Review on de Bruijn shapes in one, two and three dimensions

Pedro J. Roig$^{1,2}$, Salvador Alcaraz$^1$, Katja Gilly$^1$, Cristina Bernad$^1$, Carlos Juiz$^2$

$^1$Miguel Hernández University, Elche (SPAIN)  
$^2$University of the Balearic Islands (SPAIN)

E-mail: \{proig, salcaraz, katya, cbernad\}@umh.es, cjuiz@uib.es

Abstract. Working with ever growing datasets may be a time consuming and resource exhausting task. In order to try and process the corresponding items within those datasets in an optimal way, de Bruijn sequences may be an interesting option due to their special characteristics, allowing to visit all possible combinations of data exactly once. Such sequences are unidimensional, although the same principle may be extended to involve more dimensions, such as de Bruijn tori for bidimensional patterns, or de Bruijn hypertori for tridimensional patterns, even though those might be further expanded up to infinite dimensions. In this context, the main features of all those de Bruijn shapes are going to be exposed, along with some particular instances, which may be useful in pattern location in one, two and three dimensions.

1. Introduction

To start with, let us consider the character set $\Sigma_k = \{0, \ldots, k-1\}$ as an alphabet of length $k$ (where $k \geq 2$), meaning the number of distinct symbols available. On the other hand, let us consider the set $\Sigma_k^n$ as the collection of all possible strings of length $n$ being built upon $\Sigma_k$ (where $n \geq 1$), representing all $k$-ary strings of length $n$ available [1].

In this context, if $S$ is a non-empty subset of $\Sigma_k^n$, a universal cycle for $S$ may be defined as a sequence of length $|S|$ containing every string of length $n$ (being encoded in alphabet $\Sigma_k$) exactly once when such a sequence is seen in a circular manner. Additionally, if $S$ includes the whole set $\Sigma_k^n$, such as $S = \Sigma_k^n$, resulting in $|S| = k^n$, then a universal cycle for $S$ is known as a de Bruijn sequence [2].

Therefore, the main feature of such sequences is that they contain all possible combinations of $k$-ary $n$-strings precisely once, which opens up a wide range of applications. For instance, they may be used for testing purposes in an efficient way [3], obtaining pseudorandom values [4] or dealing with coding theory [5].

Additionally, the most prominent use of those circular sequences is related to shotgun DNA sequencing in order to reassemble large DNA strings out of smaller substrings [6], even though it is also widely used in sequence analysis techniques by means of bioinformatics algorithms [7].

However, this Paper is going to be related to the use of de Bruijn shapes for location detection by means of pattern spotting, as each available pattern in a given dimension may arise just once. Basically, de Bruijn shapes are going to be defined for one, two and three dimensions, even though they might be determined for higher dimensions, even for infinite ones. Moreover, this Paper may be seen as an extension of [8], even though a different approach has been taken herein.
Regarding the dimensions involved, the name of the shape and its representation change:

- **Unidimensional (1D):** de Bruijn sequence, represented as a toroidal array.
- **Bidimensional (2D):** de Bruijn torus, represented as a toroidal matrix.
- **Tridimensional (3D):** de Bruijn hypertorus, represented as a hypertoroidal matrix.

The rest of the Paper is organized as follows: first, Section 2 exposes de Bruijn sequences, then, Section 3 explains de Bruijn tori, after that, Section 4 exhibits de Bruijn hypertori, and finally, Section 5 draws some conclusions.

### 2. De Bruijn Sequences: one dimension

Sticking to de Bruijn sequences, which are related to one dimension as sequences grow linearly, the key point is that an item being in different positions within a certain circular trajectory (considered as a loop) will be spotting different \( n \)-string patterns along the way.

Therefore, the pattern being spotted at a certain point may indicate the position of the moving item along that round trajectory, which may also apply for angle encoding techniques.

Furthermore, this behaviour may also be extrapolated to rectilinear trajectories by unwrapping the loop, which may produce that a whole \( n \)-string pattern may not be seen near both ends of the rectilinear segment, but just a certain subset of it near the tip. Hence, focusing on rectilinear trajectories, it may be said that de Bruijn sequences allow the location detection in one dimension, as the patterns spotted throughout a rectilinear path depend on the position.

In this sense, Figure 1 depicts a de Bruijn sequence where the alphabet is \( \{0, 1\} (k = 2) \) and the length is \( n = 2 \), being represented as a wraparound linear array and in a circular fashion, containing all \( k \)-ary strings of length \( n \) precisely once. All those strings are \((0,0), (0,1), (1,1), (1,0)\), which may be spotted right in that order by starting from the left and going rightwards in the former, or starting on top and going clockwise in the latter.

It is to be said that the number of different instances of a particular \( k \)-ary \( n \)-length de Bruijn sequence is given by (1), which was deduced in [9], and it just depends on the values of \( k \) and \( n \).

\[
\frac{(k!)^{k^{n-1}}}{k^n}
\]

It is to be noted that rotations induce an equivalence relation in de Bruijn sequences, yielding an equivalence class under rotation, thus applying a circular rotation on a given instance does not account for a new one. On the contrary, reflections does not induce any equivalence relation in de Bruijn sequences, hence applying reflection on a particular instance does create a new one, resulting in a clockwise solution and a counterclockwise solution, provided that one of them may not be achieved by simple rotation of the other one, as it does happen above in Figure 1.

At this point, it is to be said that these sequences are named after the de Bruijn’s research [10] in the middle of the 20th century regarding combinatorics, specifically about strings of integers including all available substrings of a certain length appearing exactly once. He was not the first one to look into it, but he proved a conjecture stating that the count for \( n \)-length binary sequences was \( 2^{2^{n-1}} - n \). Therefore, as a homage to him, those sequences are denoted as \( B(k, n) \).

As an example, table 1 shows one instance of all de Bruijn sequences with length up to 64.

---

**Figure 1.** Unidimensional toroidal array (4 nodes).
Table 1. De Bruijn sequences with length up to 64.

| k | B(k, n) | One particular instance of that B(k, n) |
|---|--------|---------------------------------------|
| {0, 1} | B(2, 2) | 1100 |
| {0, 1} | B(2, 3) | 11101000 |
| {0, 1} | B(2, 4) | 1111010110010000 |
| {0, 1} | B(2, 5) | 1111101101110101010000100101100 |
| {0, 1} | B(2, 6) | 111111011011101101101100110010010011100010001100001000000 |
| {0 .. 2} | B(3, 2) | 221100201 |
| {0 .. 2} | B(3, 3) | 22212111000112201202021010 |
| {0 .. 3} | B(4, 2) | 33222110023130120 |
| {0 .. 3} | B(4, 3) | 33303002211222020010111000321201231201302323130133131332 |
| {0 .. 4} | B(5, 2) | 44313233034242414021120100 |
| {0 .. 5} | B(6, 2) | 5544534352432251413121150403020100 |
| {0 .. 6} | B(7, 2) | 66456233642446343532522615505130416034104021120100 |
| {0 .. 7} | B(8, 2) | 776675655746547363334337262524232271615141312117106050403020100 |

Figure 2. De Bruijn graph: Hamiltonian B(2,3) and Eulerian B(2,4).

In order to achieve de Bruijn sequences with an alphabet \( k \), it is to be reminded that an \( n \)-string contains an \( (n-1) \)-string as a prefix and an \( (n-1) \)-string as a suffix, which may be seen as if each \( n \)-string is the result of overlapping two \( (n-1) \)-strings, whereas overlapping all \( n \)-strings in a circular way gives out a \( k^n \)-length superstring. Such a superstring is called a de Bruijn sequence, which happens to contain all available \( n \)-strings being composed with a \( k \) alphabet precisely once. This special feature helps build up de Bruijn sequences in two main ways, such as graphically and algorithmically.

Regarding the graphical way, de Bruijn graphs do the job in a pretty straightforward manner. They are defined as directed nonempty connected regular graphs, which are also simple, although loopback links appear in every single-digit string. Those characteristics permit the establishment of both a Hamiltonian and a Eulerian cycle, hence de Bruijn graphs may be considered as both Hamiltonian and Eulerian. Furthermore, each de Bruijn graph with \( n \)-length strings may be seen as the line digraph of \( (n-1) \)-length strings, which accounts for its dual representation.

Hence, it may be said that a single de Bruijn graph may generate a de Bruijn sequence \( B(k, n) \) by means of a Hamiltonian cycle when going around each of its \( k^n \) nodes just once, and also a de Bruijn sequence \( B(k, n+1) \) by following a Eulerian cycle when going through each of its \( k^{n+1} \) edges only once. For instance, Figure 2 depicts a binary de Bruijn graph with 3-length nodes, permitting the buildup of \( B(2,3) \) and \( B(2,4) \), whereas Figure 3 exhibits a quaternary de Bruijn graph with 1-length nodes, allowing the construction of \( B(4,1) \) and \( B(4,2) \).

Regarding the algorithmic way, there have been different algorithms proposed in order to achieve de Bruijn sequences in an analytical way [11], such as greedy, feedback shift register sequence or FKM construction. However, the most efficient one has been proposed by Wong [12],
as it may generate each successive symbol within a sequence in $O(1)$-amortized time, being valid for any type of alphabet, even though the method slightly differs between a binary and a $k$-ary.

On the one hand, the version for a binary alphabet, where $k = \{0, 1\}$, just involves to verify whether a particular test string is indeed a necklace, that being defined as the lexicographically smallest string within an equivalence class of strings under rotation \[13\]. Hence, depending on that verification, the following symbol within a binary de Bruijn sequence is obtained, that being the opposite of the leftmost bit in case of positive verification, or otherwise, that bit, as in (2).

\[
f_2(b_1 b_2 \cdots b_n) = \begin{cases} 
    b_2 b_3 \cdots b_n b_1 & \text{if } b_2 b_3 \cdots b_n b_1 \text{ is a necklace;} \\
    b_2 b_3 \cdots b_n b_1 & \text{otherwise.}
\end{cases} \tag{2}
\]

On the other hand, the version for a generic $k$ alphabet, only requires to search for the highest value of variable $b$ where the string $a_2 \cdots a_n b$ is not a necklace, or 0 instead, or otherwise, checking whether $a_2 \cdots a_n (a_1 + 1)$ is a necklace, or alternatively, performing a circular left shift, as in (3).

\[
f_k(a_1 a_2 \cdots a_n) = \begin{cases} 
    a_2 a_3 \cdots a_n b & \text{if } a_1 = k - 1; \\
    a_2 a_3 \cdots a_n (a_1 + 1) & \text{if } a_1 \neq k - 1 \text{ and } a_2 a_3 \cdots a_n (a_1 + 1) \text{ is a necklace;} \\
    a_2 a_3 \cdots a_n a_1 & \text{otherwise.}
\end{cases} \tag{3}
\]

As a side note, necklaces are also called fixed necklaces, which are not to be confused with bracelets, also known as turnover necklaces, those being part of an equivalence class of strings under both rotation and reflection \[14\]. As stated before, this equivalence relation does not apply to de Bruijn sequences, but it does to other entities such as combinatorial necklaces, those being circles formed by $n$ coloured beads with up to $k$ diverse colors.

As an example of Wong’s algorithm for binary alphabets, table 2 exhibits its execution in order to obtain a binary de Bruijn sequence $B(2, 4)$, which may be compared to the Eulerian path indicated in Figure 4, as both methods give out the same outcome, leading to the $2^4$-length de Bruijn sequence 0000111101100101.

Likewise, as an instance of Wong’s algorithm for generic $k$ alphabets, table 3 exposes its execution so as to attain a de Bruijn sequence $B(4, 2)$, which may be confronted to the Eulerian path shown in Figure 5. Both methods get the same result, leading to the $4^2$-length de Bruijn sequence 0011223321310203.

It is to be reminded that applying rotation in a de Bruijn sequence results in another instance of it belonging to the same equivalence class under rotation. Hence, focusing on de Bruijn graphs, the aforementioned sequences may be obtained by taking in a sequential manner the weights of all edges, and then applying rotation to reach the desired outcome. On the other hand, focusing on Wong’s algorithm, the aforesaid sequences may be achieved by taking a fixed bit in all $n$-strings in a sequential fashion, and then applying rotation to get the desired result.
Table 2. Wong’s algorithm to obtain a de Bruijn sequence $B(2,4)$.

| $b_1b_2b_3b_4$ | $b_2b_3b_41$ | Necklace? | YES: $b_2b_3b_41$ | NO: $b_2b_3b_4b_1$ |
|-----------------|---------------|-----------|---------------------|---------------------|
| 0000            | 0001          | Yes       | 0001                |                     |
| 0001            | 0011          | Yes       | 0011                |                     |
| 0011            | 0111          | Yes       | 0111                |                     |
| 0111            | 1111          | Yes       | 1111                |                     |
| 1111            | 1111          | No        | 1110                |                     |
| 1110            | 1101          | No        | 1101                |                     |
| 1101            | 1011          | No        | 1011                |                     |
| 1011            | 0111          | Yes       | 0110                |                     |
| 0110            | 1101          | No        | 1000                |                     |
| 1001            | 1001          | No        | 1001                |                     |
| 1000            | 0011          | Yes       | 0010                |                     |
| 0010            | 0101          | Yes       | 0101                |                     |
| 0101            | 1011          | No        | 1010                |                     |
| 1010            | 0101          | Yes       | 0100                |                     |
| 0100            | 1001          | No        | 1000                |                     |
| 1000            | 0001          | Yes       | 0000                |                     |

Figure 4. De Bruijn graph pointing out a Eulerian cycle $B(2,4)$.

Figure 5. De Bruijn graph indicating a Eulerian cycle $B(4,2)$.

3. De Bruijn Tori: two dimensions

If the concept of de Bruijn sequences is extended to higher dimensions, it may also be possible to carry most of their uses to those higher dimensions, such as coding, communications, pseudorandom arrays or spectral imaging [15]. However, location detection may well be one of the most interesting applications, as each pattern will occur exactly once. Hence, if two dimensions are involved, bidimensional patterns may be spotted throughout a wraparound 2D array [16], whereas if three dimensions are implied, tridimensional patterns may do accordingly.
Table 3. Wong’s algorithm to obtain a de Bruijn sequence $B(4, 2)$.

| Now: $a_1a_2$ | $a_1 = 3$? | $a_2b$ | Necklace? | $a_2(a_1 + 1)$ | Necklace? | Next string |
|---------------|------------|---------|-----------|----------------|-----------|-------------|
| 00            | No         | -       | 01        | Yes            | 01        |             |
| 01            | No         | -       | 11        | Yes            | 11        |             |
| 11            | No         | -       | 12        | Yes            | 12        |             |
| 12            | No         | -       | 22        | Yes            | 22        |             |
| 22            | No         | -       | 23        | Yes            | 23        |             |
| 23            | No         | -       | 33        | Yes            | 33        |             |
| 33            | Yes        | 32      | No        | -              | -         | 32          |
| 32            | Yes        | 21      | No        | -              | -         | 21          |
| 21            | No         | -       | 13        | Yes            | 13        |             |
| 13            | No         | -       | 32        | No             | 31        |             |
| 31            | Yes        | 10      | No        | -              | -         | 10          |
| 10            | No         | -       | 02        | Yes            | 02        |             |
| 02            | No         | -       | 21        | No             | 20        |             |
| 20            | No         | -       | 03        | Yes            | 03        |             |
| 03            | No         | -       | 31        | No             | 30        |             |
| 30            | Yes        | 00      | Yes       | -              | -         | 00          |

Considering the number of dimensions as $N$, it may be said that a de Bruijn hypertorus is an $N$-dimensional shape where each $k$-ary $N$-dimensional pattern available is spotted precisely once. Therefore, that shape and that pattern may be defined as $\mathbf{r} = (r_1 \cdots r_N)$ and $\mathbf{m} = (m_1 \cdots m_N)$, respectively, where $r_i > m_i$ for $1 \leq i \leq N$, resulting in the compacted expression $(\mathbf{r}; \mathbf{m})_k^N$.

Different conditions may induce the existence of a de Bruijn hypertorus, although it has been proved that if expression (4) applies, then it is a sufficient condition for it to exist [17]. Furthermore, an $N$-hypercubic form for either the shape or the pattern is achieved if the value for all its dimensions match, or otherwise, some type of rectangular hypervolume form is obtained.

$$\prod_{i=1}^{N} r_i = k \prod_{i=1}^{N} m_i$$ (4)

Focusing on 2 dimensions, the aforesaid shape may be referred to as de Bruijn torus, and in such a case, the aforesaid expression is simplified to (5), where $r_1$ and $r_2$ define the size for both dimensions of the toroidal matrix and $m_1$ and $m_2$ do it for the pattern, whereas $k$ denotes the alphabet where the values may be chosen from.

$$r_1 \cdot r_2 = k^{m_1 \cdot m_2}$$ (5)

The usual way to express a de Bruijn torus is $(r_1, r_2; m_1, m_2)_k$, which presents a rectangular de Bruijn torus with a rectangular pattern. However, it may happen that the toroidal shape is squared, which accounts for $r_1 = r_2 = r$, giving a square de Bruijn torus with a rectangular pattern. On the other hand, it may occur that the pattern is squared, which results in $m_1 = m_2 = m$, obtaining a rectangular de Bruijn torus with a square pattern. Moreover, both cases may arise at the same time, representing a square de Bruijn torus with a square pattern, and in such a case, it may be represented as $(r; r; m, m)_k$, or additionally, as $(k^{m/2}, k^{m/2}; m, m)_k$ if variable $r$ is taken from the aforesaid expression (5). Additionally, not only regular patterns are possible, but other kind of shapes might be allowed as well, such as patterns with L-shapes [18].
The smallest binary square de Bruijn torus with a square pattern is \((4, 4; 2, 2)_2\), which represents a \(4 \times 4\) toroidal array containing all binary \(2 \times 2\) toroidal patterns only once. This accounts for combining all possible values in the \(4\) positions within the pattern, and as each of those only may hold a 0 or a 1 in a binary alphabet, then \(k^{m \cdot m} = 2^{2 \cdot 2} = 2^4 = 16\).

Figure 6 exhibits the layout of \((4, 4; 2, 2)_2\) where the 1’s form a clockwise Brigid’s cross, which along with its counterclockwise are the only possible configurations, whereas Figure 7 shows the mapping of all 16 patterns in that shape thanks to its wraparound nature, where the top left element in each submatrix is called its handle, which is the reference point for all \(2 \times 2\) mappings.

Other square de Bruijn tori with square patterns are feasible by using a binary alphabet, such as \((256, 256; 4, 4)_2\), \((2^{18}, 2^{18}; 6, 6)_2\), whereas other \(2 \times 2\) patterns are available, such as \((9, 9; 2, 2)_3\) and \((16, 16; 2, 2)_4\). Moreover, other options are available, such as a rectangular de Bruijn torus with square pattern is given by \((16, 32; 3, 3)_2\), whilst a square de Bruijn torus with rectangular pattern is attained in \((8, 8; 3, 2)_2\), whereas a rectangular de Bruijn torus with a rectangular pattern is shown in \((4, 16; 3, 2)_2\). In this sense, Figure 8 exhibits an instance of those with small size. It is to be noted that all these instances meet the requirement imposed in expression (5).

![Figure 6. Bidimensional toroidal array.](image)

![Figure 7. Square de Bruijn torus \((4, 4; 2, 2)_2\) mapping its square patterns.](image)

![Figure 8. Various bidimensional toroidal arrays.](image)
4. De Bruijn Hypertori: three dimensions

As stated in the previous section, this scenario is just the particular case of the general extension of de Bruijn sequences for higher dimensions where the number of dimensions is just 3. In this case, if the size of all dimensions of the de Bruijn 3D hypertorus is the same, then it is known as cubic de Bruijn 3D hypertorus, and likewise, if the pattern meets this condition, then it is called cubic pattern. Besides, if both requirements are fulfilled, it results in \( r = k^{m^3/3} \), thus the representation becomes \((r, r, r; m, m, m)_k\), or otherwise, \((k^{m^3/3}, k^{m^3/3}, k^{m^3/3}; m, m, m)_k\), which may be extended for higher dimensions, leading to \( r_1 = \cdots = r_N = k^{mN/N} \). Anyway, no matter if shapes and patterns are cubic or rectangular prism, expression (6) derives from (4) for 3D.

\[
 r_1 \cdot r_2 \cdot r_3 = k^{m_1 \cdot m_2 \cdot m_3}
\]  

Regarding the smallest cubic de Bruijn 3D hypertorus with cubic patterns, it accounts for \((256, 256, 256; 2, 2, 2)_8\) [19]. However, focusing on a binary alphabet, the smallest rectangular prism de Bruijn 3D hypertorus with binary cubic patterns accounts for \((16, 4, 4; 2, 2, 2)_2\) [20].

With respect to that pattern, which is a \(2 \times 2 \times 2\) binary cubic hypertoroidal array, also known as b-cube, it is presented in Figure 9, where its front, top, left item is its handle, whose role has been exposed above. On the other hand, the de Bruijn 3D hypertorus is exhibited in Figure 10, which is composed by 16 wraparound \(4 \times 4\) matrices, where each of those may be considered as a flat torus, as all of them are also wrapped around. As a consequence, each node has 2 neighbours per dimension if its size is higher than 2, yielding up to 6 neighbours per node.

![Figure 9. B-cube (2×2×2 nodes).](image1)

![Figure 10. Tridimensional hypertoroidal array (16 × 4 × 4 nodes).](image2)

In this context, each b-cube has a pair of \(2 \times 2\) matrices wrapped around together, where each one may keep up to 16 different patterns, as exposed in the previous section. Hence, the amount of combinations for both matrices is \(16^2 = 256\), accounting for all diverse b-cubes available within the de \(16 \times 4 \times 4\) Bruijn hypertorus, which ought to consider its own handle as its coordinate origin in order to better organize all b-cube handles as 3-tuples (front, top, left).

A strategy to go through all those combinations may be to fix the front \(2 \times 2\) matrix and to quote all combinations of the back \(2 \times 2\) matrix in an ordered way, and then, do the same with each combination of the front \(2 \times 2\) matrix. Having done this, it is to be noted that the group of 16 elements sharing the same front \(2 \times 2\) matrix hold their handles in a different \(4 \times 4\) matrix.

Likewise, if the handles of a given group are taken as references for each of the 16 arrays, it occurs that the handles of any other particular group occupy the same relative position in all matrices with respect to those references in all 16 arrays, considering the wraparound links.

An instance is shown in table 4, where the de Bruijn 3D hypertorus is presented as a wraparound array of 16 wraparound \(4 \times 4\) matrices, and where each one of the 256 available combinations of b-cubes are spotted precisely once.
Table 4. An instance of a $16 \times 4 \times 4$ de Bruijn hypertorus.

| Array ID | Bidimensional Array | Array ID | Bidimensional Array |
|----------|---------------------|----------|---------------------|
| 0        | $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$ | 1        | $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ |
| 2        | $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$ | 3        | $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$ |
| 4        | $\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ | 5        | $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$ |
| 6        | $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$ | 7        | $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$ |

5. Conclusions

In this Paper, a review has been undertaken on the theory and practice about de Bruijn shapes, whose main characteristic is that all available patterns made of a $k$-ary alphabet and a determined size are present exactly once within the shape. To start with, de Bruijn sequences and the corresponding patterns have been presented as unidimensional strings, and two methods have been used to obtain them, such as graphically, by means of de Bruijn graphs, and algorithmically, by means of Wong’s algorithm, verifying that the same results have been obtained in both cases. Besides, some other instances of such sequences with length up to 64 have been proposed.

Furthermore, de Bruijn sequences have been extended to higher dimensions, generating de Bruijn hypertori, and presenting a generic template for the most common condition to achieve
such shapes. Focusing on de Bruijn tori, which are bidimensional, as well as the associated patterns, it has been exposed that both the tori and the patterns may be either square or rectangular, making possible any kind of combination between them. Moreover, an instance of the smallest square de Bruijn torus with a square binary pattern has been presented, along with some other instances with a manageable size related any of those combinations.

Additionally, regarding de Bruijn 3D hypertori, which are tridimensional and so are the related patterns, it has been explained that the hypertori and the patterns may be either cubic or rectangular prism, being possible to combine them. In that sense, an instance of a cubic de Bruijn 3D hypertorus with a cubic pattern has just been introduced, whilst an instance of the smallest rectangular prism de Bruijn 3D hypertori with a cubic binary pattern has been presented as a wraparound array of wraparound matrices.

The main feature of de Bruijn shapes is that all available patterns of a certain size, being built with a particular alphabet, are present in each of those shapes exactly once, which opens up a great range of applications and opportunities. Specifically, location detection has been proposed as one of the most interesting ones, because the location of an item may be defined by means of the pattern being spotted by that item, whereas movement and its direction may be detected when the pattern spotted changes. Eventually, as de Bruijn shapes may be defined for any dimensions, such location detection skills may be applied to one, two or three dimensions.

References
[1] Gabric D and Sawada J 2018 Constructing de Bruijn sequences by concatenating smaller universal cycles Theoretical Computer Science 743 12-22
[2] Sawada J, Williams A and Wong D 2016 Generalizing the Classic Greedy and Necklace Constructions of de Bruijn Sequences and Universal Cycles The Electronic Journal of Combinatorics 23 1.24
[3] Almansi E and Becher V 2020 Completely uniformly distributed sequences based on de Bruijn sequences Mathematics of Computation 89 2537-2551
[4] Tsatsin A A 2021 Synthetic pseudo-random sequence IOP Conf. Ser.: Mater. Sci. Eng. 1027 012028
[5] Chee Y M et al 2020 Constrained de Bruijn Codes: Properties, Enumeration, Constructions, and Applications arXiv (Preprint 2005.03102)
[6] Song L F et al 2021 Robust data storage in DNA by de Bruijn graph-based decoding bioRxiv (Preprint 10.1101/2020.12.20.423642)
[7] Brinda K, Baym M and Kucherov G 2021 Simplitigs as an efficient and scalable representation of de Bruijn graphs Genome Biology 22 96
[8] Roig P J et al 2021 De Bruijn-based and k-ary n-cube-based algebraic models in fog environments (Cham: Springer) in press
[9] van Aardenne-Ehrenfest T and de Bruijn N 1951 Circuits and trees in oriented linear graphs Mathematical Reviews 28 758-764
[10] de Bruijn N G 1946 A combinatorial problem Proc. Koninklijke Nederlandse Akademie V. Wetenschappen 49 758-764
[11] Fredricksen H 1982 A Survey of Full Length Nonlinear Shift Register Cycle Algorithms Society for Industrial and Applied Mathematics 24(2) 195-221
[12] Wong C H 2015 Novel universal cycle constructions for a variety of combinatorial objects (PhD Thesis in Computer Science: University of Gelph)
[13] Sawada J, Williams A and Wong D 2017 A simple shift rule for k-ary de Bruijn sequences Discrete Mathematics 340(3) 524-531
[14] Sawada J 2001 Generating Bracelets in Constant Amortized Time Society for Industrial and Applied Mathematics 31(1) 259-268
[15] Hurlbert G and Isak G 1993 On The De Bruijn Torus Problem Journal of Combinatorial Theory Series A 64(1) 50-62
[16] Horan V Stevens B 2016 Locating Patterns in the De Bruijn Torus Discrete Mathematics 339(4) 1274-1292
[17] Kapinya J B 2004 Evolutionary Computing Solutions for the de Bruijn Torus Problem (Master’s Thesis in Computer Science: University of Vrije)
[18] Pudwell L and Rockey R 2014 de Bruijn Arrays for L-Fillings Mathematics Magazine 87(1) 57-60
[19] Horváth M Ivány A 2008 Growing perfect cubes Discrete mathematics 308(19) 4378-4388
[20] Casteels K and Tinker T 2018 De Bruijn Sequences of Higher Dimension (Master’s Thesis in Computer Science: University of California)