5D$ De Bruijn array, and so on. Let us extend definitions from prior sections to n dimensions to facilitate a proof by induction.

Definition 11. In n dimensions, a *Hyper-Toroidal Array* is an equivalence class of $m_1 \times m_2 \times \cdots \times m_n$ arrays under the relation $A \approx B$ if A and B are both $m_1 \times m_2 \times \cdots \times m_n$ arrays and there exist integers $x_1, x_2, \dots, x_n \geq 0$ with $x_1 < m_1, x_2 < m_2, \dots$, and $x_n < m_n$ such that for all entries of A and B , $A_{a_1, a_2, \dots, a_n} = B_{b_1, b_2, \dots, b_n}$ with $b_i = a_i + x_i \pmod{m_i}$ for all i from 1 to n .

Definition 12. An $m_1 \times m_2 \times \cdots \times m_n$ hyper-toroidal array $[A]$ contains a $d_1 \times d_2 \times \cdots \times d_n$ array B (with $d_i \leq m_i$ for all i from 1 to n) if there exists an $A \in [A]$ so that $A_{i_j} = B_{i_j}$. In other words, B is in A 's former, front, top, left, etcetera corner.

$$\forall \substack{1 \leq j \leq n, \\ 0 \leq i < d_j} A_{i_j} = \forall \substack{1 \leq j \leq n, \\ 0 \leq i < d_j} B_{i_j}$$

If A uniquely satisfies this property in $[A]$, then $[A]$ contains B *precisely once*.

Definition 13. A $\left(\prod_{i=1}^{n-1} 2^{2^i}\right) \times 4$ *De Bruijn array of binary 2^n -cube sub-arrays* is a $2^{2^{n-1}} \times 2^{2^{n-2}} \times \cdots \times 2^{2^2} \times 2^{2^1} \times 4$ hyper-toroidal array containing precisely one of each $\underbrace{2 \times 2 \times \cdots \times 2}_n$ array of 0s and 1s.

Definition 14. The former, front, top, left, etcetera corner of a $\underbrace{2 \times 2 \times \cdots \times 2}_n$ array is its *Handle*.

Definition 15. The *Origin* of a $\left(\prod_{i=1}^{n-1} 2^{2^i}\right) \times 4$ *De Bruijn array of binary 2^n -cube sub-arrays* is the handle of the binary $\underbrace{2 \times 2 \times \cdots \times 2}_n$ array of only zeroes, which is contained precisely once by definition 13.

For $n = 1$ through 3, we know $\left(\prod_{i=1}^{n-1} 2^{2^i}\right) \times 4$ De Bruijn arrays of binary 2^n -cube sub-arrays exist.

If $n = 1$, $\left(\prod_{i=1}^{n-1} 2^{2^i}\right) \times 4 = 4$. The four-length De Bruijn sequence (0011), which contains as subsequences 00, 01, 10, and 11, is such a De Bruijn array.

If $n = 2$, $\left(\prod_{i=1}^{n-1} 2^{2^i}\right) \times 4 = 4 \times 4$. The Clockwise and Counterclockwise arrays, which contain the 16 binary 2×2 arrays, are such De Bruijn arrays.

If $n = 3$, $\left(\prod_{i=1}^{n-1} 2^{2^i}\right) \times 4 = 16 \times 4 \times 4$. The array in figure 36, which contains the 256 b -cubes, is such a De Bruijn array. Having produced these initial cases, move to the inductive step.

6.1 Constructing an $(n + 1)D$ De Bruijn Array

Presume $[N]$ is a $\left(\prod_{i=1}^{n-1} 2^{2^i}\right) \times 4$ De Bruijn array of binary 2^n -cube sub-arrays. Let $[M]$ be a $2^{2^n} \times 2^{2^{n-1}} \times \dots \times 2^{2^2} \times 2^{2^1} \times 4$, $n + 1^{th}$ dimensional hyper-toroidal array whose elements are yet unassigned, and let $M \in [M]$. For all integers i with $0 \leq i \leq 2^{2^n} - 1$, denote by M_i the i^{th} "Instance" in M .

Over the same index i define n -tuples $(x_{i,1}, x_{i,2}, \dots, x_{i,n})$ so that $0 \leq x_{i,n} \leq 3$, and for j from 1 to $n - 1$, $x_{i,j}$ is an integer with $0 \leq x_{i,j} \leq 2^{2^{n-j}} - 1$. Ensure also that for each integer k with $0 \leq i \neq k \leq 2^{2^n} - 1$, at least one of $x_{i,1} \neq x_{k,1}$, $x_{i,2} \neq x_{k,2}$, \dots , or $x_{i,n} \neq x_{k,n}$. The cardinality of the set of all possible $(x_{i,1}, x_{i,2}, \dots, x_{i,n})$ is therefore equal to the product of N 's dimensions.

$$2^{2^{n-1}} \times 2^{2^{n-2}} \times \dots \times 2^{2^2} \times 2^{2^1} \times 4 = 2^{2^n}.$$

So, the set of all n -tuples $(x_{i,1}, x_{i,2}, \dots, x_{i,n})$ is an ordering of precisely one copy each of $(0, 0, \dots, 0), \dots, (2^{2^{n-1}} - 1, 2^{2^{n-2}} - 1, \dots, 2^{2^1} - 1, 3)$. Reorder and relabel so that $(x_{0,1}, x_{0,2}, \dots, x_{0,n}) = (0, 0, \dots, 0)$ without loss of generality due to $[M]$'s hyper-toroidal shape.

Let M_0 be congruent to the $N \in [N]$ which has the origin in its former, front, top, left, etcetera corner. For each subsequent instance of M , assign

$$M_{i+1, a_1, a_2, \dots, a_n} = M_{i, b_1, b_2, \dots, b_n}$$

with $b_j = a_j - x_{i,j} \bmod 2^{2^{n-j}}$ for j from 1 to $n - 1$, while $b_n = a_n - x_{i,n} \bmod 4$.

6.2 Proving that $[M]$ satisfies Definition 13

Extending Lemma 2 from page 21 relates M_0 , $M_{2^{2^n}-1}$, and $(x_{2^{2^n}-1,1}, x_{2^{2^n}-1,2}, \dots, x_{2^{2^n}-1,n})$.

Lemma 5. $M_{0,a_1,a_2,\dots,a_n} = M_{2^{2^n}-1,b_1,b_2,\dots,b_n}$ with $b_j = a_j - x_{2^{2^n}-1,j} \pmod{2^{2^{n-j}}}$ for j from 1 to $n-1$, and $b_n = a_n - x_{2^{2^n}-1,n} \pmod{4}$.

Proof. As in Lemma 2. □

Now that the relationship between any two adjacent instances of $[M]$ is guaranteed to be unique, the following Lemmas assess the uniqueness of the $\underbrace{2 \times 2 \times \dots \times 2}_n$ arrays of 0s and 1s they contain.

Lemma 6. All 2^{2^n} of the $\underbrace{2 \times 2 \times \dots \times 2}_n$ arrays contained in $(M_i \times M_{i+1} \pmod{2^{2^n}})$ are distinct for all $0 \leq i \leq 2^{2^n} - 1$.

Proof. As in Lemma 3. □

Lemma 7. For integers i, j with $0 \leq i \neq j \leq 2^{2^n} - 1$, $(M_i \times M_{i+1} \pmod{2^{2^n}})$ and $(M_j, M_{j+1} \pmod{2^{2^n}})$ share none of the $\underbrace{2 \times 2 \times \dots \times 2}_n$ arrays they contain.

Proof. As in Lemma 4. □

Theorem 4. $[M]$ satisfies definition 13 as a $(\prod_{i=1}^n 2^{2^i}) \times 4$ De Bruijn array of binary 2^{n+1} -cube sub-arrays.

Proof. $[M]$ is a $2^{2^n} \times 2^{2^{n-1}} \times \dots \times 2^{2^2} \times 2^{2^1} \times 4$, $n+1^{th}$ dimensional hyper-toroidal array, so it has the size and shape dictated by Definition 13 using $n+1$ in place of n .

Lemmas 6 and 7 guarantee 2^{2^n} distinct $\underbrace{2 \times 2 \times \dots \times 2}_n$ arrays of 0s and 1s in each of its 2^{2^n} pairs of instances, so there are $2^{2^n} \times 2^{2^n} = 2^{2^{n+1}}$ distinct $\underbrace{2 \times 2 \times \dots \times 2}_{n+1}$

sub-arrays contained in $[M]$. Each space in the sub-array may be filled with either 0 or 1, so there are exactly $2^{2^{n+1}}$ possible $\underbrace{2 \times 2 \times \cdots \times 2}_{n+1}$ sub-arrays. $[M]$ must contain each one precisely once. \square

Having shown several base cases in lower dimensions, and using this as the inductive step, we verify that $2^{2^{n-1}} \times 2^{2^{n-2}} \times \cdots \times 2^{2^2} \times 2^{2^1} \times 4$ De Bruijn arrays of binary 2^n -cube sub-arrays exist for all $n \in \mathbb{N}$. This specific case of the De Bruijn sequence is generalizable to any finite number of dimensions.

In addition, the inductive nature of the construction implies a lower bound for the quantity of $\left(\prod_{i=1}^{n-1} 2^{2^i}\right) \times 4$ De Bruijn arrays of 2^n -cube sub-arrays. Consider that we found $2 \times 15!$ distinct $16 \times 4 \times 4$ De Bruijn arrays of b -cubes in section 4.1. Using any of these examples as $[N]$ in the inductive proof above with $n = 3$, we may permute the set of all 3-tuples $(x_{i,1}, x_{i,2}, x_{i,3})$ in $(2^{2^3} - 1)!$ ways with $(x_{0,1}, x_{0,2}, x_{0,3}) = (0, 0, 0)$ without loss of generality. So there are at least

$$2 \times 15! \times 255! \approx 8.7636 \times 10^{516}$$

distinct ways to construct $[M]$, which has dimension $n + 1 = 4$. Applying the inductive step, for any $n \geq 3$ we are guaranteed at least

$$2 \times \left(\prod_{i=2}^n (2^{2^i} - 1)! \right)$$

$\left(\prod_{i=1}^{n-1} 2^{2^i}\right) \times 4$ De Bruijn arrays of binary 2^n -cube sub-arrays.

7 Applications of De Bruijn Sequences and Torii

7.1 Mapping of Space and 3D Position Tracking

Locating Patterns in the De Bruijn Torus notes the use of De Bruijn torii as a means of locating position in 2D space[3]. For an abstract example, in a room with a De Bruijn array printed on the floor, a robot could determine its position by comparing the pattern on the floor beneath it to an internal map of the room. This general concept is used to detect positions in space for projective touchscreen displays and digital paper.

3D De Bruijn arrays expand the theoretical utility of the practice. Instead of a robot confined to the floor, imagine a flying drone capable of movement in three dimensions in a warehouse. We might suspend black and white markers in the warehouse in the shape of $F \in [F]$, any $16 \times 4 \times 4$ De Bruijn array of b -cubes. Then the drone could auto-self-locate by finding the pattern of the eight markers surrounding it in an internal map of F .

Using the same reasoning one might imagine a 3D “touchscreen,” a holographic 3D De Bruijn array with which a person interacts using a stylus. The stylus senses its location by detecting the combination of lights at its tip and finding that combination uniquely in the De Bruijn array.

7.2 DNA Sequencing and Efficient Packing

In shotgun DNA sequencing and genome assembly, lengths of one-dimensional genetic material are cut into pieces and the original DNA strand must be reconstituted. One algorithmic solution to this problem supposes that out of all the DNA strings obtainable by combining the pieces, the strings with the least repetition are probably the most accurate recreations of the original DNA. Applying this solution utilizes concepts from graph theory similar to the construction of a De Bruin sequence outlined in section 1. The process of making a De Bruijn sequence requires merging sequences of elements on their areas of overlap, a useful tactic in DNA sequencing[8].

With analogues to De Bruijn sequences existent in any finite number of dimensions, one might imagine an algorithm to combine multidimensional noise in a compact way. The process of finding overlapping or shared regions in data to merge the data in a maximally efficient space could be useful in any number of dimensions.

7.3 Error Detection and Correction

If any single binary element of $[F]$, a $16 \times 4 \times 4$ De Bruijn array of b -cubes, were to be inverted from a 0 to a 1 or vice versa, the properties of the De Bruijn array could be used to detect the change and reverse it. Compare this to Hamming Codes and error-detecting/correcting codes which ensure text messages are correctly transmitted even with messy cell signals[5]. Before we list rules which turn ‘wounded’ De Bruijn arrays into sudoku-like puzzles, we present an example method for encoding short messages in $16 \times 4 \times 4$ De Bruijn arrays of b -cubes.

As $2^{41} = 2199023255552 < 2 \times 15!$, it should be possible to injectively map 41-bit strings onto $16 \times 4 \times 4$ De Bruijn arrays of b -cubes (so that the arrays could be translated back into the strings they represent). Using 256 bits to represent 41 bits is inefficient, but if the De Bruijn array allows for powerful enough data-restoration algorithms, it may be worth the extra storage costs.

For efficiency’s sake, the process of translating binary strings to arrays and back should not take much computational time. The Appendix contains several Python functions which perform the task of encoding and decoding messages quickly using a binary tree ‘search’ algorithm, but only for binary strings of length 35 instead of 41 because of its methodology. The following algorithm describes the Python’s methodology:

Let B be a binary string with length 35. If the first digit of B is 0, we will make $[F]$ a $16 \times 4 \times 4$ De Bruijn arrays of b -cubes in the manner specified in section 5.1 using the Clockwise array. If the first digit of B is 1, we will use the Counterclockwise array to construct $[F]$. Having made this decision, remove the first element from B .

We assume the offset pair $(0, 0)$ is first without loss of generality due to $[F]$ ’s hyper-toroidal shape. Then make an ordered list of the unused pairs $(0, 1), (0, 2), (0, 3), (1, 0), (1, 1), \dots, (3, 3)$ and call the list *unusedPairs*. Until *unusedPairs* is empty, repeat the following task:

Divide *unusedPairs* into two halves in the middle, order-wise, including the central element in the second half if there are an odd number of elements. If the first digit of B is 0, focus on the first half of *unusedPairs*. If the first digit of B is 1, focus on the second half. Split the specified half in half again and focus on one half using the second digit of B , then split that half in half again, repeating until the list in question has only one element in it. Remove that element and use it to construct the next layer of $[F]$. Also remove the digits of B used to select that element.

When B is empty, $[F]$ will be a $16 \times 4 \times 4$ De Bruijn array of b -cubes which implicitly conveys B ’s message to anyone who knows the order of the original *unusedPairs* list.

For example, the phrase “Help!” in 7-bit ASCII is 1001000 1100101 1101100 1110000 0100001. Using Python functions from the appendix, `makePermutation([1,0,0,1,0,0,0,1,1,0,0,1,0,1,1,1,0,1,1,0,0,1,1,1,0,0,0,0,1,0,0,0,0,1])` produces [‘Counterclockwise’, (0, 0), (0, 3), (0, 1), (3, 1), (1, 2), (3, 3), (2, 0), (1, 0), (3, 0), (0, 2), (1, 1), (2, 3), (1, 3), (2, 1), (3, 2), (2, 2)], which describes a $16 \times 4 \times 4$ De Bruijn array of b -cubes. Feeding that list into `decodePermutation()` produces 1001000 1100101 1101100 1110000 0100001, or “Help!” In this manner, short messages can be stored in an error resistant format and later recovered.

If $[F]$ is a De Bruijn array with form as described in Section 5.1, but some elements are flipped, the following rules can help detect and correct the errors.

Rule 1. *Every layer of $[F]$ is the Clockwise or Counterclockwise array.*

If we may determine whether $[F]$ uses the Clockwise or Counterclockwise array, this rule allows us to reform any wounded layer which still has a 2×2 square of 0s and 1s which is known to be correct (or, indeed, any shape and quantity of correct elements that uniquely determines the placement of the Clockwise or Counterclockwise array forming that layer).

As each layer is congruent to the Clockwise or Counterclockwise array, each row and column of each layer of $[F]$ must contain three 0s and a 1, or three 1s and a 0. This can help detect errors and potentially correct those errors by, for example, inverting the element at the intersection of a row and a column in violation of this rule.

Rule 2. *$[F]$ contains each b -cube precisely once.*

This rule (from the definition 9) can fix wounded values using information elsewhere in $[F]$. If, for $s, t, u, v, w, x, y, z \in \{0, 1\}$, $[F]$ contains the b -cubes

$$\begin{aligned} & \left(\begin{bmatrix} s & t \\ u & v \end{bmatrix} \times \begin{bmatrix} w & x \\ y & z \end{bmatrix} \right), \quad \left(\begin{bmatrix} s & t \\ u & v \end{bmatrix} \times \begin{bmatrix} w & x \\ y & z \end{bmatrix} \right), \\ & \left(\begin{bmatrix} s & t \\ 1-u & v \end{bmatrix} \times \begin{bmatrix} w & x \\ y & z \end{bmatrix} \right), \quad \left(\begin{bmatrix} s & t \\ u & 1-v \end{bmatrix} \times \begin{bmatrix} w & x \\ y & z \end{bmatrix} \right), \\ & \left(\begin{bmatrix} s & t \\ u & v \end{bmatrix} \times \begin{bmatrix} 1-w & x \\ y & z \end{bmatrix} \right), \quad \left(\begin{bmatrix} s & t \\ u & v \end{bmatrix} \times \begin{bmatrix} w & 1-x \\ y & z \end{bmatrix} \right), \\ & \left(\begin{bmatrix} s & t \\ u & v \end{bmatrix} \times \begin{bmatrix} w & x \\ 1-y & z \end{bmatrix} \right), \quad \text{and} \quad \left(\begin{bmatrix} s & t \\ u & v \end{bmatrix} \times \begin{bmatrix} w & x \\ y & 1-z \end{bmatrix} \right), \end{aligned}$$

we might check if either of the duplicated copies of $\left(\begin{bmatrix} s & t \\ u & v \end{bmatrix} \times \begin{bmatrix} w & x \\ y & z \end{bmatrix} \right)$ should actually be $\left(\begin{bmatrix} s & 1-t \\ u & v \end{bmatrix} \times \begin{bmatrix} w & x \\ y & z \end{bmatrix} \right)$. Then, having confirmed a 2×2 square of elements on the layer containing errors, rule 1 corrects all errors on the layer.

Using these rules we turn wounded $16 \times 4 \times 4$ De Bruijn arrays of b -cubes into puzzles to be solved. In fact, if all elements are totally removed except for a 2×2 square of elements on each layer (so 75% of the data is lost) then if we know whether $[F]$ used Clockwise or Counterclockwise layers, the whole array may be repaired using rule 1 alone.

8 Further Questions

- Are there $16 \times 4 \times 4$ De Bruijn arrays of b -cubes which do not have the form prescribed in section 5.1? We conjecture that there are in section 5.3.
- Can De Bruijn arrays of b -cubes have shape other than $16 \times 4 \times 4$?
- Can stacking De Bruijn tori be used to generate 3D De Bruijn hyper-tori for sub-arrays of different sizes, shapes, or number of elements? Does the stacking-with-offsets algorithm generalize to more complicated cases?
- A common question is the minimum time required to locate specific sequences in a De Bruijn sequence. What is the minimum time required to locate an arbitrary cube in a $16 \times 4 \times 4$ De Bruijn array of b -cubes? Could the structure in three dimensions allow for efficient search algorithms?

9 Appendix

9.1 Encoding and Decoding messages in De Bruijn Arrays (Section 7.4.1)

```
import math

#####
# Recursive step for the makePermutation function

def nextStep(bitString, pairList,i):
    if(len(pairList)==1): # If passed a list of length 1, returns its element as the answer
        return([pairList[0],i])
    elif(bitString[0]==0): # Otherwise, cuts the list of remaining permutations in half
        nextString = bitString[1:len(bitString)]
        return(nextStep(nextString,pairList[:math.floor(len(pairList)/2)],i+1))
    else: # Which half is used in the recursive step is based on the sequence of 0s and 1s
        nextString = bitString[1:len(bitString)]
        return(nextStep(nextString,pairList[math.ceil(len(pairList)/2):],i+1))

#####
# Accepts 35-digit string, returns a list unique to that string beginning with
# 'Clockwise' or 'Counterclockwise, then (0,0), then ordering remaining offset pairs,
# specifying a specific 16*4*4 De Bruijn array of cubes

def makePermutation(bitString):
    ourPermutation = []
    unusedPairs = [(0,1),(0,2),(0,3),(1,0),(1,1),(1,2),(1,3),(2,0),(2,1),(2,2),(2,3),(3,0),
(3,1),(3,2),(3,3)] # List of the 15 elements to be organized

    if(bitString.pop(0)==0):
        ourPermutation.append("Clockwise")
    else:
        ourPermutation.append("Counterclockwise") # First digit chooses direction of the swirl

    ourPermutation.append((0,0)) # WLOG, use this offset first

    while(unusedPairs != []):
        nextElement = nextStep(bitString,unusedPairs,0) # Recursive step
        ourPermutation.append(nextElement[0])
        unusedPairs.remove(nextElement[0])
        bitString = bitString[nextElement[1]:]

    print(ourPermutation)
    return(ourPermutation)
```

```

#####
# Recursive step for the decodePermutation function
def stepReverser(element,pairList,toAppend):
    if(len(pairList)==1): # If the list is finished, return the constructed string
        return(toAppend)
    # If element is in first half, append 0
    elif(pairList.index(element) < math.floor(len(pairList)/2)):
        toAppend.append(0)
        return(stepReverser(element,pairList[:math.floor(len(pairList)/2)],toAppend))
    else: # If element is in second half, append 1
        toAppend.append(1)
        return(stepReverser(element,pairList[math.ceil(len(pairList)/2):],toAppend))

#####
# Accepts a list just like makePermutation outputs
# and returns the (first 35 digits of) the binary string which produced it
def decodePermutation(permutation):
    theBitString = []

    if(permutation[0] == "Clockwise"):
        theBitString.append(0)
    else:
        theBitString.append(1)

    permutation = permutation[2:] # Already processed first element, ignore (0,0)

    asItWas = [(0,1),(0,2),(0,3),(1,0),(1,1),(1,2),(1,3),(2,0),(2,1),(2,2),(2,3),(3,0),
(3,1),(3,2),(3,3)]

    while(asItWas != []):
        nextDigits = stepReverser(permutation[0],asItWas,[])
        for i in nextDigits:
            theBitString.append(i)
        asItWas.remove(permutation[0])
        permutation = permutation[1:]

    print(theBitString)

```


References

- [1] Alexander Bogomolny, Existence of the de Bruijn Cycles via Graphs from *Interactive Mathematics Miscellany and Puzzles*. <http://www.cut-the-knot.org/arithmetic/combinatorics/deBruijnCycles.shtml>. Accessed 21 December 2016. (This is the web-author's stated preference for citation notation.)
- [2] Eggen, Bernd R. (1990). "The Binatrix B2". *Private communication*.
- [3] Horan, Victoria, and Brett Stevens. "Locating Patterns in the De Bruijn Torus." *Discrete Mathematics* 339.4 (2015): 1274-282. Web. 11 Feb. 2017. <<https://arxiv.org/pdf/1505.04065.pdf>>.
- [4] Hurlbert, Glenn, and Garth Isaak. "On the De Bruijn Torus Problem." *Journal of Combinatorial Theory A* 64.1 (1993): 50-62. *Science Direct*. Web. 21 Dec. 2016. <<http://www.sciencedirect.com/science/article/pii/009731659390087O>>.
- [5] Parker, Matt. *Things to Make and Do in the Fourth Dimension*. London: Penguin, 2015. Print.
- [6] Perry, Nick. "De Bruijn Sequences." *Data Genetics*. 2 Oct. 2013. Web. 11 Jan. 2017. <<http://www.datagenetics.com/blog/october22013/index.html>>.
- [7] Perry, Nick. "Wounded QR Codes." *Data Genetics*. 1 Nov. 2013. Web. 11 Feb. 2017. <<http://datagenetics.com/blog/november12013/index.html>>.
- [8] Zerbino, Daniel Robert. "Genome Assembly and Comparison Using De Bruijn Graphs." *Genome Assembly and Comparison Using De Bruijn Graphs*, University of Cambridge, 2009. <www.ebi.ac.uk/sites/ebi.ac.uk/files/shared/documents/phdtheses/daniel_zerbino.pdf>.