FINITE STATE CODES AND GENERALIZED DE BRUIJN SEQUENCES

Thesis by

Lada Popović

In Partial Fulfillment of the Requirements
for the Degree of
Electrical Engineer

California Institute of Technology
Pasadena, California

1991

(Submitted September 28, 1990)

Acknowledgements

I would like first of all to thank my advisor, Professor Robert McEliece, for his constant guidance and encouragement during the last few years. The main impetus for this thesis came from our weekly conversations, in which he gave generously of his ideas, insight and experience.

I thank Dr. Posner and Dr. Kechris for serving on my supervising committee.

Caltech was a stimulating, although sometimes demanding, environment in which to live and work. The courses which I took proved invariably challenging and mostly well-lectured and enjoyable. I am grateful to the professors who gave them.

I would like to thank the members of Dr. Paul Siegel's group at IBM for useful discussions and financial support during my second year. Special thanks are due to Dr. Brian Marcus, who took the time to introduce me to Symbolic Dynamics and to help me in my thesis research.

Lastly, I give my thanks to all those who made my life here enjoyable—family and friends scattered throughout the world, and especially to Vojkan, for being (among other things) my immediate, non-stop support system.

Abstract

This thesis is divided into two self-contained chapters. The first chapter is a study of Finite State Codes. We address the question of unique decodability (UD) of the labelled state diagram which defines the finite state machine–part of the encoder for these codes. Bounds on the number of labels and resolving length are given for regular state diagrams. We show that minimal convolutional encoders can be used as finite state machines to produce UD labellings which are often optimal with respect to number of labels and resolving length. We finish with some constructions which demonstrate how block and convolutional codes can be integrated to obtain finite state codes of high rate and large free distance.

The second chapter introduces and analyzes a family of shift-register sequences which generalize the well-known de Bruijn sequences. A (q, ν) de Bruijn sequence is a periodic sequence of letters from a q-ary alphabet such that any given ν-tuple appears exactly once as a 'window' in a period. We introduce Generalized de Bruijn Sequences, which involve several sequences in parallel and an irregularly shaped window, plus the requirement that every possible window content should appear n times per period. The existence of such sequences is proved by construction, and a formula for their number is derived. The classical de Bruijn sequences can then be regarded as a special case of this new family.

Table of Contents

Chapter 1

Labelling the State Diagram of a Finite State Code

1.1. Introduction

The two main paradigms in the theory of error-correcting coding are block coding and trellis coding.

In *block* coding, the encoder receives k input symbols at the time, and converts them into n output symbols. The n-tuples which can be obtained at the output of the encoder are called codewords, the set of all codewords constituting the (block) code. The error-correcting capability is measured by the (Hamming) *minimal distance*—minimal number of places where two codewords differ. The succesive k-blocks are encoded independently—the previous inputs do not affect the encoding of the current input.

In *trellis* coding, the encoder similarly receives k input symbols, but the way in which they are encoded into n output symbols depends on the history of previously received input blocks. The significant history is usually constrained to some finite number of previous blocks, and hence can be expressed via a finite number of *states* of the encoder. Each state represents a history, and the input causes the encoder to change the current state and output an n-tuple. The functioning of the encoder can thus be conveniently represented by a state diagram or a trellis (hence the name), labelled by inputs and outputs. The codewords are the semi-infinite output sequences obtained by reading the output labells off every semi-infinite path in the trellis. The error-correcting capability is measured by minimal Hamming distance between two (semi-infinite) codewords—usually referred to as the *free distance* of the code.

The theory of block codes is well developed and rich in mathematical content. Much less has been done with respect to trellis codes. In fact, while block codes tend to have a lot of structure and are often constructed using complicated algebraic and combinatorial machinery, most of the good trellis codes have been found by computer search†. However, a class of linear trellis codes known as *convolutional codes* was found to perform suprisingly well in many applications, and a considerable body of literature has grown which describes and analyses these codes.

Inspired by work of Ungerboeck ([Un]) on channel coding, Pollara, McElliece and Abdel-Ghaffar have defined in [PMA] the *finite state codes*, which seek to combine the block- and trellis-coding approach. One of the attractions of this idea is that it promises new uses for the well-known, powerful block codes, so that it builds on the existing knowledge, rather that starting from scratch. Some of the codes constructed in [PMA] to illustrate the idea have been shown to have the free distance as large as possible.

In this chapter we develop their work further. In Section 2, the finite state codes are introduced and issues related to decoding and distance properties are discussed. This leads to a graph labelling problem which is addressed in Section 3. In Section 4, we discuss the use of convolutional codes in the context of finite state codes. Finally, in Section 5 some code constructions are proposed and compared

† Recent applications of symbolic dynamic to coding provide a mathematical basis for analysis of trellis codes. However, the codes which were produced using this method were as a rule source codes, not error-correcting codes.

with respect to rate, free distance and complexity.

1.2. Finite State Codes

Following [PMA] we define an $(n, k, \nu = \log_q v)$ *finite state code.* This section is mainly a review of results from [PMA].

The input is any semi-infinite sequence of symbols from the alphabet $\mathcal{A} = \{0, 1, \cdots, q-1\}$. The encoder maps this sequence into a codeword—again a semi-infinite sequence of symbols from $\mathcal{A}$. Two distinct output sequences differ by at least d_{free} symbols, thus providing the error-correcting capability. The mapping performed by the encoder must of course be 1–1, and we shall later find it necessary to impose additional restrictions.

The encoder consists of two parts (Figure 1.1):

- The finite state machine (FSM) is represented by an out-regular state diagram—a directed graph whose vertices represent the states, and edges the allowed transitions between the states (the number of states is denoted by v). From each vertex emanate q^{k_1} edges, corresponding to all possible inputs. The edges are labelled from the set $\mathcal{L} = \{0, 1, \cdots M-1\}$ of possible outputs of the finite state machine.
- The block encoder can encode according to any of the M disjoint (n, k_2) block codes $C_2^{(0)}$, $C_2^{(1)}$, $\cdots$, $C_2^{(M-1)}$.

The input stream is broken into nonoverlapping segments of length $k = k_1 + k_2$. The first k_1 blocks drive the finite state machine, determining which of the q^{k_1} edges outgoing from the current state to use for transition to the next state. The label on this edge determines the block code according to which the remaining k_2 symbols are encoded into the current n-segment of the output stream.

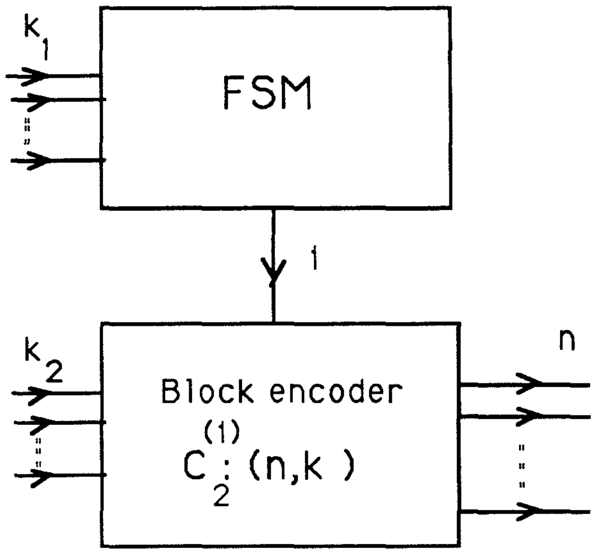


Figure 1.1.

Example 1.1. Let $\mathcal{A} = \{0,1\}$, $\mathcal{L} = \{0,1,2,3,\}$. The $(3,2,\log_2 3)$ finite state encoder appears in Figure 1.2. The finite state machine operates according to the rule:

- if the input is 0, go to the same state,
- if 1, go to the next larger state modulo 3.

The block codes $C_2^{(i)}$ are the repetition code ($C_2^{(0)}$) and its cosets.

The finite state machine and the entire encoder may be represented by trellises. The ones for Example 1.1 appear in Figure 1.3. The input sequences are represented by all semi-infinite paths in the overall trellis (starting at s_0), the output sequences by the corresponding edge labels. The Hamming distance between two paths is the Hamming distance between their label sequences.

Free distance d_{free} of a finite state code is the smallest Hamming distance between two nonidentical paths in the trellis. Let d_2 be the smallest among the

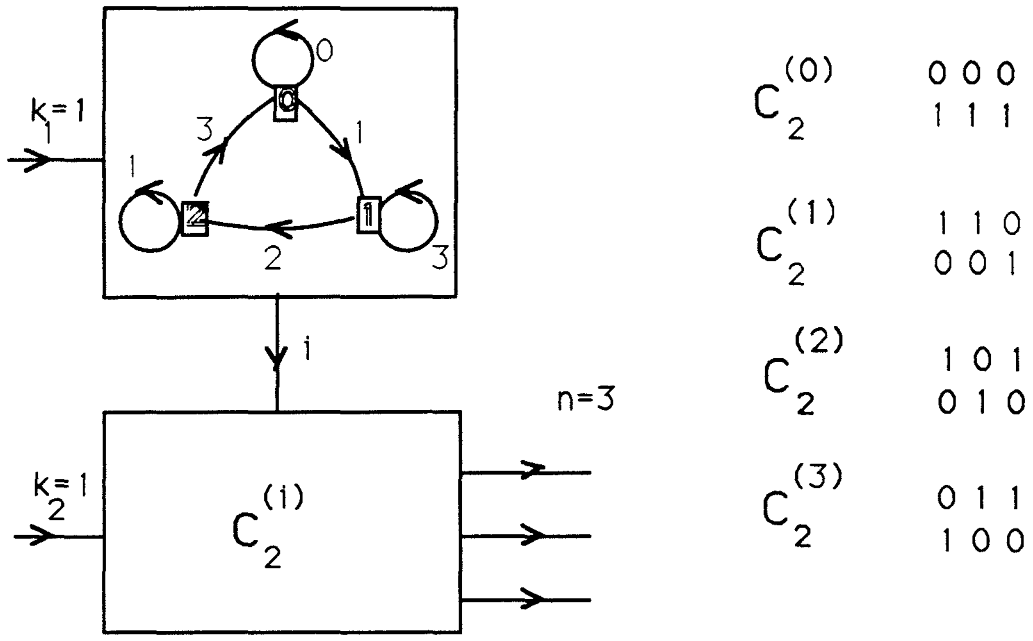


Figure 1.2.

minimal distances of codes $C_2^{(i)}$, and let d_1 be the minimal distance of the code $C_1 = \bigcup_{i=0}^{M-1} C_2^{(i)}$. Clearly the free distance is less then equal d_2 because we can have two paths with identical states which differ only in one edge (Figure 1.4). The distance between them can be d_2 because $e_1^{(1)}$ and $e_1^{(2)}$ are labelled by codewords of the same code $C_2^{(i)}$.

Before placing a lower bound on d_{free}, we require that the labelling of the state diagram be *nonsingular*: for any state, all the outgoing edges have different labels, and the same holds for the incoming edges. Let $e_1^{(1)}$ and $e_1^{(2)}$ be the first edges where the two compared paths differ. There are two possibilities, depending on whether $e_1^{(1)}$ and $e_1^{(2)}$ end in the same state or not.

When they end in the same state, the distance is at least d_2 (Figure 1.4).

If $e_1^{(1)}$ and $e_1^{(2)}$ lead to different states, by nonsingularity, the labels belong to

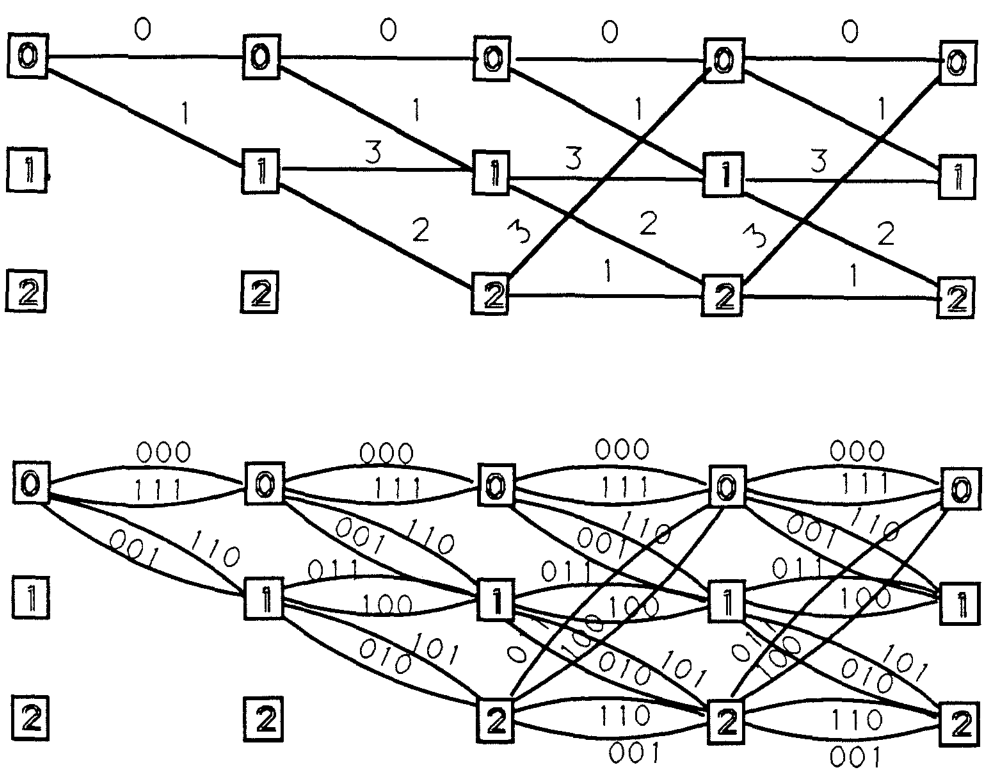


Figure 1.3. The trellis of the FSM and the overall trellis for Example 1.1

different codes $C_2^{(i)}$ and so are different codewords of C_1, at distance at least d_1. When the two paths merge again, the incoming edges to the common state will by nonsingularity also have different labels. Thus the distance between the two paths is at least $2d_1$ (Figure 1.5).

What if the two paths never merge again? We need not be concerned with this case because the condition of *noncatastrophicness*, which we shall impose shortly, ensures that two such paths always have infinite Hamming distance between them.

So, we have

$$\min(d_2, 2d_1) \leq d_{\text{free}} \leq d_2. \tag{1.1}$$

We shall improve this bound in Section 1.4.

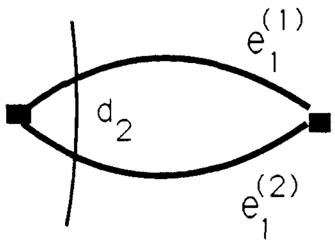


Figure 1.4.

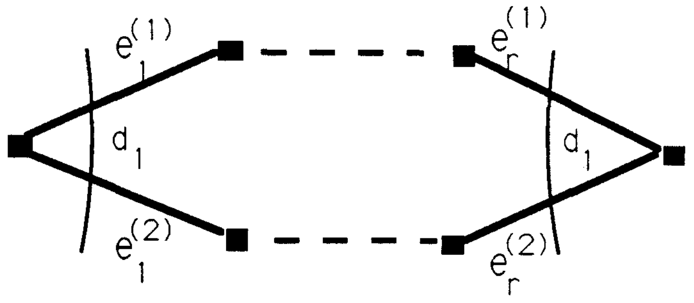


Figure 1.5.

The philosophy behind the design of finite state codes now becomes apparent: d_2 is typically large compared to d_1. The $C_2^{(i)}$ are often obtained from one block code C_2 as its cosets. We hope to keep the free distance close to d_2, while the rate has increased from k_1/n to $(k_1 + k_2)/n$, compared to using C_2 alone. This improvement is payed for by the increased complexity in encoding/decoding.

We now address the issue of *noncatastrophicness* which was mentioned briefly before. Noncatastrophicness of the encoder means that no two input sequences which differ in infinitely many places are mapped into output sequences which differ in finitely many places. Otherwise finite number of channel errors may produce infinite number of decoder errors.

We shall actually put a stronger restriction, which is anyhow necessary for all practical decoders. We require the code to be *sliding-block decodable*, which means: given outputs at times $-m$, $-m+1$, $\cdots$, 0, 1, $\cdots$, $K-m-1$, it is always possible to determine the input at time 0.

Hence we require the labelled state diagram to be sliding-block decodable (replace, in the above definition, 'inputs' by 'edges' and 'outputs' by 'labels'). But since the labelling is also nonsingular, if we can reconstruct one edge in the K-

length block, we can also reconstruct the others. Thus we arrive at the following condition:

The state diagram of the finite state code must be *uniquely decodable*, meaning that given a sequence of K labels there is at most one path of length K (sequence of K successive edges) in the state diagram which produces those labels.

In Example 1.3, K must be at least three, because there are two paths producing the label sequence 1 3, for instance. The reader may check that three labels are always enough to reconstruct the corresponding path.

Remark 1

The finite state machine may be viewed as an encoder for a trellis code, with the output alphabet $\mathcal{L}$ and output blocks of length 1. We shall see later that convolutional encoders may be used in place of the FSM—this is the synthesis of trellis and block codes that was promised in the introduction.

Remark 2

Although the decoder for the code of Example 1.1 is sliding block, the encoder is not: it is not possible to specify some fixed number of previous input blocks which will always determine the current state. In fact, the rule is: the current state is equal to the number modulo 3 of 1's seen in the input so far from the beginning of the sequence. Thus the state 0 corresponds to histories 0, 111, 0111, 1011, 1101, 1110, $\cdots$.

1.3. A Problem in Graph Labelling

We shall consider directed graphs with v vertices and edges given by a relation on the set of vertices. So, if the set of vertices is $S = \{s_1, s_2, \cdots, s_v\}$, the edges are given by a subset of $S \times S$, i.e., by certain ordered pairs of vertices. We say that an edge (s_1, s_2) starts in vertex s_1 and ends in vertex s_2.

We define a *path* of length l to be a succesion of l edges, such that the starting vertex of each edge is the ending vertex of the previous edge. Edges can be repeated. A closed path is called a *circuit.*

A path is said to connect vertex s_1 to vertex s_2 if those are respectively the starting vertex of the first and the ending vertex of the last edge in the path. We shall restrict our attention to graphs such that for any given two vertices, there is a path connecting the first vertex to the second. Such graphs are said to be *strongly connected.*

To motivate the problem, let us look at a simple example.

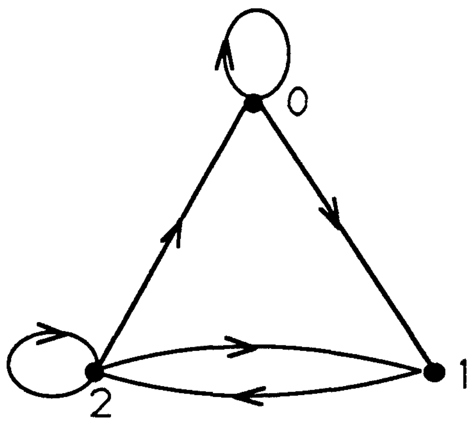


Figure 1.6.

Example 1.2. We want to *label* the edges of the graph on Figure 1.6 in such a way that, given a list of edge labels in a sufficiently long path, we can decide what the path was. We can get a trivial solution using a different label for each edge. The challenge is to use the minimal possible number of labels. Consider the labelling in

Figure 1.7, which uses four labels $\{a, b, c, d\}$. Given the list $a\,b\,b\,d$ we can deduce that the path was $(1,2)\,(2,2)\,(2,0)\,(0,0)$. But with the list: $a\,b\,c\,a\,b\,c \cdots\cdot$ (and so on, repeating $a\,b\,c$) we have a problem: no matter how long the list, we cannot decide whether the path was $(1,2)\,(2,0)\,(0,2)\,(2,1)\,(2,0)\,(0,1), \cdots\cdot$ or $(1,2)\,(2,2)\,(2,1)$ $(1,2)\,(2,2)\,(2,1) \cdots\cdot$.

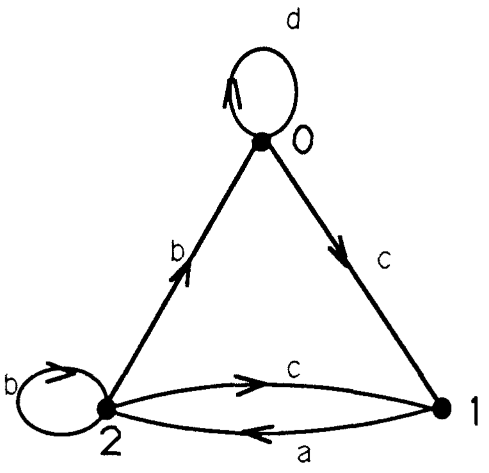


Figure 1.7.

In Figure 1.8, only three labels are used, but every path of three or more edges yields a different label sequence. Two labels are not enough for a decision: for instance, $(1,2)\,(2,2)$ and $(2,2)\,(2,2)$ both give the same label list $b\,b$.

In the attempt to label this graph, several questions arose:

- what is the minimal number of labels needed;
- what is the minimal length of the label list which is sufficient for reconstructing the path;
- what algorithm should be used for labelling a given graph.

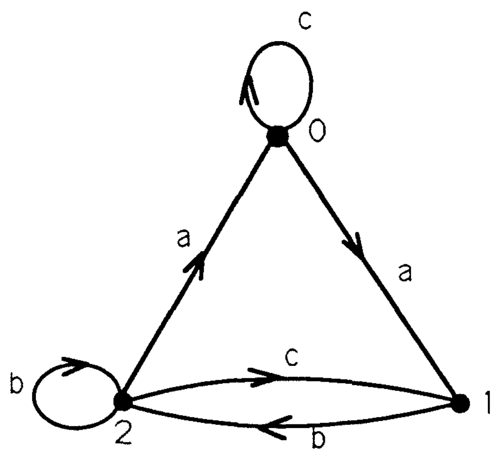


Figure 1.8.

We shall consider these questions in this and the next section. To begin, we formalize the notions which were casually introduced before.

Two paths are *label-indistinguishable* if they produce the same list of labels.

Definition 1.3. An edge labelling of a graph is *uniquely decodable* (UD) iff there is an integer K such that any two label-indistinguishable paths of length K or more are identical. The smallest such K is called the *resolving length* of the labelling, K_0.

In other words, any sufficiently long path can be recovered from its label sequence. We shall also use the expression 'a UD state diagram' for a state diagram whose labelling is UD.

Definition 1.4. An edge labelling is *nonsingular* iff all the edges that go out of the same vertex are labelled differently, and the same holds for the edges that go into the same vertex.

Lemma 1.5. ([PMA]) Nonsingularity is necessary for a uniquely decodable labelling.

Proof: If $L(s_0, s_1) = L(s_0, s_2) = L_0$, then we can construct arbitrarily long pairs of paths, ending in (s_0, s_1) and (s_0, s_2) respectively, which are label-indistinguishable but differ in the last edge.

Similarly, if $L(s_1, s_0) = L(s_2, s_0)$, we can construct arbitrarily long label-indistinguishable paths that differ in the first edge. ∎

We will henceforth consider only nonsingular labellings.

Rephrasing the Definition 1.3, we can say that a nonsingular labelling is UD iff there do not exist two infinitely long, label-indistinguishable, nonidentical paths.

Notice that an infinite path in a finite graph need not be periodic: in a complete graph with two states 0 and 1, an infinite aperiodic path is determined by a binary expansion of an irrational number. However, whenever a nonsingular labelling is not UD, there are two periodic label-indistinguishable paths. Or, since a periodic path is a sequence of repetitions of a circuit, we have

Theorem 1.6. Let N_E be the number of edges in a labelled graph, and let r be the maximal number of edges with the same label. A labelling is UD iff it is nonsingular and it does not contain two distinct label-indistinguishable circuits of length $\leq 1 + rN_E$.

(We shall see that this constant is not important, but simply serves to show that only the circuits up to a certain definite length need be checked.)

Proof:

(=>) We have shown in Lemma 1.1 that nonsingularity is necessary. If we do have two label-indistinguishable circuits, traversing them repeatedly constructs two infinitely long nonidentical label-indistinguishable paths.

(<=) Assume a nonsingular labelling is not UD. Then there exists an infinitely long label sequence $L_1L_2L_3\cdots$ that corresponds to different paths, i.e. to two sequences of edges. After $1 + rN_E$ edges, some edge would have been repeated in the first path $r + 1$ times. Let it have a label L_0. Then in the corresponding places in the second path, we would have edges with the same label L_0. Since there are $r + 1$ repetitions of L_0, and there are at most r distinct edges labelled by L_0, two of them must be the same. Mark these two places. The edge-sequences between them are two label-indistinguishable circuits. They are also distinct because of nonsingularity (in fact, no two corresponding edges in the first and the second path can be the same). ▮

So, having a nonsingular labelling, it is enough to check the circuits up to a certain length. The bound on length can be improved as the following lemma shows.

Lemma 1.7. A nonsingularly labelled graph with v vertices contains two label-indistinguishable circuits iff it contains two label-indistinguishable paths of length $\binom{v}{2}$.

Proof:

(=>) Trivial.

(<=) Given a labelled graph G, define another graph $\hat{G}$ as follows:

If $S = \{s_1, s_2, \cdots, s_v\}$ is the set of vertices of G, then $S^2 = \{(s_i, s_j) \mid s_i \in S, s_j \in S, s_i \neq s_j\}$ are the vertices of $\hat{G}$, and there is a branch labelled l from (s_{i_1}, s_{j_1}) to (s_{i_2}, s_{j_2}) in $\hat{G}$ iff $L(s_{i_1}, s_{i_2}) = L(s_{j_1}, s_{j_2}) = l$ in G. An example is given in Figure 1.9.

$\hat{G}$ has symmetry in the sense that whenever $L((s_{i_1}, s_{j_1}), (s_{i_2}, s_{j_2})) = l$, also $L((s_{j_1}, s_{i_1}), (s_{j_2}, s_{i_2})) = l$. It has $v(v-1)$ states, and every path in $\hat{G}$ corresponds to two distinct, label indistinguishable paths in G. Hence we need to show that $\hat{G}$ contains a circuit.

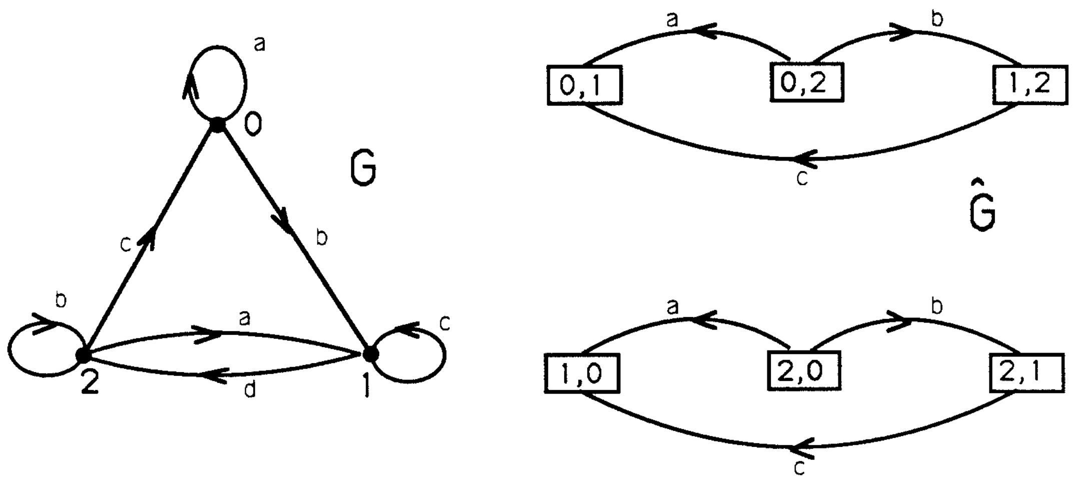


Figure 1.9.

We claim: whenever $\hat{G}$ contains a path of length $\binom{v}{2}$, it must contain a circuit. The path of this length must contain at least two vertices of the form $(v_i, v_j), (v_j, v_i)$ since there are only $\binom{v}{2}$ distinct subsets of two vertices from G (or else, it must contain some vertex twice, and consequently, a circuit). Let P_1 be the section of the path between these two vertices:

$$P_1 : (v_i, v_j) \rightarrow (x_1, y_1) \rightarrow (x_2, y_2) \rightarrow \cdots \rightarrow (x_k, y_k) \rightarrow (v_j, v_i).$$

Then $\hat{G}$ also contains the symmetric path

$$P_2 : (v_j, v_i) \rightarrow (y_1, x_1) \rightarrow (y_2, x_2) \rightarrow \cdots \rightarrow (y_k, x_k) \rightarrow (v_i, v_j).$$

But $P_1 P_2$ is a circuit and thus G contains two distinct label-indistinguishable circuits. ∎

As a corollary, we have:

Theorem 1.8. A labelling of a graph with v vertices is UD iff it is nonsingular and it does not contain two distinct label-indistinguishable paths of length $\geq \binom{v}{2}$.

Given this bound, an algorithm can be devised for checking whether a labelling is uniquely decodable. One can for instance use the depth-first search to compare all paths in the graph up to length $\binom{v}{2}$.

Corollary 1.9. The resolving length K_0 of a labelled graph with v vertices is bounded by

$$K_0 \leq \binom{v}{2}.$$

This bound is tight for $v = 4$ as Figure 1.10 shows. G is UD, yet the sequence $a\,b\,c\,d\,e$ of length $\binom{v}{2} - 1$ cannot be resolved before one more label is given.

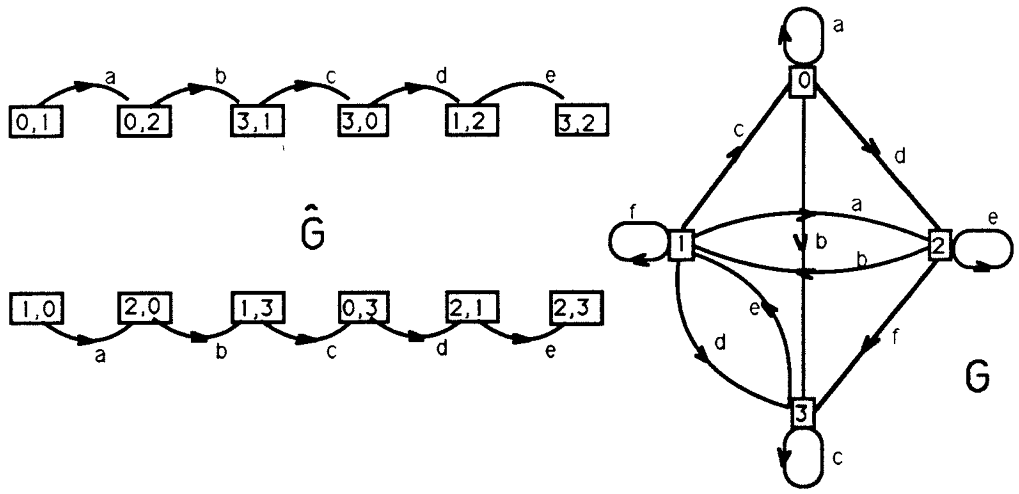


Figure 1.10.

We now turn to the question of how many labels are needed for a UD labelling of a graph. This is related to resolving length, as was apparent in the Example 1.2. The more labels are used, the shorter resolving length can be obtained. We shall consider a labelling to be efficient if it gives the shortest resolving length possible when using the minimal number of labels required to obtain a UD labelling on a given graph.

Definition 1.10. A graph is *D-regular* iff every vertex has exactly D outgoing edges and exactly D incoming edges.

In [PMA] it was shown that

- at least $2D$ labels are needed for a UD-labelling of a D-regular graph.

Although $2D$ labels are sometimes enough, that is not always the case: the 2-regular graphs in Figure 1.11 cannot be labelled using four labels.

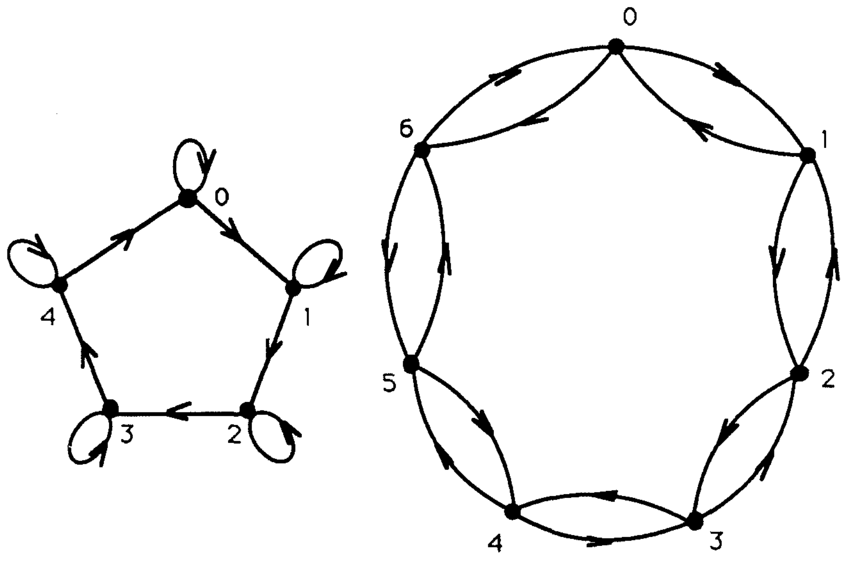


Figure 1.11.

Before introducing a class of D-regular graphs which *can* be labelled using $2D$ labels, we state a bound on resolving length of a v-vertex, D-regular graph.

Lemma 1.11. If a D-regular graph with v vertices is labelled using M labels, the resolving length is bounded below by

$$K_0 \geq \frac{\log v}{\log M - \log D}.$$

Proof : Each path of length K_0 can be uniquely described by its starting vertex and a sequence $\{x_1, x_2, \cdots, x_{K_0}\}$, $x_i \in \{0, 1, \cdots, D-1\}$, where x_i determines which of the D outgoing edges was taken at the step i. There are vq^{K_0} such paths. Also, there are M^{K_0} possible sequences of labels of length K_0. The inequality follows. ▮

If $M = 2D$, we get $K_0 \geq \log_2 v$. This bound will be achieved for a group of labellings introduced in the next section.

1.4. Convolutional Encoders as Finite State Machines

We have not given yet any examples of UD-labelled graphs which could be used to define the finite state machine. The general problem of finding an optimal labelling for any given graph is hard. In [PMA], a procedure is given for labelling complete graphs with q^m vertices using q^{m+1} labels.

It was suggested by Forney ([F3]) that convolutional encoders could be used to obtain UD labellings. We shall elaborate this idea, and analyze just what kinds of convolutional encoders are applicable in this context, and what are the parameters of labellings thus obtained. The first part of this section overlaps with work of Cheung ([Ch]).

We now review some well-known facts about convolutional codes. A reader unfamiliar with the subject can find a detailed introduction in [McE].

Let the input and output symbols belong to a finite field $F = GF(q)$. At the beggining of each clock cycle, the next k input symbols arrive in parallel to the input of the encoder, and n symbols appear at the output.

A (n, k, ν) *convolutional encoder* is a linear sequential circuit which consists of k shift registers of lengths $\nu_1 \geq \nu_2 \geq \cdots \geq \nu_k$, $\sum \nu_i = \nu$, and a memoryless arithmetic circuit which can perform additions and multiplications by constants in F. The ν symbols that are memorized in the shift registers and the k input symbols are led into the arithmetic circuit, where the n output symbols are computed. At the beginning of the next clock cycle, the contents of each shift register are shifted by one place to the right, the previous k input symbols are shifted into the leftmost cells of the shift registers, and next k symbols appear at the input. For an example, see Figure 1.12.

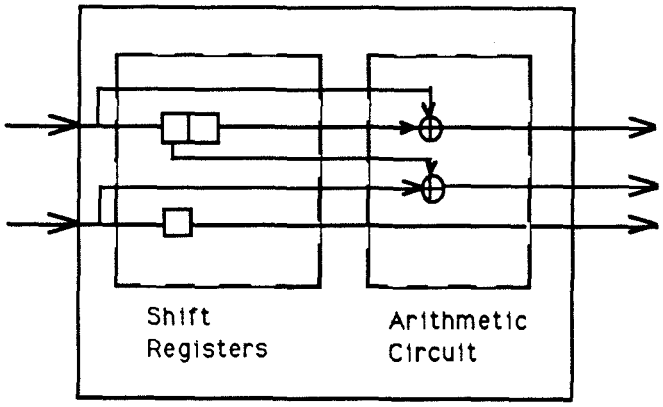


Figure 1.12. A (3,2,3) binary convolutional encoder with $\nu_1 = 2$, $\nu_2 = 1$.

Suppose that input k-tuples arrive at integer time intervals, starting at some (possibly negative, but finite) time d, and continuing to infinity. If the input at the time m is represented by the vector $\mathbf{x}_m = [x_m^{(1)}\, x_m^{(2)} \cdots x_m^{(k)}]^T$, then the whole input sequence can be represented as a transform in the delay operator D:

$$\mathbf{x} = \sum_{i=d}^{\infty} \mathbf{x}_i D^i .$$

$\mathbf{x}$ may also be viewed as a vector of k sequences in D over F. The set of all such sequences is the field $F(D)$ of Laurent series over F. The effect of the encoder on the input sequence—because of finite memory and linearity—can be represented by a $k \times n$-matrix $G(D)$ whose elements are polynomials in D over F (then ν_i is the largest among the degrees of polynomials in the i-th row of $G(D)$). The output coprresponding to $\mathbf{x}(D)$is the vector of n sequences from $F(D)$ given by $\mathbf{x}(D)G(D)$. The set of outputs when $\mathbf{x}$ ranges over all possible input sequences—in fact, the row space of $G(D)$ over $F(D)$—is the *code.* Since first of all the mapping must be 1–1, $G(D)$ must have rank k. For the encoder of Figure 1.12,

$$G(D) = \begin{pmatrix} 1+D^2 & D & 0 \\ 0 & 1 & D \end{pmatrix}.$$

We can also view the encoder as a finite state machine, and represent it by a state diagram labelled by outputs. Figure 1.13 shows the partial state diagram for the encoder of Figure 1.12—all states are there, but only the edges exiting from the all-zero and all-one state are shown. The two digits in brackets at the start of each edge are the input corresponding to that edge, and the 3-tuple in parentheses is the resulting output. Thus there are q^ν states, corresponding to the possible contents of the shift registers. From each state exit q^k edges, one for each input k-tuple. The edge leads to the state obtained by the shifting operation described above, and is labelled by the output n-tuple. We shall call the underlying graph a *Generalized de Bruijn graph*—it is analyzed in detail in the next chapter, Section 2.4.

Our next goal is to establish the conditions under which the encoder defined by a matrix $G(D)$ has a state diagram which is UD-labelled.

Let G_0 be the matrix obtained from G by picking the 0-th coefficient from each polynomial in $G(D)$. Similarly, G_h is obtained by choosing, in the i-th row, only the ν_i-th coefficient from each polynomial.

Lemma 1.12. The labelling is nonsingular iff both G_0 and G_h have rank k (over F).

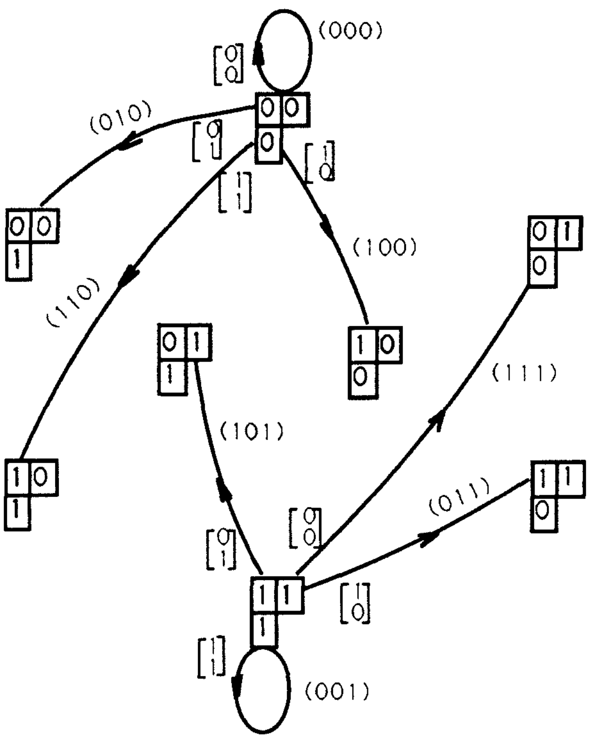


Figure 1.13.

Proof : We shall represent the current state by a k-tuple of polynomials in D^{-1}

$$\mathbf{s} = [s^{(1)}(D)\, s^{(2)}(D)\, \cdots\, s^{(k)}(D)],$$

$$s^{(i)}(D) = s_1^{(i)} D^{-1} + s_2^{(i)} D^{-2} + \cdots + s_{\nu_i}^{(i)} D^{-\nu_i}$$

where $s_j^{(i)}$ is the content of the j-th cell of the i-th shift register.

If $\mathbf{x}(D)$ is an n-tuple of polynomials, let $\mathbf{x}(D)\,|_0$ be the n-tuple of the coefficients of order zero. Let $\mathbf{s}_0$ be the column vector representing the current input k-tuple. The edges exiting from the state represented by $\mathbf{s}(D)$ are labelled by

$$\mathbf{l} = ((\mathbf{s}_0 + \mathbf{s}(D))G(D))\,\Big|_0$$

$$= (\mathbf{s}_0 G(D))\,\Big|_0 + (\mathbf{s}(D)G(D))\,\Big|_0$$

where $\mathbf{s}_0$ is the input vector corresponding to the edge. The second summand is common to all the edges that emanate from the same state. Since

$$(\mathbf{s}_0 G(D)) \Big|_0 = \mathbf{s}_0 G_0,$$

the edges will have distinct labels iff G_0 has full rank.

Similarly, the labels of edges entering a given state are

$$\mathbf{l} = \left((\mathbf{s}_h \mathrm{diag}(D^{-\nu_1}, D^{-\nu_2}, \cdots, D^{-\nu_k}) + D\mathbf{s}(D))G(D)\right) \Big|_0$$

where $\mathrm{diag}(x_1, x_2, \cdots, x_k)$ stands for a $k \times k$ diagonal matrix. Again, the second summand is common to all the edges entering the same state, so the labels will differ in

$$\left((\mathbf{s}_h \mathrm{diag}(D^{-\nu_1}, D^{-\nu_2}, \cdots, D^{-\nu_k})G(D)\right) \Big|_0 = \mathbf{s}_h G_h$$

(because the zero-coefficients after myltiplying $G(D)$ by the diagonal matrix will be exactly the high-order coefficients represented by G_h). Thus, the labels will be distinct iff G_h has full rank. ∎

One of the most important conditions which convolutional encoders must satisfy is *noncatastrophicness.* We gave its definition for trellis codes in Section 1.2, and observed that a trellis of a noncatastrophic code cannot contain two paths which differ in infinitely many edges, such that the Hamming distance between them is finite. There is a well-known criterion for catastrophicness of convolutional encoders, first observed by Massey and Sain in [MS]:

Definition 1.13. The k-th invariant factor of a $k \times n$ polynomial matrix $G(D)$ is the greatest common divisor of its $\binom{n}{k}$ $k \times k$-minors.

Remark: the elements of $G(D)$ belong to the principal ideal domain of polynomials over F, so with stipulation that gcd be monic, it is well-defined.

Theorem 1.14. ([MS]) A convolutionar encoder $G(D)$ is noncatastrophic iff its k-th invariant factor is D^m, for $m \geq 0$.

Clearly, if a labelling produced by a convolutional encoder is UD, the encoder has to be noncatastrophic (UD => sliding-block decodable => noncatastrophic). The following lemma shows that converse also holds.

Lemma 1.15. Suppose that the convolutional encoder is noncatastrophic and that it produces a nonsingular labelling. Then the labelling is also UD.

Proof : Suppose the labelling in not UD. Then, according to Theorem 1.5, there must be two nonidentical label-indistinguishable circuits, $s^1\, C^1\, s^1$, and $s^2\, C^2\, s^2$. In a convolutional encoder, we can reach any state from s_0 in exactly ν_1 edges. Let P^1 and P^2 be the paths of this length from s_0 to s^1 and s^2, respectively. Then the paths

$$s_0\, P_1\, s^1\, C^1\, s^1\, C^1\, s^1\, C^1\, \cdots$$

and

$$s_0\, P_2\, s^2\, C^2\, s^2\, C^2\, s^2\, C^2\, \cdots$$

differ in infinitely many edges, but have Hamming distance at most $n\nu_1$. ∎

So, we have proved

Theorem 1.16. The labelled state diagram corresponding to a (n, k, ν) convolutional encoder defined by the polynomial matrix $G(D)$ is UD iff

(i) G_0 has rank k,

(ii) G_h has rank k,

(iii) the k-th invariant factor of $G(D)$ is D^m for $m \geq 0$.

Actually, (i) implies that m in (iii) must be zero.

The encoders defined by the Theorem 1.16 have a name in the theory of convolutional codes—they constitute the family of *minimal* encoders. The term was introduced by Forney in [F1]. Since a convolutional code is defined as the row space of a polynomial matrix, there are obviously other matrices which have the same row space, and consequently other encoders which produce the given code. Forney proves that for any convolutional code there exists a minimal encoder which generates it. He further argues that minimal encoders should (almost) always be used because they are noncatastrophic, require the smallest possible number of memory elements for realization, and have other desirable properties which we shall not discuss here.

In [F2], Forney considers the question of the *longest zero-run* for a given encoder—the length of the longest path in the trellis whose labels are all zero, and which does not touch the zero state. He arrives at the following bounds (see [F2], Corollary 2):

Lemma 1.17. ([F2]). For minimal encoders, the longest zero-run l_0 is bounded by

$$\left\lceil \frac{\nu}{n-k} \right\rceil - 1 \leq l_0 \leq \nu - 1.$$

This will enable us to bound the resolving length of the state diagram of a convolutional encoder.

Lemma 1.18. The resolving length K_0 of the state diagram of a convolutional encoder is by one longer than the longest zero-run.

Proof : This follows from linearity of convolutional codes. We define the difference between two states, $s^1 - s^2$, as the state whose shift register cells contain the difference between contents of the corresponding cells of s^1 and s^2. The difference between two paths of same length is the path whose states are the difference between corresponding states in the first and the second path (we assume there are no multiple edges—$\nu_k \geq 1$—so the states are enough to specify a path). Since the mapping which produces the labels is linear, the labels of the new path will be the difference (componentwise) of the labels on the original paths.

Suppose there are two distinct label-indistinguishable paths, P_1 and P_2, of length $l_0 + 1$. Because of nonsingularity they cannot touch, so the path $P_1 - P_2$, which has all zero labels, does not touch the zero state. But such a path cannot exist if the longest zero-run is l_0, so $K_0 \leq l_0 + 1$.

On the other hand, if we take any path P of length l_0 and add to it the path P_0 which exhibits the longest zero-run (addition is defined similarly as subtraction), P and $P + P_0$ are two label-indistinguishable paths of length l_0, so $K_0 > l_0$. ∎

Corollary 1.19. The resolving length of the state diagram of a minimal (k, n, ν) convolutional encoder is bounded by

$$\left\lceil \frac{\nu}{n-k} \right\rceil \leq K_0 \leq \nu.$$

Let us summarize the results of this section.

A (k, n, ν) minimal encoder over $GF(q)$ produces a q^k-regular, q^ν-vertex UD-labelled state diagram where

- at most q^n labels are used,
- the resolving length K_0 satisfies

  $\left\lceil \frac{\nu}{n-k} \right\rceil \leq K_0 \leq \nu.$

  If $n = k + 1$ this K_0 is optimal.

Notice that for $n = k + 1$ the two bounds coincide: $K_0 = \nu$. Let M be the number of labels which are actually used. According to the bound on resolving length stated in Lemma 1.10

$$K_0 \geq \frac{\log q^{\nu}}{logM - \log q^k} \geq \frac{\nu}{(k+1)-k} = \nu,$$

so the resolving length is minimal.

If in addition the encoder is binary, the number of labels which are used is $2q^k$, which is also minimal (as is shown in [PMA]).

For given parameters k, ν and a partition of ν into $\nu_1 + \nu_1 + \cdots \nu_k$, one can specify the generic encoder as follows:

$$G(D) = \begin{pmatrix} 1 & D^{\nu_1} & 0 & \ldots & 0 & 0 \\ 0 & 1 & D^{\nu_2} & \ldots & 0 & 0 \\ 0 & 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & D^{\nu}_k \end{pmatrix} \qquad (*)$$

It is easy to see that such encoder is minimal. So, by construction:

For given K, $\nu \geq k$, and any $\nu_1 \geq \nu_2 \geq \cdots \geq \nu_k > 0$, $\sum \nu_i = \nu$, $(*)$ defines a binary convolutional encoder which produces a labelled state diagram which is 2^k-regular and has 2^{ν} states.
The labelling is uniquely decodable and optimal with respect both to the number of labels and the resolving length.

Before going on to construct some codes with the techniques developed here, let us refine the bound on d_{free}.

In Section 1.2 it was stated that the free distance of a finite state code is bounded by:

$$\min(2d_1, d_2) \leq d_{\text{free}} \leq d_2. \tag{1.1}$$

This can be improved if we have the information about the *free distance of the labelling*, D_{block}. D_{block} is the minimal number of corresponding labels in which two distinct semi-infinite paths in the trellis of the state diagram differ.

Let us return to the proof of (1.1) in Section 1.2. In the case where $e_1^{(1)}$ and $e_2^{(2)}$ lead to different states, by the time when the two paths remerge, at least D_{block} corresponding labels would have been distinct. Hence the accumulated distance in the overall trellis will be at least $D_{\text{block}}d_1$. Thus, the lower bound in (1.1) can be improved to

$$\min(D_{\text{block}}d_1, d_2) \leq d_{\text{free}}.$$

In [P], P. Piret introduced a class of convolutional codes, called *multiple-word correcting convolutional codes*, which are constructed with the aim of maximizing D_{block}. He defines the *word weight* of a path as the number of nonzero labels on it (a nonzero label being everything but $(0\,0 \cdots 0)$). A path in the state diagram which starts and ends in the zero state, does not have repeated edges, and does not contain the loop at the zero state is sometimes called a *fundamental path.* Because of linearity of convolutional codes, D_{block} equals the smallest word weight of a fundamental path. Clearly this is at most equal to the length of the shortest fundamental path, which is $\nu_k + 1$. Thus

$$2 \leq D_{\text{block}} \leq \nu_k + 1.$$

Piret also shows how to construct two families of convolutional encoders which are noncatastrophic and meet the above upper bound. We shall not repeat his results here, but they will be used in constructing an example in the next section.

0.1. Code Design

We now propose a method for constructing finite state codes and discuss the issues involved. We refer the reader to Section 1.2 for meaning of the parameters. The size of the input/output alphabet is q. If an encoder accepts k and outputs n symbols at the time, we say that its *rate* is k/n.

In code design, one usually starts with some requirements with respect to d_{free} and rate, and looks for the encoder of least complexity which will satisfy them. In case of finite state codes, the design can be divided into two steps:

1) Finding a set of disjoint block codes with minimal distance d_2 close to d_{free}. If we want $d_1 > 1$, then their union must also be some code C_1 with prescribed minimal distance. This determines d_1, d_2, the number of labels M, the size of the output n, and k_1.

2) From the rate, k_1 and n we get k_2; D_{block} should satisfy $D_{\text{block}} d_1 \simeq d_{\text{free}}$. We have to find a q^{k_2}-regular state diagram with the least number of states, labelled UD with $\leq M$ labels and with prescribed D_{block}.

Of course, other approaches are possible: one can start with a labelled state diagram and then look for block codes to match it.

One way to achieve 1) is to pick a linear code for C_1, its linear subcode for $C_2^{(0)}$ and for $C_2^{(i)}$ the i-th coset of C_2 in C_1. Then, if a convolutional encoder is used as a finite state machine, the overall encoder is the circuit shown in Figure 1.12. C_2 is a rate k_1/n block code and there are $\geq q^{n_1}$ cosets of C_2 in C_1. The n_1 symbols at the output of the convolutional encoder select an element of C_1 which is a coset representative for C_2 in C_1. This n-tuple is added to the n-tuple produced by the encoder for C_2 to get the final output.

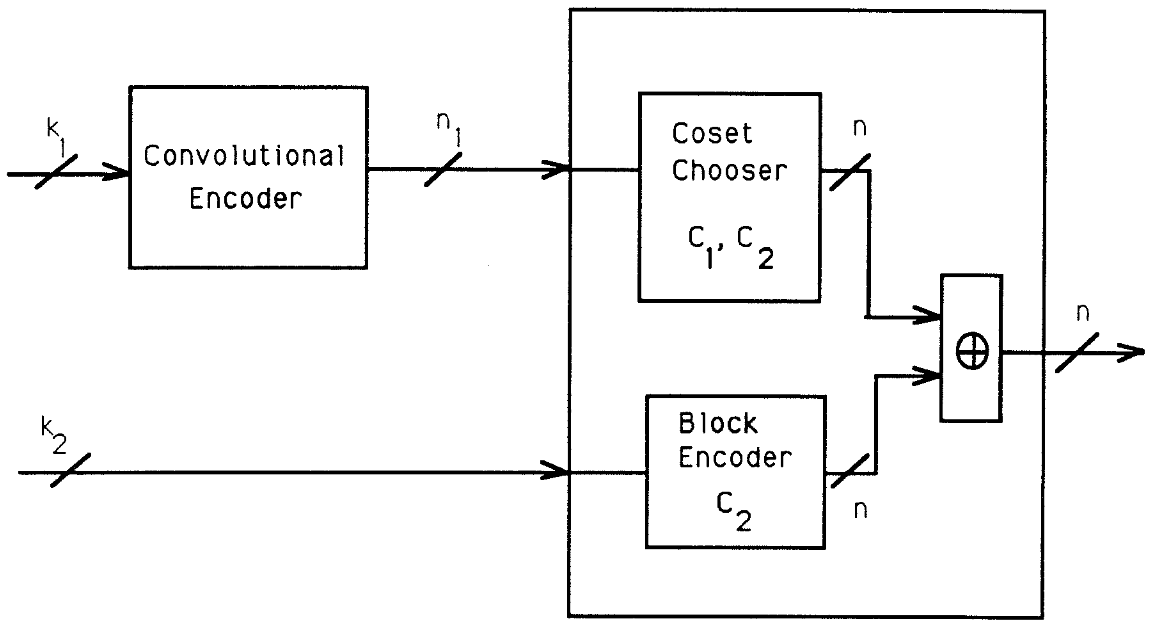


Figure 0.1.

Good candidates for such constructions are Reed-Muller codes (see [McWS], Chapter 13). For any $0 \leq r \leq m$, there exists a (binary) r-th order Reed Muller code $\mathcal{R}(r, m)$—an (n, k) linear block code such that

$$n = 2^m, \qquad k = \binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{r},$$

whose minimal distance is

$$d_{\min} = 2^{m-r}.$$

Also, $\mathcal{R}(r, m)$ is a subcode of $\mathcal{R}(r+1, m)$. So we get a sequence of codes

$$\mathcal{R}(0, m) \subset \mathcal{R}(1, m) \subset \cdots \subset \mathcal{R}(m, m)$$

where the minimal distance of each code is half that of the previous one.

Example 0.1. We shall work with the following Reed-Muller codes:

	(n,k)	$d_{\min}$
$\mathcal{R}(1,4)$	$(16,5)$	8
$\mathcal{R}(2,4)$	$(16,11)$	4
$\mathcal{R}(3,4)$	$(16,15)$	2

The goal is to have $d_{\text{free}} = 8$, $n = 16$, and the rate as large as possible. So $C_2^{(0)}$ will be $\mathcal{R}(1,4)$ and we want $D_{\text{block}} d_1 = 8$. Here are two ways to achieve that:

a) $D_{\text{block}} = 2$, $d_1 = 4$. Let $C_1 = \mathcal{R}(2,4)$. There are $M = 2^{k_1 - k_2} = 2^6$ cosets. We can use the generic encoder of $(*)$ with $k = 5$ and $\nu_1 = \nu_2 = \cdots = \nu_5 = 1$. The result is a $(16, 10, 5)$ finite state code with free distance 8.

b) $D_{\text{block}} = 4$, $d_1 = 2$. We now use $\mathcal{R}(3,4)$ for C_1, and get $M = 2^{10}$ cosets.

For the convolutional encoder, we shall use the construction for *free encoders* due to Piret ([P], Section III). Setting $a = 0$, $k = 7$, $m = 3$ we get the encoder whose $G(D)$ is

$$\begin{pmatrix}
1+D & D^2 & 0 & D^3 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1+D & D^2 & 0 & D^3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1+D & D^2 & 0 & D^3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1+D & D^2 & 0 & D^3 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1+D & D^2 & 0 & D^3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1+D & D^2 & 0 & D^3 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1+D & D^2 & 0 & D^3
\end{pmatrix}$$

This encoder is minimal, with rate $7/10$, memory $\nu = 21$ (the smallest needed to get $\nu_k = 3$), and $D_{\text{block}} = 4$. The resulting finite state code has parameters

$(16, 12, 21)$ and $d_{\text{free}} = 8$. So the rate is larger then in a), but the encoder is more complex.

References

[Ch] K.-M. Cheung: "Error-Correction Coding in Data Storage Systems," Ph.D. Thesis, California Institute of Technology, 1987.

[F1] G. D. Forney, "Convolutional Codes I: Algebraic Structure," *IEEE Trans. Inform. Theory* vol IT-16, 1970, pp. 720-738.

[F2] G. D. Forney, "Strucural Analysis of Convolutional Codes via Dual Codes," *IEEE Trans. Inform. Theory* vol IT-19, 1973, pp. 512-518.

[F3] G. D. Forney, "Coset Codes–Part I: Introduction and Geometrical Classification," *IEEE Trans. Inform. Theory*, vol.34, pp. 1123–1151, 1988.

[McE] R. J. McEliece, *The Theory of Information and Coding*, (Vol. 3 of the *Encyclopedia of Mathematics and its Applications*). Reading, MA: Addison-Wesley, 1977.

[McWS] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam: North-Holland, 1977.

[MS] J. L. Massey and M. K. Sain, " Inverses of Linear Sequential Circuits," *IEEE Trans. Computers*, vol. C-17, pp. 330–337, April 1968.

[P] P. Piret, "Multiple-Word Correcting Convolutional Codes," *IEEE Trans. Inform. Theory*, vol. IT-30, pp. 637–644, July 1984.

[PMA] F. Pollara, R. J. McEliece and K. Abdel-Ghaffar, "Finite-State Codes," *IEEE Trans. Inform. Theory*, vol. 34, pp. 1083-1089, Sept. 1988.

[Un] G. Ungerboeck, "Channel Coding with Multilevel/Phase Signals," *IEEE Trans. Inform. Theory* , vol. IT-28, Jan. 1982, pp. 55-67.

Chapter 2

Generalized de Bruijn Sequences

2.1. Introduction

A *de Bruijn sequence* with parameters (q, k) is a circular array of symbols from the alphabet $A = \{0, 1, ..., q-1\}$ such that every k-tuple from A^k appears once and only once as k consecutive figures on the circle. Let $B(q, k)$ be the set of all sequences with this property (two sequences which differ only by a rotation are considered to be equal). Throughout this chapter we shall refer to a member of $B(q, k)$ as 'a $B(q, k)$ sequence'. A $B(2, 3)$ sequence appears in Figure 2.1.

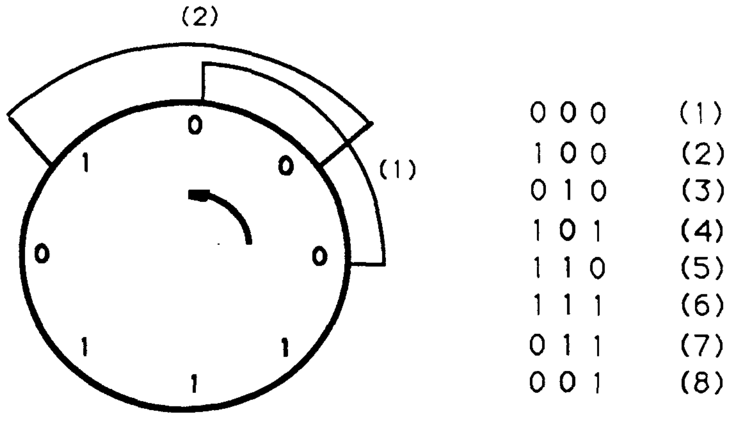


Figure 2.1.

In order to construct such a sequence, consider the so-called *de Bruijn graph* $G_B(q, k-1)$. Its vertices are the q^{k-1} $(k-1)$-tuples over A, and there is an edge from state a to state b iff the first $(k-2)$ coordinates of a agree with the last $(k-2)$ coordinates of b. There is a natural 1–1 correspondence between Euler circuits in $G_B(q, k-1)$ and the sequences $B(q, k)$: for example, the circuit in Figure 2.2 corresponds to the sequence in Figure 2.1.

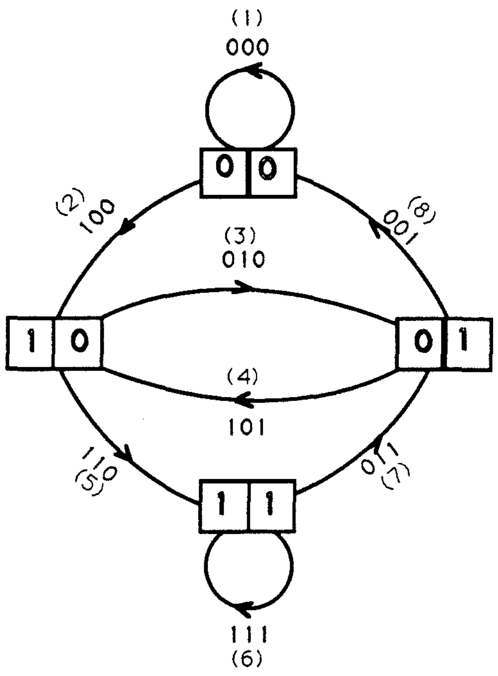


Figure 2.2. $G_B(2,2)$.

Thus, the problem is translated into one concerning graphs, where it becomes possible to show the existence of de Bruijn sequences and to compute their number using techniques for counting the Euler circuits in a graph. The number is

$$|B(q,k)| = q^{-k}(q!)^{q^{k-1}}.$$

This result was apparently known as far back as 1894 ([FSM]), but it was forgotten and rediscovered in [EB], as a corollary of more general theorems stated in that paper.

In engineering literature, de Bruijn sequences are often called *shift register sequences*, since shift registers are the usual hardware used for generating them. Apart from their intrinsic combinatorial interest, they have found applications in communications and computer science as pseudo–random sequences, in period generators, etc. A detailed review of properties and applications of shift register sequences can be found in [Gol]. The search for good algorithms which would generate those sequences is still on (see [Fr]).

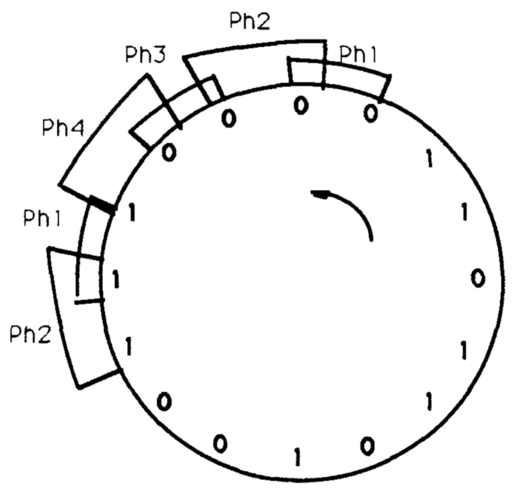
In the following sections, in an attempt to generalize the concept of a de Bruijn sequence, we propose two new families of *Generalized de Bruijn* (GDB) sequences : *Type 1* and *Type 2.*

Type 1 GDB sequences are obtained by introducing 'phases' on the circle, which correspond to positions modulo m on it. We require that each k-tuple appears exactly once in each phase. In Figure 2.3 we have an example over alphabet $A = \{0,1\}$, with $k = 2$ and 4 phases.

Type 2 GDB sequences will be formally defined in Section 2.3; for now, the example in Figure 2.4 will suffice. There are two concentric circles, same number of positions on each. We observe an irregularly shaped "window" and require that each of the eight possible window contents be displayed exactly once as it slides across the circles.

By relating the above sequences to Euler circuits in certain graphs, we will, using the results from [1], prove existence and derive formulae for the number of sequences of Type 1 and 2.

In Section 2.5, the Generalized de Bruijn sequences are finally obtained. They include the sequences of Type 1 and 2. We give the formula for their number, which contains the previously derived formulae as special cases.



Phase 1	Ph 2	Ph 3	Ph 4
0 0	0 0	0 0	1 0
1 1	1 1	0 1	0 0
1 0	0 1	1 0	1 1
0 1	1 0	1 1	0 1

Figure 2.3.

2.2. Euler Circuits in Directed Graphs

In this section, we introduce the notions and results of graph theory which will be used in the sequel.

Definition 2.1. A *graph* $G = G(V, E, \tau, \chi)$ is determined by two finite sets, V and E, and two mappings, τ and $\chi : E \to V$.

The elements of V are called *vertices*, the elements of E *edges*, and in the usual pictorial representation, an edge is an oriented arc from vertex $\tau(e)$ (tail of e) to $\chi(e)$ (head of e). The definition thus allows more than one edge from vertex v_i to v_j, and also edges whose head and tail is the same vertex. Such graphs are sometimes

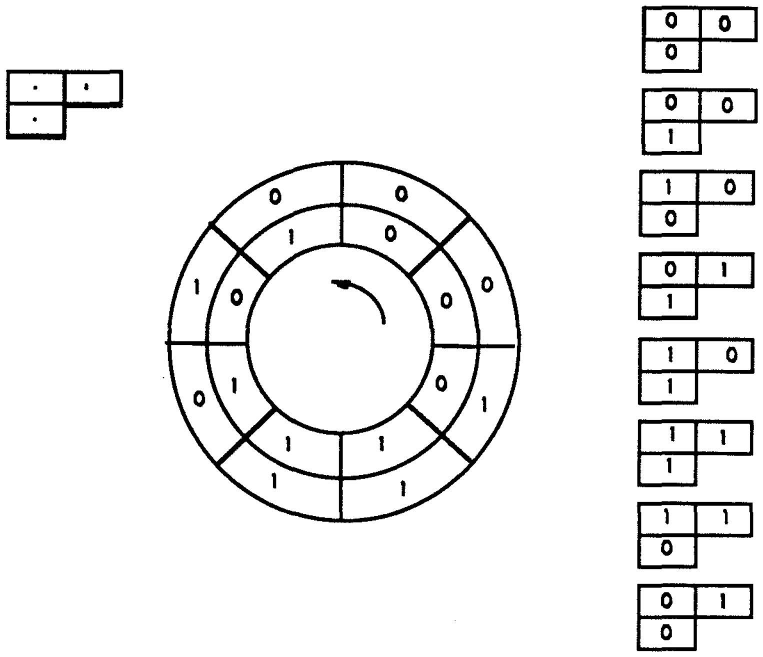


Figure 2.4.

called multigraphs with loops.

Definition 2.2. Two graphs, $G_1(V, E, \tau_1, \chi_1)$ and $G_2(W, F, \tau_2, \chi_2)$, are *isomorphic* iff there are bijective maps

$$\phi : V \to W$$

$$\psi : E \to F$$

such that

$$\tau_2(\psi(e)) = \phi(\tau_1(e)),$$

and similarly for χ. We denote this relation by $G_1 \cong G_2$.

Two edges e_1, e_2 are *consecutive* if $\chi(e_1) = \tau(e_2)$. A sequence of consecutive edges is called a *path*; a closed path is a *circuit.* A graph is *strongly connected* iff for any two vertices v_1, v_2 there is a path from v_1 to v_2. We shall assume that all graphs in this chapter are strongly connected.

We shall be interested in circuits with a special property:

Definition 2.3. An *Euler circuit* in G is a circuit which contains every edge of G exactly once.

For a given vertex v, its *out-degree* $\sigma_o(v)$ is the number of edges with tail at v, and *in-degree* $\sigma_i(v)$ is the number of edges with head at v. A graph in which for each vertex v, $\sigma_o(v) = \sigma_i(v)$, is *pseudosymmetric*. A graph such that $\sigma(v) = \sigma$ for all v is *σ-regular*. There is a well-known, easy criterion for determining whether a graph posesses an Euler circuit (see, e.g., [Ber]).

Theorem 2.4. A graph has an Euler circuit iff it is pseudosymmetric.

We now define three operations on graphs: *doubling*, *higher edge* and *product*. The first two operations are unary, while the third acts on two graphs.

Definition 2.5. The *doubling* operation:

For a positive integer λ, let G^λ be the graph obtained by replacing each edge in G by λ edges between the same vertices, with the same orientation. In terms of Definition 2.1, if G is given by (V, E, τ, χ), then the set of vertices of G' is V, for each edge $e \in E$ there are λ edges $e_1, e_2, \cdots, e_\lambda$ in E', and the two mappings are defined by:

$$\tau'(e_i) = \tau(e), \qquad \chi'(e_i) = \chi(e).$$

Notice that $(G^{\lambda_1})^{\lambda_2} = G^{\lambda_1 \lambda_2}$.

Definition 2.6. The *higher edge* operation:

For a given graph G, let G' be the *higher edge graph*, obtained from G as follows: the edges of G are the vertices of G', and there is an edge from vertex i

to vertex j in G' iff edge j follows edge i in G. So if $G = G(V, E, \tau, \chi)$, we define $G' = G'(V', E', \tau', \chi')$ as follows:

$$V' = E,$$

$$E' = \{(e_1, e_2) \mid e_1 \in E, e_2 \in E, \chi(e_1) = \tau(e_2)\},$$

$$\tau(e_1, e_2) = e_1, \quad \chi(e_1, e_2) = e_2.$$

Definition 2.7. A *product* of graphs $G_1(V, E, \tau_1, \chi_1)$ and $G_2(W, F, \tau_2, \chi_2)$, denoted $G1 \times G2$, is the graph $G(U, H, \tau, \chi)$, where

$$U = V \times W,$$

$$H = E \times F,$$

$$\tau(e, f) = (\tau_1(e), \tau_2(f)),$$

$$\chi(e, f) = (\chi_1(e), \chi_2(f)).$$

The product operation is associative, so the above definition extends inductively to a product of finitely many graphs. Also the product is commutative up to a graph isomorphism.

We now give several well-known theorems about Euler circuits in graphs, which were originally stated in [EB]. Let $\mid \xi(G) \mid$ denote the number of Euler circuits in G. Two circuits are considered the same if one can be obtained from the other by a cyclic shift of edges. We formulate the theorems for the special case of regular graphs, since other graphs will not appear in the sequel.

Theorem 2.8. ([EB]) Let G be a σ-regular graph with N vertices, and G' the higher edge graph. Then

$$\mid \xi(G') \mid = \sigma^{-1}(\sigma!)^{N(\sigma-1)} \mid \xi(G) \mid .$$

Theorem 2.9. ([EB]) Let G be a σ-regular graph with N vertices and G^λ as in Definition 2.5. Then

$$|\xi(G^\lambda)| = \lambda^{-1}\left(\frac{(\lambda\sigma)!}{\sigma!}\right)^N |\xi(G)|.$$

2.3. Sequences of Type 1

We start with a formal definition of these sequences.

Definition 2.10. Let q, k, n be positive integers. A *Generalized de Bruijn sequence of Type 1* (a $B1(q,k,n)$ sequence) is a circular array of letters from the set $A = \{0, 1, ..., q-1\}$ with the following property:

Enumerate positions on the circle $1, 2, 3, \cdots$, and consider a k-tuple of consecutive symbols to belong to phase i, $i = 0, 1, ..., n-1$, if the position of the first symbol is equal to i modulo n. We require that in each phase, every k-tuple of A^k appear exactly once.

Again, the sequences can be corresponded to Euler circuits in certain graphs.

Definition 2.11. A *Cycle* graph of order n, C_n, is the graph with vertices $V = \{0, 1, ..., n-1\}$, edges $E = \{(0,1), (1,2), ..., (n-1, 0)\}$, and the two mappings defined by: $\chi(i, i+1) = i+1$, $\tau(i, i+1) = i$ (addition is mod n). (See Figure 2.5.)

Definition 2.12. For positive integers q and k, *de Bruijn* graph with parameters q and k, $G_B(q,k)$, is the graph with vertices $V = A^k$ (where $A = \{0, 1, 2, ..., q-1\}$) and edges $E = A^{k+1}$. The mappings χ, τ are

$$\chi(x_1, x_2, ..., x_{k+1}) = (x_1, x_2, ..., x_k),$$

$$\tau(x_1, x_2, ..., x_{k+1}) = (x_2, x_3, ..., x_{k+1}).$$

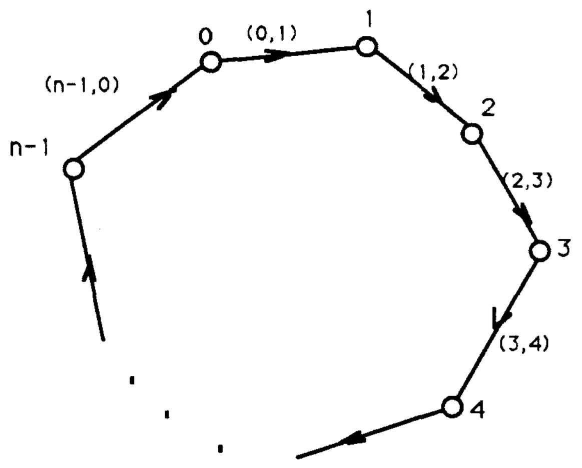


Figure 2.5.

We also define $G_B(q,0)$ to be the graph with one vertex and q edges-loops.

For $q = 2$, the first few graphs of this kind are given in Figure 2.6. A $G_B(q,k)$ has q^k vertices, and is q-regular.

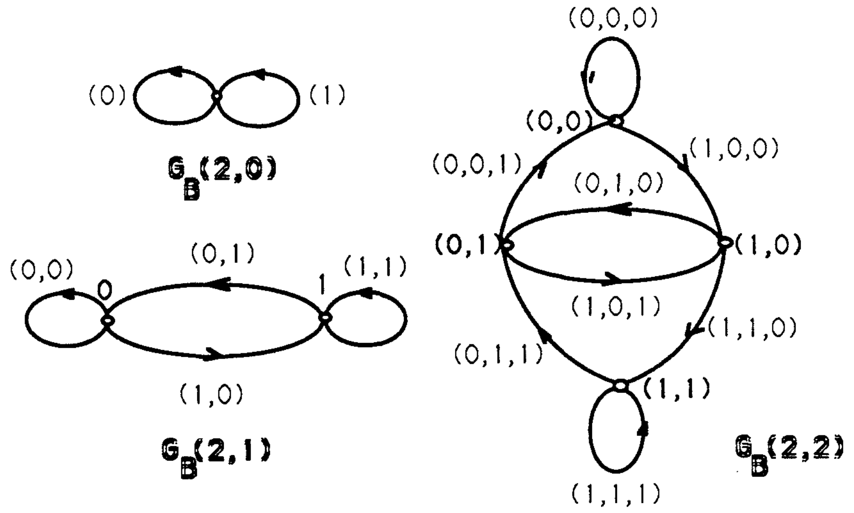


Figure 2.6.

Lemma 2.13. For given q and k, there is a 1–1 correspondence between sequences

$B1(q,k,n)$ and Euler circuits in $C_n \times G_B(q,k-1)$.

Proof: Consider the product $C_n \times G_B(q,k)$. The new graph is regular, and so has an Euler circuit. According to the notation introduced in Definitions 2.10, 2.11 and 2.12, and assuming w.l.o.g. that the first edge is fixed, any Euler circuit is a sequence of edges $s_1, s_2, \ldots s_{|E|}$ where

$$s_i = (e^i \in E_{C_n}, f^i \in E_{G_B}) = ((i_{\text{mod}n}, (i+1)_{\text{mod}n}), (x_1, x_2, \ldots x_{k+1})),$$

that is, the first indices e^i go cyclically modulo n. Looking at the second indices f^i, we get a succesion of k-overlapping $k+1$-tuples; moreover, for given i, $0 \le i \le n-1$, if we fix the first coordinate of the first index to be i, every possible $k+1$-tuple will appear exactly once as the second index. But this looks like a sequence of type two, and indeed we get such a sequence if we take as its i-th member the last coordinate of the second index of s_i. This mapping is easily seen to be invertible and onto, and we have thus established the required 1–1 correspondence. ∎

In order to determine the number of Euler circuits in a direct product of Cycle and de Bruijn graph, we first establish some preliminary results.

Lemma 2.14. $G_B(q,k+1)$ is the higher edge graph of $G_B(q,k)$, i.e.,

$$G_B(q,k+1) \cong (G_B(q,k))'.$$

The proof is straightforward, and we omit it.

Lemma 2.15. The higher edge graph of the product of two graphs is the product of their higher edge graphs:

$$(G_1 \times G_2)' \cong (G_1') \times (G_2').$$

Proof : Let $G_1 = G_1(V, E, \tau_1, \chi_1)$, $G_2 = G_2(W, F, \tau_2, \chi_2)$.

Consider the left-hand side first. It is a graph whose vertex set is $E \times F$, and whose edge set is the subset of $((E \times F) \times (E \times F))$ given by

$$\{((e_1, f_1), (e_2, f_2)) \mid \chi_{G_1}(e_1) = \tau_{G_1}(e_2), \chi_{G_2}(f_1) = \tau_{G_2}(f_2)\}.$$

The mappings are

$$\chi((e_1, f_1), (e_2, f_2)) = (e_2, f_2),$$

$$\tau((e_1, f_1), (e_2, f_2)) = (e_1, f_1).$$

The right-hand side graph has for the vertex set also $E \times F$, and for the edge set the direct product of the subset of $E \times E$ of the form

$$\{(e_1, e_2) \mid \chi(e_1) = \tau(e_2)\}$$

with the similar subset of $F \times F$. The two functions are

$$\chi((e_1, e_2), (f_1, f_2)) = (e_2, f_2),$$

$$\tau((e_1, e_2)), f_1, f_2)) = (e_1, f_1).$$

Now it is easy to see that the mappings

$$(e, f) \rightarrow (e, f)$$

$$((e_1, f_1), (e_2, f_2)) \rightarrow ((e_1, e_2), (f_1, f_2))$$

establish the required isomorphism. ∎

Corollary 2.16.

$$(C_n \times G_B(q, k))' \cong C_n' \times (G_B(q, k))' \cong C_n \times G_B(q, k + 1).$$

Proof : After observing that $C_n' = C_n$, the proof is immediate from Lemmas 2.14 and 2.15. ∎

We now proceed to determine the number of Euler circuits in $C_n \times G_B(q,k)$.

Lemma 2.17.

$$\mid \xi(C_n \times G_B(q,k)) \mid = q^{-k-1}(q!)^{nq^k}.$$

Proof : We use induction on k. For $k=0$, we have the graph $C_n \times G_B(q,0) = C_n^\lambda$ with $\lambda = q$. It is easy to see that the number of Euler circuits here is $q^{-1}(q!)$, which agrees with the formula.

Suppose the claim is true for k. By Corollary 2.16 and Theorem 2.8,

$$\mid \xi(C_n \times G_B(q,k+1)) \mid = \mid \xi((C_n \times G_B(q,k))') \mid =$$

$$= q^{-1}(q!)^{nq^k(q-1)} \mid \xi(C_n \times G_B(q,k)) \mid = q^{-(k+1)-1}(q!)^{nq^{k+1}}.$$

∎

We summarize the results of this paragraph in the following theorem:

Theorem 2.18. For a given triple of positive integers q, k, n, there are exactly

$$\mid B1(q,k,n) \mid = q^{-k}(q!)^{nq^{k-1}}$$

distinct sequences of Type 2. These sequences are in 1–1 correspondence with the Euler circuits in the graph $C_n \times G_B(q,k-1)$, so that given one, we can determine the other.

Example 2.19. A $B1(2,2,4)$ appeared in Figure 2.3. The corresponding Euler circuit can be found in the graph $C_4 \times G_B(2,1)$ (Figure 2.7). There are 64 different $B1(2,2,4)$ sequences.

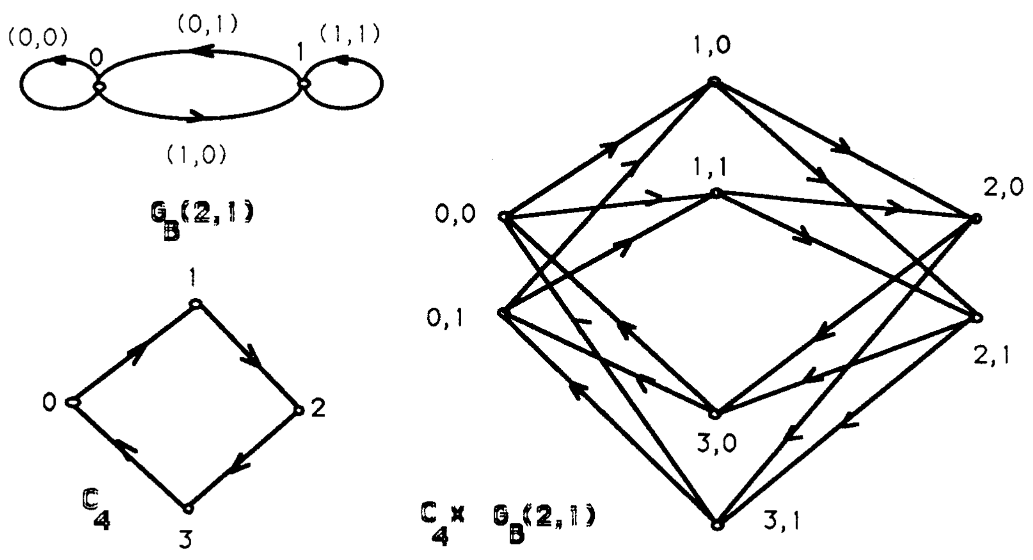


Figure 2.7.

2.4. Sequences of Type 2

Before introducing sequences of Type 2, we give the definition of partition.

Definition 2.20. A *partition* of a positive integer ν into m parts, $m \leq \nu$, is a set of integers $\nu_1 \geq \nu_2 \geq \ldots \geq \nu_m \geq 1$ such that $\nu_1 + \nu_2 + \ldots + \nu_m = \nu$.

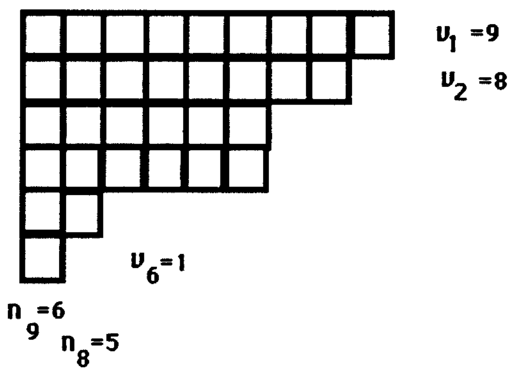


Figure 2.8. The Ferrers diagram of the partition of 29 into $9 + 8 + 6 + 6 + 2 + 1$.

A convenient way to represent a partition is by the so-called *Ferrers diagram* (Figure 2.8). The number of squares in the first row is ν_1, in the second ν_2, etc. The area (i.e., number of squares) of the whole picture is ν. We could equally well describe this diagram by counting the number of squares in columns instead of rows. For the sake of simpler notation in the sequel, we number the columns from right to left, so that we have the *conjugate* parameters $\eta_k \geq \eta_{k-1} \geq .. \geq \eta_1 \geq 1$. Naturally, $\sum \eta_i = \nu$, $\eta_k = m$, $k = \nu_1$. In the rest of this section and in the next, we shall always denote the original parameters of a Ferrers diagram by ν_i, and the conjugate parameters by η_i.

Suppose now that the squares of a given Ferrers diagram are not empty, but each contains a letter from an alphabet $A = \{0, 1, ..., q-1\}$. We call such a configuration a *Ferrers crossword* . There are q^ν Ferrers crosswords corresponding to a given Ferrers diagram of area ν.

Let an $m \times q^\nu$ matrix of letters from A be given. If we position a Ferrers diagram with m rows so that its leftmost column covers the i-th column of the matrix, the letters which are thus covered constitute the *Ferrers crossword at position i* of the matrix. Suppose that the matrix continues periodically to the right so that Ferrers crosswords are defined for all q^ν positions (Figure 2.9). If all Ferrers crosswords of such a matrix are distinct, we have a Generalized de Bruijn Sequence of Type 2 (with the given parameters). The name is motivated by the connection to de Bruijn sequences, although this is not a sequence but a matrix, or a collection of m sequences.

We are now ready for the definition.

Definition 2.21. An alphabet $A = \{0, 1, ..., q-1\}$, positive integers ν and m, $m \leq \nu$, and a partition of ν into m parts, $\nu_1 \geq \nu_2 \geq ... \geq \nu_m \geq 1$, are given. An

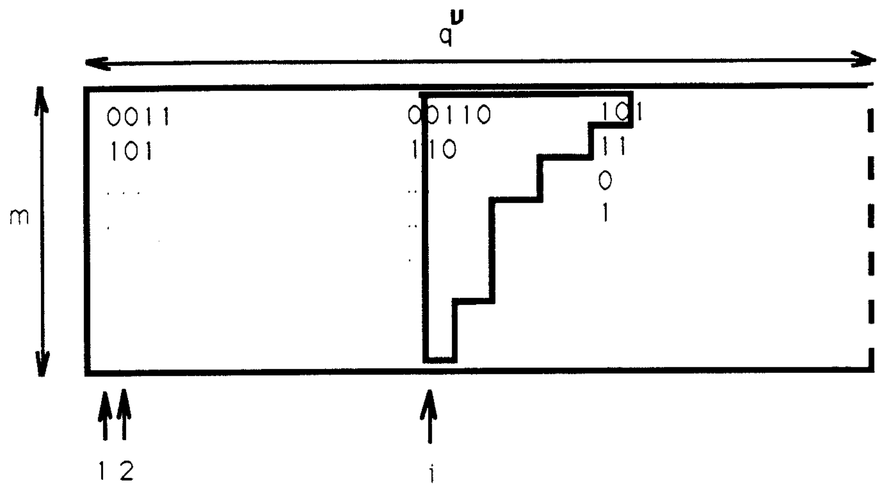


Figure 2.9.

$m \times q^\nu$ matrix of letters from A is a *Generalized de Bruijn Sequence of Type 2*,

$$GB_2(q, m, \nu; \nu_1, \nu_2, ...\nu_m),$$

iff the Ferrers crosswords (corresponding to the Ferrers diagram of the partition) at positions $1, 2, ..., q^\nu$ are all distinct.

Alternatively, we shall use the conjugate parameters and denote the sequence by $GB_2(q, k, \nu; \eta_k, \eta_{k-1}, ...\eta_1)$.

Corresponding to a sequence with a specified set of parameters, we introduce the Generalized de Bruijn graph.

Definition 2.22. Given positive integers ν and m, a partition of ν into m parts, and an alphabet $A = \{0, 1, ..., q-1\}$, we define a graph such that:

- the set E of edges is: all Ferrers crosswords over A corresponding to the partition;
- the set V of vertices is: all Ferrers crosswords over A corresponding to the above Ferrers diagram with the leftmost column removed;

- the τ function is: 'remove the last square from each row';
- the χ function is: 'remove the first square from each row'.

We name it *Generalized de Bruijn* (GDB) graph, and denote it by

$$G_{GDB}(q, m, \nu; \nu_1, \nu_2, ...\nu_m),$$

or via conjugate parameters, by

$$G_{GDB}(q, k, \nu; \eta_k, \eta_{k-1}, ..., \eta_1).$$

The GDB graph with $q = 2$, $m = 2$, $\nu = 3$; $\nu_1 = 2$, $\nu_2 = 1$, appears in Figure 2.10.

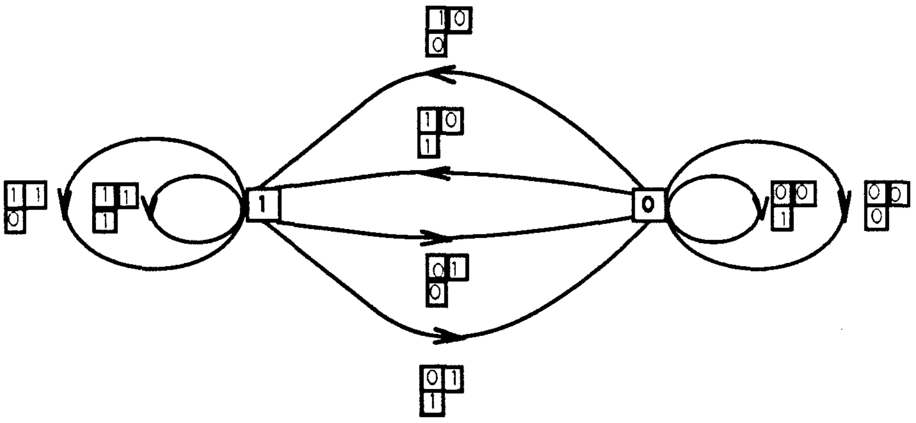


Figure 2.10.

Given this definition, the following is almost obvious:

Claim 2.23. There is a 1–1 correspondence between Euler circuits in a GDB graph and sequences of Type 2 with same parameters.

There is an equivalent, more elegant definition of GDB graphs, which we shall not, however, use in the sequel.

Definition 2.24. GDB graph is a product of ordinary de Bruijn graphs:

$$G_{GDB}(q,m,\nu;\nu_1,\nu_2,...,\nu_m) = G_B(q,\nu_1-1)\times G_B(q,\nu_2-1)\times ...\times G_B(q,\nu_m-1).$$

Notice that multiplying by $G_B(q,0)$ amounts to "doubling" by a factor $\lambda = q$. Thus in general a GDB graph will have either 0 or $q^{\eta_k-\eta_{k-1}}$ edges between any two vertices.

Lemma 2.25. Any GDB graph over a q-ary alphabet can be obtained from a suitable 1-vertex graph by a sequence of higher edge and doubling operations, as follows:

$$G_{GDB}(q,k,\nu;\eta_k,\eta_{k-1},...,\eta_1) \cong ((G_{GDB}(q,k-1,\nu-\eta_k;\eta_{k-1},\eta_{k-2},...,\eta_1)')^{\lambda},$$

where

$$\lambda = q^{\eta_k-\eta_{k-1}}.$$

The proof is obtained by bookkeeping according to the definitions of the two operations, and by verifying that the two graphs are indeed the same.

Lemma 2.26. The number of Euler circuits in a GDB graph is

$$\mid \xi(G_{GDB}(q,k,\nu;\eta_k,...,\eta_1)) \mid = q^{-\nu}((q^{\eta_k})!)^{q^{\nu-\eta_k}}.$$

Proof : Induction on k.

For $k=1$, we have a 1-vertex graph with q^{η_1} loops. The formula gives $((q^{\eta_1}-1)!)$, which is true.

Suppose the formula is true for $k-1$. We use the Theorem 2.9 together with Lemma 2.24 and obtain

$$\mid \xi(G_k) \mid = \lambda^{-1}(\frac{(\lambda\sigma)!}{\sigma!})^N \mid \xi((G_{k-1})') \mid,$$

where

$$\lambda = q^{\eta_k - \eta_{k-1}},$$

$$\sigma = q^{\eta_{k-1}},$$

$$N = q^{\nu - \eta_k}.$$

By Theorem 2.8,

$$\mid \xi((G_{k-1})') \mid = {\sigma_1}^{-1}(\sigma_1!)^{N_1(\sigma_1 - 1)} \mid \xi(G_{k-1}) \mid,$$

where

$$\sigma_1 = q^{\eta_{k-1}},$$

$$N_1 = q^{\nu - \eta_k - \eta_{k-1}}.$$

Substituting the formula for $k-1$ according to the induction hypothesis, the correct formula for k is obtained. ▮

We end this paragraph with the theorem which summarizes the main result.

Theorem 2.27. There are exactly

$$q^{-\nu}((q^m)!)^{q^{\nu - m}}$$

distinct sequences $GB_2(q, m, \nu; \nu_1, \nu_2, ..., \nu_m)$; they correspond to all Euler circuits in the GDB graph with the same parameters.

Example 2.28. The $B(2,2,3;2,1)$ sequence that appeared in Figure 2.4 can be easily traced using the edge labels in Figure 2.10. According to the formula, there are five more sequences to be found in the graph.

2.5. Generalized de Bruijn Sequences

As was said in the Introduction, these sequences combine the generalizations of de

Bruijn sequences which yielded the Type 1 and Type 2 sequences. The methods used in the previous two paragraphs will readily extend to this case.

We start with a sequence like one of Type 2, but now we introduce n phases. A Ferrers crossword at position i belongs to phase l iff $(i \equiv l) mod_n$.

Definition 2.29. Given a q-ary alphabet A, positive integers n, ν, m, and a partition of ν: $(\nu_1, \nu_2, ..., \nu_m)$ we say that an $m \times nq^\nu$-matrix of letters from A is a Generalized de Bruijn sequence

$$GB(q, m, \nu, n; \nu_1, \nu_2, ..., \nu_m)$$

iff for each of the n phases, all q^ν Ferrers crosswords belonging to that phase are distinct.

After the discussion in the previous paragraphs it is easy to see that the following is true:

Lemma 2.30. The GDB sequences with n phases are in 1–1 correspondence with the Euler circuits in the product of Cycle graph of order n with the appropriate Generalized de Bruijn graph, and so

$$\mid GB(q, m, \nu, n; \nu_1, \nu_2, ..., \nu_m) \mid = \mid \xi(C_n \times G_{GDB}(q, m, \nu; \nu_1, ..., \nu_m)) \mid .$$

Before proceeding to the main lemma, we need another property of the product operation.

Lemma 2.31. Let $G_1 = G_1(V, E, \tau_1, \chi_1)$ and $G_2 = G_2(W, F, \tau_2, \chi_2)$ be two graphs and let G^λ denote the doubling operation. Then

$$(G_1 \times G_2)^\lambda \cong G_1 \times ((G_2)^\lambda).$$

Proof : Consider the left-hand side first.

The set of vertices is $V \times W$.

Edges are

$$(E \times F)^{(\lambda)} = \{(e,f)^{(i)} \mid e \in E, f \in F, i = 1,2,...,\lambda\}.$$

The two mappings are

$$\tau((e,f)^{(i)}) = (\tau_1(e), \tau_2(f)),$$

$$\chi((e,f)^{(i)}) = (\tau_1(e), \tau_2(f)).$$

On the right-hand side, the vertices are $V \times W$,

edges

$$(E \times F^{(\lambda)}) = \{(e, f^{(\subset)}) \mid e \in E, f \in F, i = 1,2,...,\lambda\},$$

the mappings

$$\tau((e,f)^{(i)}) = (\tau_1(e), \tau_2(f)),$$

$$\chi((e,f)^{(i)}) = (\tau_1(e), \tau_2(f)).$$

Then, the required graph isomorphism is established by

$$(e,f)^{(i)} \to (e, f^{(i)}).$$

∎

The next lemma gives the inductive step for building the larger graph from a smaller. We are using the conjugate parameters.

Lemma 2.32.

$$C_n \times G_{GDB}(q,k,\nu;\eta_k,...,\eta_1) = ((C_n \times G_{GDB}(q,k-1,\nu-\eta_k;\eta_{k-1},...,\eta_1))')^{\lambda},$$

where $\lambda = q^{\eta_k - \eta_{k-1}}$.

Proof :

$$((C_n \times G_{k-1})')^{\lambda} = (C_n' \times G_{k-1}')^{\lambda} = (C_n \times G_{k-1}')^{\lambda} = C_n \times (G_{k-1}')^{\lambda}) = C_n \times G_k$$

∎

Lemma 2.33. The number of Euler circuits in the product of Cycle and Generalized de Bruijn graph is

$$\mid \xi(C_n \times G_{GDB}(q, m, \nu; \nu_1, ..., \nu_m)) \mid = q^{-\nu}((q^m)!)^{nq^{\nu - m}}.$$

Proof : Using the conjugate parameters and induction on k we have, for $k = 1$, a doubled Cycle graph with n vertices and $\lambda = q^{\eta_1}$ edges between successive vertices. The number of Euler circuits in this graph is obviously

$$\lambda^{-1}(\lambda!)^n$$

which agrees with our formula.

For the induction step, if Lemma 2.30 is used, the proof is completely analogous to that of Lemma 2.25. ∎

The main theorem of this chapter follows.

Theorem 2.34. There are exactly

$$q^{-\nu}((q^m)!)^{nq^{\nu - m}}$$

distinct sequences $GB(q, m, \nu, n; \nu_1, \nu_2, ..., \nu_m)$, and they correspond 1–1 to the distinct Euler circuits in the graph

$$C_n \times G_{GDB}(q, m, \nu; \nu_1, \nu_2, ..., \nu_m).$$

For $m = 1$ we get the sequences of Type 1, for $n = 1$ those of Type 2. Finally, if both $m = 1$ and $n = 1$, we are back to the original de Bruijn sequences.

REMARK

The ordinary de Bruijn sequences have a two-dimensional counterpart: the $r \times v$-periodic sequences in which each possible content of an $n \times m$-window appears exactly once per period. E.g., in Figure 2.11, the 16 binary 2×2-matrices all appear in a 4×4-periodic array. The two-dimensional de Bruijn sequences have found applications in optics and communications (see e.g. [Etz]).

A natural extension of the work done here would be to investigate the existence of two-dimensional Generalized de Bruijn sequences. Stating this problem, we end the chapter.

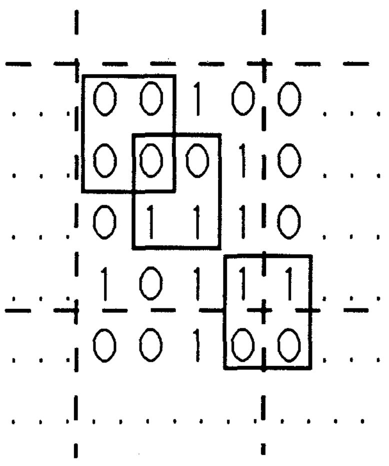


Figure 2.11.

References

[Ber] C. Berge, *Graphs*. North Holland Mathematical Library, 2nd revised edition, 1985.

[EB] T. van Aardenne–Ehrenfest and N. G. de Bruijn, "Circuits and Trees in Ordered Linear Graphs," *Simon Stevin*, vol.28, pp. 203–217, 1951.

[Etz] T. Etzion, "Constructions for Perfect Maps and pseudorandom arrays," *IEEE Trans. Information Theory*, vol.34, pp.1308–1316, September 1988.

[Fr] H.M. Fredericksen, "A Survey of Full Length Nonlinear Shift Register Cycle Algorithms," *SIAM Rev*, vol.24, pp. 195–221, April 1982.

[FSM] C. Flye-Sainte Marie, "Solution to Problem Number 58," *l'Intermediare des Mathematiciens*, 1 (1894), pp. 107–110.

[Gol] S. W. Golomb, *Shift Register Sequences*. Laguna Hills, CA: Aegean Park Press, 1982.