Bachelorproject: Shift Registers and De Bruijn Graphs

Bachelorproject - Version 3.0

Christine van Vredendaal
0651675

July 20, 2011
Abstract

This essay is an attempt to create a generalized periodic shift register function that produces a De Bruijn sequence. To this end we first devise an algorithm to create all De Bruijn sequences. In this algorithm all spanning trees of a De Bruijn graph are created, these trees are converted into Euler paths and finally the De Bruijn sequences are extracted from the Euler paths. Then the focus shifts onto creating the boolean functions that produce these sequences. The minimal Sum-of-Products boolean functions and the Exclusive-OR-Sum-of-Products are discussed. Finally some general properties of the functions are derived, but no general function is found.
Contents

1 Introduction 4

2 De Bruijn 4
   2.1 De Bruijn Sequence 4
   2.2 De Bruijn Graph 4
   2.3 Finding a De Bruijn Sequence 5

3 Shift Registers 6
   3.1 A Shift Register 6
   3.2 Shift Register Functions 7
      3.2.1 Karnaugh maps 7
      3.2.2 Exclusive-OR-Sum-Of-Products 7

4 TutteMatrix 8

5 Finding all spanning trees 9

6 Creating Euler circuits from spanning trees 12
   6.1 Euler circuit to spanning tree 12
   6.2 Spanning tree to Euler circuit 13

7 Lijstje 14

8 De Bruijn Graphs: OR-operator 15
   8.1 The Trivial Graph: \( n = 1 \) 15
   8.2 The De Bruijn Graph: \( n = 2 \) 15
   8.3 The De Bruijn Graph: \( n = 3 \) 16
   8.4 The De Bruijn Graph: \( n = 4 \) 18
   8.5 The De Bruijn Graph: \( n = 5 \) 20
   8.6 Finds 20
   8.7 Conjectures 21

9 De Bruijn Graphs: eXOR-operator 22
   9.1 The De Bruijn Graph: \( n = 1 \) 22
   9.2 The De Bruijn Graph: \( n = 2 \) 22
   9.3 The De Bruijn Graph: \( n = 3 \) 22
   9.4 The De Bruijn Graph: \( n = 4 \) 22
   9.5 Finds 22
   9.6 Conjectures 23

10 Conclusion 24

A Bronnen 24
1 Introduction

Suppose in a recreation center you can rent a locker. A locker is assigned to you and you can enter a code of four digits that will open and close your locker. You choose a code, put you stuff in de locker and have a nice time playing squash. When you return you find that you have forgotten your code. What now? Assuming you could enter any four numbers and because there are 10,000 different codes, you’ll have to enter 40,000 numbers to try all codes. Even using a computer this could take a while.

We can shorten this time considerably by using a De Bruijn Sequence. While this essay won’t focus on the applications of these sequences, the problem described above will be mentioned. This essay will focus on a specific kind of these sequences, functions that produce them and derive some of their properties. The question we try to answer is: Is there a generalized periodic shift register function that produces a De Bruijn sequence?

2 De Bruijn

We will start with figuring out exactly what De Bruijn sequences are and how we can find them.

2.1 De Bruijn Sequence

They get their name from the Dutch mathematician Nicolaas Govert de Bruijn. De Bruijn sequences are often denoted by $B(k, n)$. This is called a k-ary De Bruijn sequence of order n. It is a cyclic sequence of a given alphabet $A$ with size $k$ for which every possible subsequence (word) of length $n$ in $A$ appears as a subsequence of consecutive characters exactly once. [3] De Bruijn sequences have a number of interesting properties. One of these is that each $B(k, n)$ has length $k^n$. Another is that there are a limited number of distinct De Bruijn sequences for given $n$ and $k$.

An example of a De Bruijn sequence for $k = 2$ and $n = 3$, with alphabet $\{0, 1\}$, is 11101000. We can see that it has length $2^3$ and, when repeated, contains all words of length 3 and the given alphabet: $\{111, 110, 101, 010, 100, 000, 001, 011\}$.

An example of an application for such a sequence is the shortening of a brute-force attack on a pin-code protected vault. To check all the codes by hand would take $10^4 = 10000$ tries and thus 40000 buttons to press. Using a De Bruijn sequence with alphabet $\{0, 1, ..., 9\}$ and $n$ equal to 4 and a display the can shift the previous entries to the left, would reduce this number to $10^4 + 3 = 10003$.

2.2 De Bruijn Graph

To find De Bruijn sequences, we will need De Bruijn graphs. An $n$-dimensional De Bruijn graph of $k$ symbols is a directed graph, in which the overlap between the last $(n-1)$ symbols of one word and the first $(n-1)$ of another is represented. It has $k^n$ vertices, each representing one distinct word of the $k^n$ possible words of length $n$. The edges of the graph are directed. There exists an edge from one vertex to another if and only if the last $(n-1)$ symbols of the word represented by the first vertex are equal to the the first $(n-1)$ symbols of the other.
It is obvious that each vertex has exactly $k$ incoming and $k$ outgoing edges. Also each $n$-dimensional De Bruijn graph is the line digraph (a graph that represents the adjacencies between edges of G) of the $(n-1)$-dimensional De Bruijn graph with the same set of symbols[3]. This is displayed in Figure 1.

In blue we see the De Bruijn graph of dimension $k = 2$ and $n = 1$, in red the De Bruijn graph of dimension $k = 2$ and $n = 2$ and in green the one of dimension $k = 2$ and $n = 3$.

2.3 Finding a De Bruijn Sequence

To find De Bruijn sequences, we’ll be using graph theory. The De Bruijn sequences can be constructed by taking a Hamiltonian circuit of an $n$-dimensional De Bruijn graph over $k$ symbols. The problem with determining if there is a Hamilton path and finding it, is that this problem is NP-complete[5]. This means that there is no known general polynomial time algorithm to solve it.

Because of the line digraph property discussed in the previous section, there is an equivalency between the sequences you can find in a De Bruijn graph of dimension $n$ by finding Hamilton circuits and the sequences by finding Eulerian circuits of a $(n-1)$-dimensional De Bruijn graph. Because a De Bruijn graph is directed, we can find such circuits in polynomial time.

The only question that is left now is how we get an De Bruijn sequence from an Euler circuit in a De Bruijn graph. We will illustrate this with an example in Figure 2.

An Euler circuit is displayed in the numbering of the edges. The De Bruijn sequence is now
simply created by starting at a vertex, traveling through the Euler circuit and adding the last symbol of each identifier we pass. So in this case, if we take the right most vertex with identifier 00 as our starting vertex, we end up with the De Bruijn sequence 01011100.

3 Shift Registers

3.1 A Shift Register

A shift register is a cascade of flip-flops, sharing the same clock, which has the output of any one but the last flip-flop connected to the “data” input of the next one in the chain, resulting in a circuit that shifts by one position the one-dimensional “bit array” stored in it, shifting in the data present at its input and shifting out the last bit in the array, when enabled to do so by a transition of the clock input.[3]

In layman’s terms this means that the symbols of a certain sequence are each kept in a box. When a new item enters the first box, the contents of all boxes is given to its neighbor, and the content of the last box is given as output.

The shift registers we are interested in are those who have a word of a $n$-dimensional De Bruijn graph with $k = 2$ in the boxes. A function $F$ uses the values in these boxes to create the new input and the output should be a De Bruijn sequence. A diagram is displayed in Figure 3.
3 Shift Registers

3.2 Shift Register Functions

The function $F$ in the shift register is a boolean function. More specifically, a sum of products. The functions are of a form similar to $abcd + a + be$. In this function, a multiplication of variables is a boolean AND-operator. This operator returns a one if both variables are equal to one, and a zero otherwise. The addition in this function is an OR-operator. This operator returns a one if at least one of the variables is equal to one, and a zero otherwise. This is called a Sum-Of-Products boolean function.

We also can create another sort of functions called the Exclusive-OR-Sum-Of-Products functions. As the title suggests, these functions have an Exclusive OR ($\oplus$) instead of a regular OR ($+$). This Ex-OR counts the number of terms that take on the value of one and returns one if it is equal to one modulo two, and zero otherwise.

3.2.1 Karnaugh maps

The Karnaugh map is a method of creating the minimal sum of products from a truth table. In Figure 4 we see an empty Karnaugh map for a function of 4 variables.

![Figure 4: An empty Karnaugh map for a function of 4 variables.](image)

We produce the filled in Karnaugh map from the truth table by letting the 2 values above the columns be the first two variables from the truth table, the values in front of each row the third and fourth variable and filling in the value of the function in the intersection. We then can get the boolean function by extracting the product terms. These terms are found by encircling groups of ones in the map. The groups must be rectangular and must have an area that is a power of two (i.e. 1, 2, 4, 8...). The rectangles should be as large as possible without containing any zeroes. An example is shown in Figure 5.

This is the Karnaugh map for the De Bruijn sequence 011101100101000. For example, the red box translates to $AB$, because this property holds for all elements in the box. When we do this for all elements, we get the sum of products boolean function $\overline{AB} + \overline{ACD} + ABC + ABD$. In the remainder of this report we will use variables $x_0, x_1, x_2...$ instead of $A, B, C...$.

3.2.2 Exclusive-OR-Sum-Of-Products

We can also create the Exclusive-OR-Sum-Of-Products from a truth table. We will show this with an example. Suppose we have the truth table in Figure 6.

Bachelor Project
Figure 5: An example Karnaugh map for a function of 4 variables.

Figure 6: A truth table for the De Bruijn sequence 11000101.

We construct the initial function: $x_0(1 \oplus x_1)x_2 \oplus (1 \oplus x_0)x_1x_2 \oplus (1 \oplus x_1)(1 \oplus x_2) \oplus (1 \oplus x_0)x_1(1 \oplus x_2)$. We can write this as: $x_0x_1x_2 \oplus x_0x_2 \oplus x_0x_1x_2 \oplus x_1x_2 \oplus x_0x_2 \oplus x_0x_1 \oplus x_0 \oplus x_1 \oplus x_2 \oplus x_1x_2 \oplus x_0x_1x_2 \oplus x_0x_1 \oplus x_1x_2$. Then finally, because we the Exclusive OR is modulo two, we can simplify this to: $1 \oplus x_0 \oplus x_2 \oplus x_1x_2$. Which is our Ex-OR-SOP.

4 TutteMatrix

To eventually solve the problem of finding all Euler Circuits, we want to know how many there are. We do this by using the notion of the Tutte-matrix of a directed graph. Using a Tutte-matrix, we can calculate the number of spanning trees a matrix contains. We will need this to calculate the number of Euler circuits in a graph. The Tutte-matrix is defined as follows:

$$t_{ij} = \begin{cases} 
& \text{The number of arcs starting in } i \text{ and ending in } j \text{ multiplied with } -1, \text{ if } i \neq j. \\
& \text{The outdegree of vertex } i, \text{ if } i = j.
\end{cases}$$

Note that in this definition, the sum of all rows equals 0. Also, the Tutte-matrix of a De
5 Finding all spanning trees

Bruijn graph, has 2's on its diagonal. Trivially, it does not contain information about self-loops. This doesn’t matter however, because a self-loop can never be an edge of a spanning tree. Furthermore, a De Bruijn graph does not have multiple edges between the vertices.

We now use the Theorem of Tutte to compute how many spanning trees a graph contains:

**Theorem 4.1** The number of spanning trees with root \( i \) in a directed graph with Tutte-matrix \( T \) is equal to \( \det_{ii}(T) \), which is the subdeterminant of \( t_{ii} \) in \( T \).

A proof of this theorem can be found in the lecture notes of H. van Tilborg[4]. We will show this with an example found in Figure 8. Disregarding the self-loops we get the graph in Figure 7.

This gives the following Tutte-Matrix:

\[
\begin{pmatrix}
1 & -1 & 0 & 0 \\
0 & 2 & -1 & -1 \\
-1 & -1 & 2 & 0 \\
0 & 0 & -1 & 1
\end{pmatrix}
\]

The number of spanning trees with root 1 is then equal to:

\[
\begin{vmatrix}
2 & -1 & -1 \\
-1 & 2 & 0 \\
0 & -1 & 1
\end{vmatrix}
\]

Which is equal to 2 spanning trees.

5 Finding all spanning trees

One way to find all Euler circuits in a graph, is to start with all spanning trees of a graph. In Section 6.2 an algorithm will be put forth to create an Euler circuit from a spanning tree. The algorithm used in this essay to find all spanning trees is the one that was described by Takeaki Uno in his lecture notes: An Algorithm for Enumerating all Directed Spanning Trees in a Directed Graph[1]. In these notes the explanation of the algorithm can be found and the proofs of correctness. Here a short version is presented.

The idea behind the algorithm is to create a first spanning tree by a Depth First Search, and
then interchanging edges in a structured manner as to create all trees. It does this by defining
the children of each parent tree \( T_p \). These children \( T_c \) are created by removing an edge \( e \) from
the parent tree and replacing it by a valid edge from the original graph, such that the result
remains a spanning tree.

We start with a tree resulting from the Depth First Search \( T_0 \). We index the nodes from
this tree from 1 through \( m \) in order in which they were found. Because of the tree structure,
each node that is not the root, has exactly one arc leaving it and each arc can be uniquely
identified by the index of its tail node. We then define a parent-child relationship en using
this relationship we’re going to create a tree of spanning trees. The parent node of a directed
spanning tree \( T_c \neq T_0 \) is defined by the directed spanning tree \( T_p = (T_c \setminus f) \cup e \) where \( e \) is
the minimum index arc \( e \) in \( T_0 \setminus T_c \) (the arcs in \( T_0 \) that are not in \( T_c \)) and \( f \) is the arc of \( T_c \)
sharing its tail with \( e \). Since \( f \) shares its tail only one arc of \( T_c \), \( T_p \) is uniquely defined.

Now that it is known what the parent node of a child node is, we can find all the children of
a parent node. Takeaki UNO describes this process as follows:

Next we show the method of enumerating all children of a directed spanning tree \( T_p \). Let
\( v^*(T_p) \) be the minimum index among arcs in \( T_0 \setminus T_p \). Exceptionally, we define \( v^*(T_0) \) by \( \infty \).
Let us construct \( T_c \) by removing an arc \( e \) from \( T_p \) whose index is less than \( v^*(T_p) \) and adding
an arc \( f \neq e \) sharing its tail with \( e \). If \( T_c \) forms a tree, it is a child of \( T_p \) from the definition.
Conversely, in the case that \( T_p = (T_c \setminus f) \cup e \) is the parent of \( T_c \), the index of \( e \) is less than
\( v^*(T_p) \). Thus all children of \( T_p \) can be found by the above method. To find children, we deal
with only vertices and arcs whose indices are less than \( v^*(T_p) \). Hence we call them valid.[1]

To find these valid children we define back-arcs and non-back-arc. Back-arcs of a spanning
tree \( T \) are edges not in \( T \) (but existing in \( G \setminus T \)) for which the tail of the arc is an ancestor of
the head. Replacing these edges in the tree will not create a valid child, because then there
will exist a cycle in the graph. All other edges are non-Back Edges. Hence each child of \( T_p \)
is obtained by adding a valid non-back-arc \( f \) and removing an arc \( e \) sharing its tail with \( f \).
In the case of a De Bruijn Graph, there is only one choice to replace it with, because the
outdegree of each vertex is equal to two.

The algorithm is now as follows:

\[
\text{FindAllSpanningTrees}(G)
\begin{align*}
1 & \text{ Find } T_0 \text{ by a Depth-First-Search } \\
2 & \text{ Assign indices to each vertex } \\
3 & \text{ Classify arcs not in } T_0 \text{ into the back-arc set and the non-back-arc set.} \\
& \quad \text{ Sort them in order of their indices.} \\
4 & \text{ FindChildren}(T_0)
\end{align*}
\]

\[
\text{FindChildren}(G)
\begin{align*}
1 & \text{ Construct } T_c \text{ by adding } f \text{ and removing an arc } e. \text{ Save the spanning tree in an array. } \\
2 & \text{ List all valid non-back-arcs of } T_c \text{ in order of their indices.} \\
3 & \text{ FindChildren}(T_c)
\end{align*}
\]
5 Finding all spanning trees

We will illustrate this example by the De Bruijn Graph with dimension 3 in Figure 8. First we create the Depth First Tree in Figure 9. Then we assign indices to the vertices (Figure 10). Next, we create the back-arc set, which are the edges (1,3) and (2,3). Indeed, switching one of these edges will not result in a tree. The non-back-arc consists only of (3,4). We can now find the children of the Depth First Tree. In this case it has only one child. This child is created by switching the edge (3,2) with (3,4). The result is the tree in Figure 11.

We have now found all spanning trees of the graph. Note that in this process we have neglected the 2 loop arcs of the De Bruijn Graph. This is because a graph containing these edges can never contain \( m \) vertices and \( m - 1 \) edges and thus does not constitute as a tree.
Creating Euler circuits from spanning trees

To create a Euler circuit in a De Bruijn Graph, we will need the spanning trees of the graph. A spanning tree of a given graph is the set of the vertices and a subset of it’s edges, such that all vertices are covered by the subset of edges (each vertex is adjacent to at least one edge) and the subset of edges is a tree. A tree is a graph that is connected and has no circuits. Given such a tree, we can find a Euler circuit, if there is one. We will use the following theorem:

Theorem. Let \( G = (V, A) \) be an Euler graph, with a finite number of vertices. Let \( V = \{P_1, ..., P_n\} \) with \( n = |G| \). Let \( \sigma_i := \text{outdegree}(P_i) \) for \( i = 1, ..., n \). Then:

\[
P_E(V, A) = P_W(V, A, P_1) \prod_{i=1}^{n} (\sigma_i - 1)!
\]

where \( P_E(V, A) \) is the number of Euler circuits in graph \( (V, A) \) and \( P_W(V, A, P_1) \) is the number of spanning trees of graph \( (V, A) \) with root \( P_1 \) and all edges are pointed towards \( P_1 \).

This theorem involves a many-to-one mapping from a spanning tree to an Euler circuit. Furthermore, because in a De Bruijn graph the outdegree of all vertices is 2, we have that in De Bruijn graphs this mapping is one to one. We will construct an algorithm that maps the set of spanning trees of a De Bruijn graph onto the set of Euler circuits.

6.1 Euler circuit to spanning tree

This algorithm, and the one in the following section were devised by N.G. De Bruijn[2]. A proof of their correctness can be found in the lecture notes of H. van Tilborg [4]. Given an Euler circuit in an Euler graph, we construct the corresponding spanning tree as follows:

1. Fix one of the outgoing edges of \( P_1 \).
2. We now follow the given Euler circuit, starting with the fixed edge. For \( j = 2, ..., n \) we arrive at \( P_j \) \( \sigma_i \) times and leave them following the same amount of edges.
3. Let \( F \) be the set of directed edges for which holds that it was the last outgoing edge used for a vertex \( P_j \), \( j = 2, ..., n \). \( F \) consists of \( n - 1 \) edges and is a spanning tree for the Euler graph.
Creating Euler circuits from spanning trees

Figure 12: An Euler circuit in a De Bruijn graph of dimension 3.

Figure 13: The spanning tree resulting from the Euler circuit in figure 12.

We will illustrate this algorithm with an example. We will look at the example of a De Bruijn graph of dimension 3. An Euler circuit through this graph is given by a numbering of the edges in Figure 12.

In this figure, we fix edge number 1 of the vertex $P_1 = 00$. Following the Euler circuit we see that the set of edges that were the last outgoing edge for some vertex not equal to 00 is: $\{5, 7, 8\}$. This results in the spanning tree in Figure 13.

Note that this algorithm in a De Bruijn graph implies that an edge that is a selfloop (an edge from a vertex to the same vertex) for some vertex will not be part of a spanning tree. Such a vertex has two outgoing edges, and the selfloop will always be the first one visited, otherwise it is no longer reachable.

### 6.2 Spanning tree to Euler circuit

Given a spanning tree in an De Bruijn graph, we construct the corresponding Euler circuit as follows:

1. Color the edges in the graph that correspond to the spanning tree blue.
2. All nodes have now have one outgoing blue arrow, except the root node. Color one of the outgoing arrows of the root blue.
3. Starting at the root node travel through the graph, traveling each edge exactly once and taking the blue edges as a final choice. The traveled path is the Euler path corresponding to the given spanning tree.

Again we will illustrate this algorithm by an example. We will again look at a De Bruijn graph of dimension 3. A spanning tree of this graph is given in Figure 14. Coloring the edges
Figure 14: A spanning tree in a De Bruijn graph of dimension 3.

Figure 15: A coloring of the De Bruijn graph of dimension 3.

in the graph that correspond to the spanning tree and one of the outgoing arrows of the root blue, gives Figure 15. Finally, traveling through the graph, starting at the root node and taking each blue edge last, leads to the following Euler circuit in the graph, displayed in Figure 16.

Note that in a De Bruijn Graph, the blue edges completely determine the Euler circuit. In a graph were the outdegree of each vertex \( i \) is greater than 2, \( \sigma_i \), you have the choice between \( \sigma_i - 1 \) edges, before taking the blue one. So, as described in Section 6, one has \( \prod_{i=1}^{n} (\sigma_i - 1)! \) Euler circuits corresponding to each spanning tree of the graph.

7 Lijstje

<table>
<thead>
<tr>
<th>Dimension graph</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of vertices</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
</tr>
<tr>
<td>Number of edges</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>Number of Euler circuit</td>
<td>1</td>
<td>2</td>
<td>16</td>
<td>2048</td>
<td>6710864</td>
<td>144115188075855904</td>
<td>1.3E35</td>
</tr>
<tr>
<td>( z \log ) Euler circuit</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>11</td>
<td>26</td>
<td>57</td>
<td>120</td>
</tr>
</tbody>
</table>

Figure 16: An Euler circuit in the De Bruijn graph of dimension 3.
Around 1945 an engineer that was working on this subject discovered the table displayed here. He calculated it up to the double line and concluded the the number of Euler circuits in an n-dimensional De Bruijn graph was equal to $2^n - 1 - n$. Flye St. Marie (earlier in 1894) and, independently, N.G. De Bruijn did the same up to $n = 6$, and both validated and proved the formula.

In a java application I calculated the column of $n = 7$, which also validates the formula. The program was also able to calculate the number of Euler circuits for larger instances of $n$, but due to machine precision these calculations weren’t exact enough. Checking the last row of the table did give the right value for these larger instances of $n$ and thus it also confirms the formula.

8 De Bruijn Graphs: OR-operator

The next sections of this essay will look at the different De Bruijn graphs. We will analyze the De Bruijn sequences and the corresponding boolean functions. Initially we will use a shift register of $2^n$ variables (the length of a De Bruijn sequence of dimension $n$). Soon thereafter we will reduce the number of variables to $n$.

8.1 The Trivial Graph: $n = 1$

We shall first look at the trivial De Bruijn graph. This is displayed in Figure 17.

There is obviously only one Euler circuit in this graph: 01. If we have shift register of length 2 (with variables $x_0$ and $x_1$), then $x_0 = 0$ and $x_1 = 1$, and we can list all boolean functions that produce a recurring De Bruijn sequence. We have: $x_0$, $\overline{x_1}$ (NOT $x_1$), $x_0\overline{x_1}$ and $x_0 + \overline{x_1}$. We can see that we only need one of the variables to construct the sequence. Thus we can take a shift register of length 1 and then the function $x_0$ creates the De Bruijn sequence.

8.2 The De Bruijn Graph: $n = 2$

We shall now look at the De Bruijn graph for length $n = 2$. This is displayed in Figure 18.

In this case there is only one Euler circuit through this graph: 0011. This circuit has subsequences 00, 01, 11 and 10. These are all possible words of length 2. If we were to take a shift register of length 4, it has the truth table in Figure 19.

This gives as possible minimum register functions: $x_0$ or $\overline{x_2}$. Of course you could add garbage to the function. Examples are $x_0 + x_1$, $x_0\overline{x_2}$ or $x_0x_1 + x_0\overline{x_1}$. 
In this report we will focus on the non-garbage-functions. In the literature, these are defined as minimal boolean functions. We should also notice that the shift register function does not have to use all the variables. This is easy to understand if we look at the Euler circuit 0011. We simply need a function that repeats two zeroes and 2 ones. This can be accomplished by using a shift register of length 2 with a function $f$ (with variables $x_0$ and $x_1$) and putting a NOT-operator above the first variable. So $f(x, y) = \overline{x_0}$. When this function is applied to the created variables, we see that the De Bruijn sequence appears. We get exactly the same truth table:

<table>
<thead>
<tr>
<th>$x_0$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

These two variables can be any two of the $x_i, i = 0...4$, given that they do not contradict. In the previous table, we see that we cannot take $x_1$ and $x_3$, because $x_1 = 0$ and $x_3 = 1$ should give one and zero in different composition of the sequence. This cannot be accomplished by using a shift register that only uses those two variables. All the other combinations of 2,3 or 4 variables have a unique image. Again, in this report, we will focus on functions that use the last $n$ variables.

8.3 The De Bruijn Graph: $n = 3$

We continue to the 3-dimensional De Bruijn Graph. We’ve drawn the graph in Figure 20.
We know from the theorem discussed in Section 7 that the number of Eulerian circuits in a De Bruijn Graph of dimension \( n \) is equal to \( 2^{2^{n-1}-n} \). For \( n = 3 \) the number of Eulerian circuits is 2. Using the java application, we find the cycles: 11010001 and 11000101. We will call these case 1 and case 2 respectively. We will first examine case 1:

\[
\begin{array}{cccccccc}
x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & f \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
\end{array}
\]

It is easily seen that the register function \( x_0 \) can correctly give the next value. But, as is mentioned in previous sections, we wish to use only the last \( n \) variables.

\[
\begin{array}{ccc}
x_0 & x_1 & x_2 & f \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
\end{array}
\]

We have now build a register, depending only on the last three variables and we get the minimal boolean function \( x_0x_1 + x_0x_2 + x_0x_1x_2 \). Because each next iteration is a shift of the previous, we get a correct register function. Is it possible to use less than 3 variables? It is not. There would be a cycle of 4 in the De Bruijn sequence and this is not true for the case we were examining: 11010001.
We can do the same for the second De Bruijn cycle in a 3-dimensional De Bruijn graph: 11000101. Again if we have a shift register of length 8, it is again easily seen that the register function $x_0$ can correctly give the next value. We will again look at the last three variables.

<table>
<thead>
<tr>
<th>$x_0$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

This gives a shift register function of the form $x_0x_2 + x_0x_1 + x_0\overline{x_1}x_2$.

In conclusion, for $n = 3$ we can use the shift register functions: $x_0x_1 + x_0x_2 + x_0x_1\overline{x_2}$ and $x_0x_2 + x_0x_1 + x_0\overline{x_1}x_2$. These functions have the same structure. This is not illogical, because the sequences are alike: 11010001 and 11000101. The only difference is that in the first sequence 110 leads to a one and 010 leads to a zero, and in the the second sequence the reverse is true. This reverse is facilitated by the tiny difference in the functions.

There also is an internal structure that is interesting. We have $x_0(x_1 + x_2) + x_0x_1\overline{x_2}$ for the first function and the second inverts $x_1$ and $x_2$. Both consist of two terms. First we have $x_0$ multiplied with the sum of the other variables (whether inverted or not) and secondly $x_0$ multiplied with the inverted versions of the variables that were in the first term of the function.

8.4 The De Bruijn Graph: $n = 4$

This graph has 16 Euler circuits. Just as was the case with $n = 3$, we can make truth tables and derive the function. With my java application, the computer does this work for us. The circuits and their corresponding shift register functions are displayed in Figure 21.

The first thing we notice is that, like with $n = 3$, there are functions with $x_0$ multiplied with the sum of the other variables and $x_0$ multiplied with the other variables. We have for example for the De Bruijn sequence 0111100101101000 a function $x_0x_1 + x_0x_3 + x_0x_2 + x_0x_1x_2x_3$. Following this trend we could think that for an 5-dimensional De Bruijn sequence, there is also a function of this form. For example: $x_0x_1 + x_0x_3 + x_0x_2 + x_0x_4 + x_0x_1x_2x_3x_4$. By simply checking all possibilities, this turns out to not be the case.

Another structure that springs to eye is that of the functions with 4 terms of degree 3, and one of degree 4. Both functions that were found in the 3-dimensional graph had the same structure. All these functions have a fully inverted term and a term that has the same variables, but only $x_0$ is inverted. I could, however, not find more consistencies in these functions and thus could not create a general type of function, that could work for higher dimensions.

Another thing that we should notice is that the minimal boolean functions for $n = 3$ have 4 terms if the word containing only zeroes is adjacent to the word containing only ones (so 00001111 or 11110000) and 5 terms otherwise. It seems ‘simpler’ to create the sequence with
a boolean function if this adjacency of words occurs. We can intuitively see this when we draw a Karnaugh map. When for example we know that the word 0000 is followed by 1111, we know that a subsequence of the De Bruijn sequence is 1000011110. We can thus fill in some values in the Karnaugh map (Figure 22).

As seen in section 3.2.1, the larger the rectangles that can be drawn, the simpler the functions are. We also know that each row has exactly two ones. Because we know for sure that three of the ones are adjacent, there is a larger chance we can encompass all ones in only 4 rectangles. We shall see in the 5-dimensional De Bruijn graph, that this is not only the case for sequence where the word with only zeroes and the word with only ones are adjacent.
8.5 The De Bruijn Graph: $n = 5$

The 5-dimensional De Bruijn graph has 2048 Euler circuits and thus 2048 boolean functions. These are obviously too numerous to discuss all of them. Some of the results we will discuss. The first thing is to check some of the results of the lower dimensions. We checked in the 4-dimensional graph, if we could make a function with a $x_0$ term and a $x_0$ term. This turned out to be nonexistent. This is again true for the 5-dimensional functions.

In the 4-dimensional graph we also saw functions with one term of dimension 4 and the remainder of the terms of dimension 3. These sorts of functions are also found in dimension 5. An example of such a function is:

$$x_0 x_1 x_2 x_4 + x_0 x_1 x_2 x_4 + x_0 x_1 x_2 x_4 + x_0 x_1 x_2 x_4 + x_0 x_1 x_2 x_4 + x_0 x_1 x_2 x_4 + x_0 x_1 x_2 x_4.$$  There are a few nice thing about this function. The first thing is that the first 7 terms contain only 4 variables. First we have a term that is entirely inverted, the 3 in which $x_0$ is inverted and one of the other three and finally 3 in which $x_0$ is not inverted and two of the three remaining variables are inverted. The eighth term consists of all variables, with only the missing variable of the first 7 terms inverted. The ninth term consists of an inverted $x_0$ multiplied with three of the other four variables, where $x_1$ is omitted if it was in the first 7 terms and $x_2$ is omitted otherwise.

The beauty of this structure is that this function exists for each first term of four variables, as long as it contains $x_0$. The functions also exists for some mixes of these functions. Then the first 7 terms do not contain the exact same variables.

In the 4-dimensional De Bruijn graph we also looked at functions of $n = 4$ terms. These are also present in the 5-dimensional graph. But whereas with $n = 3$ all functions had 3 terms and with $n = 4$ 50% of the functions had 4 terms, in dimension number 5 the percentage of functions with 5 terms has dwindled to about 1%. Most of these indeed have the word of only zeroes and the word of ones of only zeroes. I have however found a counterexample, where a function of exactly 5 terms has a corresponding De Bruijn sequence where these words are not adjacent. The De Bruijn sequence 010010110011111001111010110000 has the De Bruijn function $x_0 x_1 x_3 x_4 + x_0 x_2 + x_0 x_2 x_4 + x_0 x_2 x_4 + x_0 x_1 x_2$. I have however not been able to find structure in these functions.

8.6 Finds

Looking at the results of the previous section, I will now supply the proof for some regularities.

**Theorem 8.1** All $n$ variables of the shift register are in all valid De Bruijn sequence functions of dimension $n$ for all $n > 2$.

**Proof:** As is seen in Section 8.3, this statement in true for $n = 3$. Suppose now that the statement is true for some $n > 2$. Then it is also true for dimension $n + 1$. For if it was possible to create a function, in which one variable was omitted, then the variables of its truth table would consist of twice the truth table of a graph of dimension $n$. Then there are two options. The first option is that two sets of the same input give different results, in which case no boolean function is possible. The other option is that all sets of the same input yield the same result. In this case, the resulting sequence is periodic, with a smaller period than $2^n$, and thus can never be a De Bruijn sequence.

That the statement does not hold for a De Bruijn graph of dimension 2 is shown in Section 8.2. Here a shift register function is found for a De Bruijn sequence, using only one variable.
Theorem 8.2  Valid De Bruijn sequence functions of dimension $n$ always contain a term with only inverted variables.

Proof: The proof for this is easy. If the function didn’t contain such a term, then the word consisting of only zeroes, would always lead to another zero. We’d then get infinite zeroes and thus the function could never be a De Bruijn shift register.

Theorem 8.3  Valid De Bruijn sequence functions of dimension $n$ never contain a term without inverted variables.

Proof: Again, the proof is easy and consists of a similar reasoning as the proof of Theorem 8.2. If the function did contain such a term, then the word consisting of only ones, would always lead to another one. We’d then get infinite ones and thus the function could never be a De Bruijn shift register.

Theorem 8.4  Valid De Bruijn sequence functions of dimension $n$ never contain a term without an instance of $x_0$.

Proof: Suppose there would be a term that did not contain an instance of $x_0$. We will prove that this term is obsolete. Suppose it is not an obsolete term. Then there is a word $x_0, x_1...x_n$ from the truth table in which case this term will be a one, where the other terms are all zeroes. But then this term will give a one in both the case that $x_0 = 0$ and $x_0 = 1$ (the other inputs remain the same). This means that the word $x_1...x_n, 1$ appears twice in the resulting sequence and thus the sequence is periodic. This would not be a De Bruijn function and thus the term is obsolete.

There is also never a term, with only inverted variables and a non-inverted instance of $x_0$. This is because this term would give a one in the case of the input of a one followed by $n - 1$ zeroes. If this happens, the word containing only zeroes can never be reached.

For a similar reason, there is always a term in the boolean function with an inverted $x_0$ and the instances of the other variables are all non-inverted. This is because the word consisting of a zero followed by $n - 1$ ones, must always be followed by another one. Otherwise the word consisting of only ones can’t be reached. We proved earlier that a boolean function without inverted variables does not exist. So the word with only ones can only be reached by a term as described.

8.7 Conjectures

Conjecture 8.5  Valid De Bruijn sequence functions of dimension $n$, $n > 2$, never have less than $n$ terms.

This conjecture is supported by the functions found for dimensions 3 through 6. However, I have not been able to supply a proof.

Conjecture 8.6  In the set of valid De Bruijn sequence functions of dimension $n$, $n > 2$, there always is a function of exactly $n$ terms.

Again, this conjecture is supported by the functions found for dimensions 3 through 6. However, I have not been able to supply a proof.
9 De Bruijn Graphs: eXOR-operator

9.1 The De Bruijn Graph: \( n = 1 \)

We shall again look at the trivial De Bruijn graph. This was displayed in figure 16. Using the Exclusive OR-operator. We get the function \( 1 + x_0 \). This is the equivalent of the \( x_0 \) function in section Section 8.1. So, changing the operator does not change much for the trivial graph.

9.2 The De Bruijn Graph: \( n = 2 \)

Now we will look again at the De Bruijn graph for length \( n = 2 \). This was displayed in figure 17. To illustrate the procedure, we again display the truth table:

<table>
<thead>
<tr>
<th>( x_0 )</th>
<th>( x_1 )</th>
<th>( f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

We create the function: \((1 \oplus x_0)(1 \oplus x_1) \oplus (1 \oplus x_0)x_1\). We simplify this to \( 1 \oplus x_0 \oplus x_1 \oplus x_0x_1 \oplus x_1 \oplus x_0x_1 \), by removing the parenthesis and then simplify it by removing terms that result in zero, modulo two: \( 1 \oplus x_0 \). Again, this functions matches the one of Section 8.2.

9.3 The De Bruijn Graph: \( n = 3 \)

We continue to the 3-dimensional De Bruijn Graph. We’d drawn de graph in figure 18. We knew this graph had two De Bruijn sequences 11010001 and 11000101. This first sequence gives an Exclusive OR function \( 1 \oplus x_0 \oplus x_1 \oplus x_1x_2 \). The second sequence gives an Exclusive OR function \( 1 \oplus x_0 \oplus x_2 \oplus x_1x_2 \). These functions differ significantly from the functions found in Section 8.3. They do, however, show just a small difference from each other.

9.4 The De Bruijn Graph: \( n = 4 \)

As was shown in section Section 8.4, this graph has 16 Euler circuits. We list the circuits and their corresponding shift register functions in Figure 23.

9.5 Finds

Looking at the results of the previous section, I will now supply the proof for some regularities.

**Theorem 9.1** All valid shift register functions have an even number of terms.

**Proof:** The function should take a word of only ones produce a zero (otherwise you’d get a constant sequence of ones). Suppose the shift register function has an odd number of terms. Using this shift register function, the word of only ones would then produce a one for each term (which were in an odd number) and thus the function would produce a one. This is a contradiction and the theorem is thus true.
### De Bruijn sequence
<table>
<thead>
<tr>
<th>De Bruijn sequence</th>
<th>Shift Register Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>011100101101000</td>
<td>$1 \oplus x_0 \oplus x_1 \oplus x_1 x_2 \oplus x_1 x_3 \oplus x_1 x_2 x_3$</td>
</tr>
<tr>
<td>010111001101000</td>
<td>$1 \oplus x_0 \oplus x_1 \oplus x_3 \oplus x_1 x_2 \oplus x_1 x_2 x_3$</td>
</tr>
<tr>
<td>0110010111101000</td>
<td>$1 \oplus x_0 \oplus x_1 \oplus x_1 x_3 \oplus x_2 x_3 \oplus x_1 x_2 x_3$</td>
</tr>
<tr>
<td>0101100111101000</td>
<td>$1 \oplus x_0 \oplus x_1 \oplus x_3 \oplus x_2 x_3 \oplus x_1 x_2 x_3$</td>
</tr>
<tr>
<td>011011100101000</td>
<td>$1 \oplus x_0 \oplus x_1 \oplus x_2 \oplus x_1 x_3 \oplus x_1 x_2 x_3$</td>
</tr>
<tr>
<td>010100110111000</td>
<td>$1 \oplus x_0 \oplus x_3 \oplus x_1 x_2 x_3$</td>
</tr>
<tr>
<td>0100110101111000</td>
<td>$1 \oplus x_0 \oplus x_2 \oplus x_3 \oplus x_1 x_3 \oplus x_1 x_2 \oplus x_2 x_3 \oplus x_1 x_2 x_3$</td>
</tr>
<tr>
<td>0110100101111000</td>
<td>$1 \oplus x_0 \oplus x_2 x_3 \oplus x_1 x_2 x_3$</td>
</tr>
<tr>
<td>011101101001000</td>
<td>$1 \oplus x_0 \oplus x_1 \oplus x_1 x_2 \oplus x_1 x_2 x_3$</td>
</tr>
<tr>
<td>011110101011000</td>
<td>$1 \oplus x_0 \oplus x_3 \oplus x_1 x_3 \oplus x_1 x_2 \oplus x_1 x_2 x_3$</td>
</tr>
<tr>
<td>011111011001000</td>
<td>$1 \oplus x_0 \oplus x_1 \oplus x_2 \oplus x_1 x_3 \oplus x_1 x_2 \oplus x_2 x_3 \oplus x_1 x_2 x_3$</td>
</tr>
<tr>
<td>0101001110101000</td>
<td>$1 \oplus x_0 \oplus x_3 \oplus x_1 x_2 \oplus x_2 x_3 \oplus x_1 x_2 x_3$</td>
</tr>
<tr>
<td>011011100101000</td>
<td>$1 \oplus x_0 \oplus x_3 \oplus x_1 x_3 \oplus x_1 x_2 \oplus x_1 x_2 x_3$</td>
</tr>
<tr>
<td>011110100101000</td>
<td>$1 \oplus x_0 \oplus x_1 x_2 \oplus x_1 x_2 x_3$</td>
</tr>
</tbody>
</table>

Figure 23: Exclusive-OR shift register functions for De Bruijn sequences of dimension $n = 4$.

**Theorem 9.2** All valid shift register functions contain an instance of $1$.

**Proof:** The function should take a word of only zeroes produce a one. Suppose the shift register function has no instance of one. Then the terms of the functions would take on the value of zero if the input is the word that contains only zeroes. The function would thus produce a one. This is a contradiction and therefore the theorem is true.

**Theorem 9.3** All valid shift register functions contain an instance of $x_0$.

**Proof:** The word 1000 should always produce a zero, because otherwise the word of only zeroes can never be reached. Because we have proven that each valid function contains a one, there must be another term that produce a one when applied to the word 1000. This can only be the $x_0$ term.

### 9.6 Conjectures

**Conjecture 9.4** Except for the instance of the single term $x_0$, valid shift register functions do not contain other terms that contain the variable $x_0$.

As this is a conjecture, I cannot provide a proof. An intuition can be given by looking at the graph theory. A word in the De Bruijn sequence can be seen as the vertex of the graph you are in and information about the last node that was visited. This information is contained in $x_0$, and the remainder of the word is the identifier of the vertex. If you know from which vertex you came, it is determined to which node you will travel. Once you know this you can determine the identifier of this vertex by looking at the remaining variables.

**Conjecture 9.5** All valid shift register functions for a De Bruijn sequence of dimension $n$, $n > 2$, always contain a term that contains all variables except for $x_0$.

This conjecture is true for $n = 3$ and $n = 4$, but I have not been able to supply a proof.
10 Conclusion

The question this essay tried to answer was: Is there a generalized periodic shift register function that produces a De Bruijn sequence? No conclusive answer was found. I have not been able to prove that such a function does not exist, nor have I been able to find one. In the Sum-of-Products boolean functions, there were some patterns within the dimensions, but no interdimensional relations were found. This result was similar for the Exclusive-OR-Sum-of-Products functions.

For the Sum-of-Products boolean functions I did prove that some properties were true for all functions. Also some properties were seen in all functions, but I have not been able to supply a proof. Again, the same holds for the Exclusive-OR-Sum-of-Products functions.

In conclusion I would say that even though the main question of this essay wasn’t answered, I accomplished some good results. With better computers and some strong pattern recognition software this problem might be solved in the future.

A Bronnen


