On Window Sequences and Position Location

Jonathan Roshan Tuliani

Royal Holloway and Bedford New College

University of London

Thesis submitted for the Degree of Doctor of Philosophy, 1998

Abstract

Window sequences with span n may be employed in certain one-dimensional position location applications, in which a movable device reads an n-window from a fixed representation of the sequence. Such systems require algorithms for determining the position within the chosen sequence of a given n-window from the sequence, a process called decoding.

De Bruijn sequences, due to their maximal period, give the optimum resolution for such systems for a given window size n. We present new constructions for de Bruijn sequences, together with decoding algorithms for these sequences. These improve significantly upon the storage requirements and/or the computational complexity of all previously known techniques for decoding de Bruijn sequences, for all but a very small proportion of the possible values of the alphabet size c and the span n of the sequence. Our results include a construction for binary de Bruijn sequences of arbitrary span, having a recursive decoding algorithm that has very low complexity and requires the storage of look-up tables of only small, constant size. We also exhibit c-ary span n sequences, for arbitrary n and any composite c, whose decoding procedure requires only slightly more computation than that required for a single decoding of a span n d-ary de Bruijn sequence, where d is any non-trivial factor of c.

Additionally, in order to provide an error control capability to the position location systems described above, we study the construction of sequences whose n-windows form an error control code for some minimum distance. We exhibit a one-to-one correspondence between sequences formed by overlapping words from

linear codes and certain families of linear recurring sequences. Inspired by the results of computer searches, we present theoretical results linking the existence of optimal sequences of a certain type to the existence of primitive polynomials with specific conditions on their coefficients. We also generalise this concept to two dimensions, and present a construction for the corresponding arrays that utilises both new and existing techniques.

Acknowledgements

Firstly, I am indebted to my supervisor, Professor Fred Piper, for his guidance, reassurance and calm confidence in my abilities, whatever concerns I may have had. I am also grateful to Professors Peter Wild and Chris Mitchell for the time and care they have devoted to reading my work, and for their many useful comments and ideas.

I should like to thank the staff and postgraduates in the Mathematics Department for making my time at Royal Holloway an enjoyable one. I am particularly grateful to Simon Blackburn and Kenny Paterson for always being available, and for many hours of interesting and enlightening discussion.

Thanks are also due to Stephen Margetts and Aileen Rabbitte for their time spent proof reading this thesis. The blame for any remaining mistakes rests purely with myself.

My studies have been greatly facilitated by financial support from the Engineering and Physical Sciences Research Council and from Hewlett-Packard Laboratories.

Finally, and most importantly, I must thank my family. Without their support, both financial and emotional, embarking on this course of study would never have been possible.

Contents

List of Figures

List of Tables

Notation

$\mathbb{F}_q$, $\mathbb{F}_q^*$	finite field of q elements, its multiplicative group
$\mathbb{F}_q[x]$	ring of polynomials in x over $\mathbb{F}_q$
$\mathbb{F}_q^n$	vector space of n-tuples over $\mathbb{F}_q$
$\mathbb{Z}$, $\mathbb{Z}_c$	the ring of integers, the ring of integers modulo c
$\mathbb{Z}_c^n, \mathcal{A}^n$	the set of n-tuples over $\mathbb{Z}_c$, or over some set $\mathcal{A}$
$\square$	end of Proof, Definition, Example, Remark, Construction, Problem or Algorithm
c-ary	pertaining to an alphabet of size c (page 20)
$\dim V$	the dimension of the vector space V
$\langle c_0, c_1, \ldots, c_n \rangle$	the linear span of the vectors c_i
$\gcd(n, m)$	the greatest common divisor of n and m
$\phi(n)$	Euler's totient function
$\mathrm{ord}(f(x))$	the order of the polynomial $f(x)$ (page 32)
O, Ω, Θ	expressions for orders of magnitude (page 25)
$\mathcal{G}_{c,n}$	the c-ary de Bruijn graph of order n (page 22)
$v \rightarrowtail w$	v and w represent adjacent vertices in $\mathcal{G}_{c,n}$ (page 22)
$\mathcal{H}_{q,n}$	the graph of the q-ary n-dimensional 'cube' (page 34)
$[s] = (s_i)_{i \in \mathbb{Z}}$	a sequence $[s]$ with terms $\ldots, s_{-1}, s_0, s_1, \ldots$ (page 19)
$\mathrm{lc}[s]$	the linear complexity of $[s]$ (page 27)
$\mathrm{wt}([s])$	the weight of the sequence $[s]$ (page 40)
$\mathrm{wt}(u)$	the (Hamming) weight of the tuple u (page 34)
$[s]_i^n$	the ith n-window of $[s]$, namely $(s_i, s_{i+1}, \ldots, s_{i+n-1})$ (page 20)
$a_{\lvert n \rvert}$	the n-tuple with all elements equal to a (page 20)

X_n	the n-tuple $(1,0,1,0,\ldots)$ (page 42)
$u \mid v$	u divides v, when these are integers, or
	the concatenation of the tuples u and v (page 97)
u^t	the t-fold concatenation of the tuple u with itself (page 97)
$\overline{u}$, $[\overline{s}]$	the bit-wise complement of the binary tuple u or sequence $[s]$
$\mathrm{d}(u,v)$	the (Hamming) distance between the tuples u and v (page 34)
E	left shift operator on sequences (page 27)
D	difference operator on tuples and sequences (page 39)
D_b^{-1}	the bth inverse difference operator (page 41)
$[s] \times [t]$	product of the sequences $[s]$ and $[t]$ (Definition 4.1)
$\mathrm{loc}([s], w)$	the least positive location of w in $[s]$ (Problem 1.3)
$\mathrm{loc}([s], w, f)$	the least positive location of w in $[s]$ that is congruent to f
	modulo m, where $[s]$ is an (n,m)-SDB sequence (Problem 4.4)
$[t] = \Phi[s]$	$[t]$ is formed by applying Construction 2.7 (the GL construction)
	to $[s]$ (Definition 46)
$[s] = \Upsilon[t]$	$[s]$ is formed by applying the interleaving construction to $[t]$
	(page 66)

(n,m)-SDB sequence	an (n,m) spaced de Bruijn sequence (Definition 4.2)
(n,t,c)-PF	an (n,t,c) perfect factor (Definition 4.22)
(n,t,m,c)-PMF	an (n,t,m,c) perfect multi-factor (Definition 4.23)
(n,δ)-CW sequence	an (n,δ) code window sequence (Definition 5.1)
(n,δ,m)-SCW sequence	an (n,δ,m) spaced code window sequence
	(Definition 5.40)
$(p,q;m,n;\delta)$-CW array	a $(p,q;m,n;\delta)$ code window array (Definition 5.47)

1 Introduction

1.1 Motivation and Outline

Consider a device moving linearly relative to a surface. In some applications, it may be desirable that such a device be able to measure its position relative to the surface. One way in which this may be achieved is to encode position-dependent information onto the surface, in such a way that sensors mounted on the device can read this information and so enable the device to determine its position. Such systems are examples of what are known as absolute type position transducers. These have practical advantages over other systems in that they require no initialisation (such as after a power failure) and that, unlike incremental systems, errors are not accumulated ([53]).

One obvious means of achieving such a scheme is to encode the surface with binary n-tuples representing the numbers from 0 to $2^n - 1$, in order. These n-tuples may be encoded in the form of n sequences of single binary digits, each parallel to the direction of movement. The moving device is able to read the n-tuple currently aligned with the sensor and thus obtain a measurement of its current position. Such a scheme is illustrated in Figure 1.1. Whilst elegant in its simplicity, this scheme has the disadvantage that the number of tracks required increases with the resolution of the system. This may have adverse space and cost implications.

As an alternative, it may be possible to record a single sequence of information symbols on the surface, and position the sensor to read a short segment of consecutive terms (known as a window) from this sequence (see Figure 1.2). In

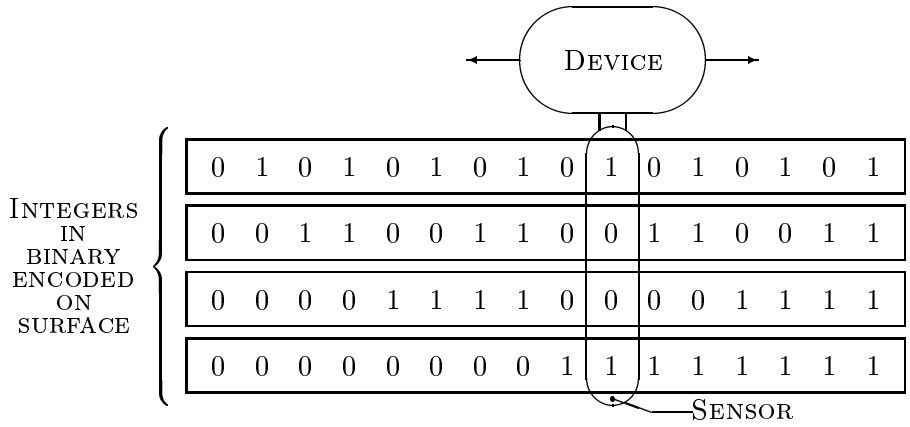

FIGURE 1.1. Position transducer using binary encoding.

order to determine its position uniquely, the device requires that no two windows of the sequence occurring at distinct positions are identical. Such sequences are called window sequences.

Due to the nature of the sensors likely to be employed, most practical applications call for binary window sequences. However, sequences over other alphabets may also be useful, and are also worthy of study. In particular, window sequences over alphabets of size 2^m, for example, may be employed by writing m binary sequences in parallel on the surface, and using an $m \times n$ array of sensors mounted to the device to read a window of n consecutive terms from each binary sequence. Such a dense array of sensors may have size advantages that offset

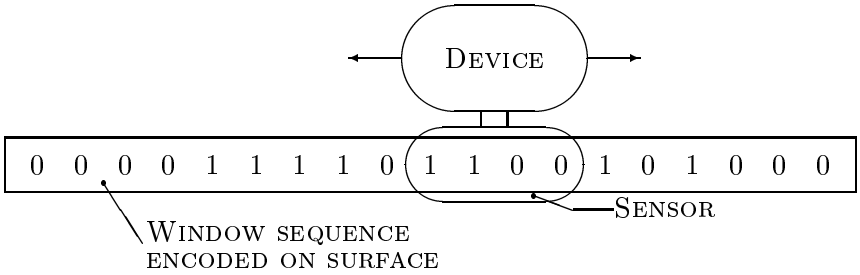

FIGURE 1.2. Position transducer using a window sequence.

the need to encode more than one track of information upon the surface. In this sense the scheme shown in Figure 1.1 is a special case of the window sequence based schemes, where the window length is one.

Each of the schemes described above appears at first sight to require that the device move in discrete steps, of size equal to the spacing of the symbols encoded on the surface. In practice, however, a continuously moving device is more likely to be required. Such a device would spend only a small fraction of its time aligned precisely with the symbols on the surface, and the remainder positioned between symbols. In the scheme of Figure 1.1, this problem may be overcome by re-ordering the integers, so that 'consecutive' integers have binary representations differing from each other in exactly one place. Such an ordering is called a Gray Code (see [29]). Unfortunately, this solution is not applicable for schemes based on window sequences. However, Petriu *et al.* demonstrate in [56] a simple means by which this problem may be overcome, and so we do not consider this issue further.

Whilst the schemes above involve linear motion, they may be adapted to the situation where the device moves cyclically. The window sequences required must then be repeating, or periodic. It is this latter case that we choose to study in this thesis. Clearly, it is simple to form a linear system using one cycle of a periodic sequence; in fact, this is the case in Figure 1.2.

Position transducers based on window sequences have the disadvantage that the position of the device needs to be computed from the sensor reading—the sensor no longer measures the absolute position of the device directly. That is, given a window of consecutive terms from the sequence used, we require a means

of computing the position of that window within the sequence. This is known as the decoding problem for the sequence. For the position of the device to be monitored continuously, the time that this decoding requires induces a limiting factor on how quickly the device may be allowed to move ([55]). Of course, rapid decoding may be achieved simply by storing a table of all windows of the sequence, together with their position. However, due to the potentially large number of windows involved, such an approach may be uneconomical, and so we seek an alternative solution.

Petriu *et al.* suggest just such a system for monitoring the position of automated guided vehicles (AGVs), such as those widely used in factories ([55], [56]). In their scheme, a binary m-sequence is encoded on the factory floor as a sequence of black and white marks, which are then read by an optical sensor mounted on each AGV. Due to their recurrence relation properties, m-sequences may be easily and rapidly generated using simple hardware. This feature is exploited to decode the m-sequence by computational means, rapidly cycling through the sequence searching for the desired window. This process has an expected running time that is exponential in the window size, and so Petriu *et al.* were able to achieve satisfactory AGV velocity only by parallelising part of their decoding algorithm and implementing it using purpose-built hardware ([54], [56]).

Clearly, in order to maximise the resolution of such position location systems in terms of the number of sensors employed, it is desirable that the sequence be as long as possible relative to the window size used. In this respect, m-sequences are well suited to this task (see Section 1.3). Unfortunately, it is known that the decoding problem for an m-sequence is equivalent to the discrete logarithm

problem in a particular finite field ([36]), a problem that is computationally non-trivial ([46]).

A de Bruijn sequence is a sequence in which every tuple of a particular size appears exactly once as a window. Perhaps the earliest known example of such a sequence is in the thousand-year old Sanskrit word *yamátárájabhánsalagám*, a nonsense word used as a mnemonic by Indian drummers. The pattern of long and short beats, represented by accented and un-accented vowels (ignore the consonants), contains every possible 3-tuple exactly once ([58]). De Bruijn sequences have more recently found a variety of other applications, which together with their mathematical elegance has lead to a significant quantity of research into their properties (see references in Section 1.2).

Clearly, as maximal length window sequences relative to their window size, de Bruijn sequences seem ideally suited to our particular application. Of particular significance to us, therefore, is the ease with which de Bruijn sequences may be decoded. Perhaps for this reason, or perhaps simply due to their natural mathematical interest, Chung *et al.* (as noted in [42]) list the decoding problem as one of the "fundamental questions" for future research on de Bruijn sequences ([10]).

Numerous constructions for de Bruijn sequences have been known for some time (see [22]); unfortunately most do not lend themselves to efficient decoding. For example, for some de Bruijn sequences, namely those that may be derived from m-sequences, the decoding problem is again equivalent to the discrete logarithm. However, recent past work has demonstrated the existence of de Bruijn sequences that may be decoded with significantly greater ease ([42],

[44, Section 5]). These results are theoretically significant. However, they are applicable only for very small subsets of the possible parameters. This implies that these techniques may not always be useful in practice, and also leaves the general theoretical question on the decodability of de Bruijn sequences open. Indeed, the authors of [42] conclude by commenting that "further refinement of existing techniques [for constructing decodable de Bruijn sequences] (possibly combined with new techniques) remains a desirable goal".

Chapter 2 of this thesis introduces a construction for binary de Bruijn sequences due to Lempel ([32]). We generalise this technique to arbitrary alphabet sizes. We also demonstrate how a decoding technique for Lempel's sequences employing reduced-size storage tables ([51]) generalises in a similar fashion. We then explain how refinements to the construction and the decoding procedure allow the storage requirements to be reduced still further. Thus the main result of this Chapter is the first construction (known to the author) for de Bruijn sequences of arbitrary alphabet size and span, that allow computationally efficient decoding with storage requirements significantly less than those of a naïve look-up table based approach (Theorem 2.17).

The interleaving construction of Mitchell *et al.* ([42]) is introduced in Chapter 3. It is shown that this may be used in conjunction with the methods of Chapter 2 to construct de Bruijn sequences for arbitrary alphabets and window sizes. The resulting sequences have a decoding procedure whose computational complexity is less than that for those sequences produced using the methods of Chapter 2 alone, and which has similar storage requirements (Theorem 3.7). Finally, in the case of binary alphabets, a relationship between the construc-

tions is exploited to eliminate the need for storage tables that grow with the window size (Theorem 3.18). The construction holds for all window sizes, and so may be of practical use. This is despite the fact that it is restricted to binary alphabets, since, as we have already remarked, this is the case most likely to occur naturally. This is the main result of this Chapter: a construction for binary de Bruijn sequences of arbitrary span that may be decoded without the use of storage tables by an algorithm of low computational complexity. Thus the problem of constructing easily decoded de Bruijn sequences can be considered 'solved' in the case of binary alphabets. An outline of the method used was suggested to the author by Professor C. Mitchell.

Suppose we have a de Bruijn sequence over some alphabet of size c, together with its decoding algorithm. Chapter 4 provides a means of using this sequence to construct another de Bruijn sequence of the same span, but with alphabet size $c \times d$ for any positive integer d (Theorem 4.14). The decoding problem for this new sequence may be solved using one decoding of the original sequence, together with one application of an algorithm of extremely low computational complexity that does not require the use of storage tables (Theorem 4.17). Thus the problem of constructing easily decoded de Bruijn sequences of arbitrary span and alphabet size is reduced to the subset of cases where the alphabet size is prime. For example, suppose that the first sequence above is a binary de Bruijn sequence constructed using the method of Chapter 3. Then together these Chapters provide a means of constructing easily decodable de Bruijn sequences of arbitrary span over any alphabet of even size.

Since easily decoded de Bruijn sequences of even span may be formed from

those of smaller, odd span by the interleaving construction ([42]), the only open problem in this area remains the construction of easily decoded de Bruijn sequences of arbitrary odd span and of arbitrary odd prime alphabet size. Moreover, the current best known constructions in these cases are the non-binary sequences of the first part of Chapter 3, or those of Chapter 2. Theorem 4.21 summarises which techniques apply for all possible values of the parameters.

Suppose now that an error occurs in reading one or more digits from a de Bruijn sequence employed in a position location scheme. The device may then compute its position as being far from its actual position. What is required is a means of detecting or correcting these errors. The theory behind the control of errors in communications applications is well developed; a summary of relevant results is presented in Section 1.4. In Chapter 5 we attempt to use this theory to construct window sequences that are resilient to sensor read errors. Central to this is the construction of window sequences whose windows differ from each other in at least a given number of positions. The larger this number, the better equipped the system will be to manage errors effectively. We study in detail the relationship between windows of sequences that satisfy linear recurrence relations (see Section 1.3) and linear codes. We also present an example of a less algebraic, more combinatorial construction for these sequences. The Chapter ends by using some of the techniques we have developed, together with an existing method, to give a construction for two-dimensional arrays with an analogous window property to the sequences considered previously. These arrays may assist in the control of errors in two-dimensional position location schemes similar to the one-dimensional schemes described above. Such a scheme is used,

for example, in a recently developed instrument for measuring the shape of the cornea. The control of errors of precisely the kind considered above is a central performance issue for this device ([59]).

We conclude by considering what directions future research in this area might take. This includes a detailed reasoning as to why the latter part of Chapter 3 is difficult to generalise to non-binary alphabets, together with a number of suggestions for areas of investigation that seem likely to yield useful results.

We assume that the reader is schooled in linear algebra and the algebra of finite fields. There are many texts available regarding the former; for the latter see [33]. We also assume familiarity with some basic concepts from Graph Theory to provide alternative descriptions of some of the combinatorial objects we consider. Other background results are presented in the remainder of this Chapter, together with formal definitions of many of the concepts and terms mentioned in this introduction. The introductions to the other Chapters contain more specialised background material. The remainder, except where clearly noted, constitutes the author's own results.

1.2 De Bruijn Sequences

Formally, we define a *sequence* over a set $\mathcal{A}$ to be a map $s : \mathbb{Z} \to \mathcal{A}$. To indicate that the map s is a sequence we shall use square brackets, as in $[s]$. We shall often write $[s] = (s_i)_{i \in \mathbb{Z}}$, implying $s_i = s(i)$ for all $i \in \mathbb{Z}$, and refer to the s_i as the *terms, entries* or *elements* of $[s]$. Henceforth, we adopt the more intuitive notion of a sequence as a bi-infinite list of elements of $\mathcal{A}$, indexed by elements of $\mathbb{Z}$.

The set $\mathcal{A}$ we refer to as the *alphabet*. We shall be concerned only with alphabets whose size (cardinality) is both finite and at least 2. Whenever $|\mathcal{A}| = c$ we may say that the sequence $[s]$ is *c-ary*. We will often use the ring $\mathbb{Z}_c$ as an example of a c-ary alphabet, as well as the set of integers $\{0, 1, \ldots, c-1\}$. At times we may use elements of the latter to represent elements of the former, and *vice versa*, in a natural if slightly informal manner. Our intended meaning should always be clear.

Let n be any integer, $n \geqslant 1$. By an *n-tuple* over $\mathcal{A}$ we mean an element of $\mathcal{A}^n$. For any $a \in \mathcal{A}$, we write $a_{|n|}$ to denote the n-tuple all of whose entries are equal to a. If $x = (x_0, x_1, \ldots, x_{n-1}) \in \mathcal{A}^n$ has the property that, for some $k \in \mathbb{Z}$, we have $x_i = s_{k+i}$ for all i in the range $0 \leqslant i \leqslant n-1$ then we say that x is an *n-window* of $[s]$, and that x *appears in* $[s]$ *at position* k. Clearly, given $[s]$, n and k there is a unique n-tuple x with this property, which we denote $[s]_k^n$.

For any $\ell \in \mathbb{Z}, \ell \neq 0$ we say that $[s]$ has *period* ℓ if and only if $s_i = s_{i+\ell}$ for all $i \in \mathbb{Z}$. Such a sequence is called *periodic*, and may be defined by giving a *generating cycle* of the sequence, namely ℓ consecutive terms of the sequence starting at position 0. These generating cycles we also denote using square brackets: for example, $[s] = [012]$ denotes the sequence $[s] = (s_i)_{i \in \mathbb{Z}}$ given by $s_i = i \bmod 3$. Note that $[0]$ denotes the all zero sequence. The least, positive period of $[s]$ is called the *least period*, and the generating cycle of this length is called *minimal*.

Clearly, if x is an n-tuple appearing in a sequence $[s]$ of least period ℓ then it does so at an infinite number of positions. Thus we are free to say that x appears exactly once in $[s]$, or that x is a unique n-window of $[s]$, when we mean, strictly,

that x appears in $[s]$ at exactly one position k in the range $0 \leqslant k \leqslant \ell - 1$. We think of the minimal generating cycle of $[s]$, considered cyclicly, as containing exactly one copy of x.

Let n be any integer, with $n \geqslant 1$.

DEFINITION 1.1. A periodic sequence $[s]$ is a *window sequence with span* n if and only if every n-window of $[s]$ is unique. □

That is, if $[s]$ has least period ℓ, then the ℓ windows of length n starting at positions $0, 1, \ldots, \ell - 1$ are distinct. It has been shown that c-ary window sequences of span n and least period ℓ exist for all values of c, ℓ and n with $c \geqslant 2$, $n \geqslant 1$ and $1 \leqslant \ell \leqslant c^n$ ([15], [31]).

DEFINITION 1.2. A *de Bruijn sequence with span* n is a periodic sequence $[s]$ in which every n-tuple over the alphabet appears exactly once in $[s]$. □

The first known study of de Bruijn sequences was made in 1894 by Flye-Sainte Marie ([21]), who demonstrated their existence for all values of $n \geqslant 1$ and $c \geqslant 2$ and even determined the precise number of such sequences: there are

$$(c!)^{c^{n-1}}$$

(distinct) de Bruijn sequences of span n over any alphabet of size c. The sequences are named, however, after N. D. G. de Bruijn, who re-discovered them in 1946. They are of considerable interest as combinatorial objects, and their properties have been extensively studied ([1], [5], [9], [10], [13], [14], [16], [18], [19], [20], [22], [24], [25], [26], [32], [42], [51], [58]).

Clearly, de Bruijn sequences are window sequences with maximal least pe-

riod. Since there are c^n distinct n-tuples over any c-ary alphabet, the least period of a c-ary de Bruijn sequence of span n must be c^n.

There is another, graph theoretic way in which to envisage a de Bruijn sequence, introduced by de Bruijn ([7]) and Good ([26]). Fix $c \geqslant 2$ and $n \geqslant 1$, and define the *de Bruijn graph* to be the directed graph $\mathcal{G}_{c,n} = (V, E)$ where $V = \mathbb{Z}_c^n$, and the relation $E \subseteq V \times V$ is defined as follows: let v_1 and v_2 be elements of V, written in coordinates as $v_i = (v_i^0, v_i^1, \ldots, v_i^{n-1})$ for $i = 1, 2$. We say $(v_1, v_2) \in E$, and write $v_1 \rightarrowtail v_2$, if and only if $v_1^j = v_2^{j-1}$ for all j in the range $1 \leqslant j \leqslant n-1$. Now, let $S = (v_0, v_1, \ldots, v_{c^n-1})$ be a sequence of vertices in $\mathcal{G}_{c,n}$ that define a Hamiltonian cycle (that is, S is a closed cycle visiting each vertex of $\mathcal{G}_{c,n}$ exactly once). Again, write $v_i = (v_i^0, v_i^1, \ldots, v_i^{n-1})$, this time for $i = 0, 1, \ldots, c^n - 1$. Then the sequence $[s]$ of period c^n with a generating cycle $[v_0^0 v_1^0 \cdots v_{c^n-1}^0]$ has $[s]_i^n = v_{i \bmod c^n}$ for all $i \in \mathbb{Z}$, and so is a c-ary de Bruijn sequence of span n. This construction may be reversed, so we see that there is a 1-1 correspondence between c-ary de Bruijn sequences of span n and Hamiltonian cycles in $\mathcal{G}_{c,n}$.

The position location application for window sequences given in Section 1.1 requires a means of determining the position of a given window within the sequence. In the case of de Bruijn sequences, we give a more formal statement of this problem.

PROBLEM 1.3 (The decoding problem for de Bruijn Sequences). Let n and c be integers with $n \geqslant 1$ and $c \geqslant 2$ and let $[s]$ be a de Bruijn sequence of span n over an alphabet $\mathcal{A}$ of size c. The decoding problem for $[s]$ is to determine, for any $w \in \mathcal{A}^n$, the unique integer h in the range $0 \leqslant h \leqslant c^n - 1$ such that $[s]_h^n = w$.

□

We write $h = \mathrm{loc}([s], w)$ to show that h represents the solution to Problem 1.3 for a given $[s]$ and w. Generalising this, where $[t]$ is a window sequence of span n and where w is an n-window of $[t]$, we write $\mathrm{loc}([t], w)$ to denote the least, positive position at which w appears in $[t]$.

Our motivation for studying Problem 1.3 centres on the need to construct sequences for which it can be solved efficiently. In order to compare alternative methods, we require a fair and meaningful measure of the efficiency of any proposed algorithm. Any such measure requires its assumptions to be carefully stated. However, we wish to formulate our efficiency measure in a manner that captures the essential features of the algorithm concerned, without unnecessary emphasis being placed on details specific to a particular implementation. That is, we would like our efficiency measure to be as implementation-independent as possible.

We assume that the algorithm concerned is to be implemented in software on a single-processor digital computer. This seems reasonable given the current technological climate. In this scenario there are three obvious factors that we may use as measures of the efficiency of a given algorithm, namely:

COMPLEXITY The number of operations performed by the processor;

STORAGE REQUIREMENT The amount of data that needs to be stored in 'look-up tables', that may be referenced by the algorithm, and

MEMORY REQUIREMENT The amount of memory required for storage of run-time variables.

We describe more fully our approach to measuring these three quantities below:

COMPLEXITY Each algorithm is described in terms of integer operations on its variables. If the values of the variables are small enough to be contained in a single computer register, then the number of these integer operations can be thought of as a good measure of the number of processor operations. In these days of 32-bit computing this does not seem unreasonable in practice, even though the variables involved in decoding a c-ary de Bruijn sequence of span n may contain values as large as c^n. This approach does have shortcomings for large c and n, where the above assumption does not hold. However, to address this properly requires assumptions to be made regarding the mode of storage used for such large integers and the algorithms used to operate upon them, which is contrary to our desire to achieve an implementation-independent assessment.

There are a number of distinct operations that we perform on our variables: assignment, comparison, addition, subtraction, multiplication, division, exponentiation and reduction modulo another variable. We think of each of these as single operations. It should be noted that in some implementations certain of these operations may take longer to perform that others; our complexity measure can then serve as an approximation only, but we note that it should be simple to extend our assessment to a more detailed one by counting the numbers of each operation separately. However, we do consider separately the number of times more complex operations need be performed, such as the solution of a pair of simultaneous congruences using the Chinese Remainder Theorem.

STORAGE REQUIREMENT We assume that each item of stored data whose size is bounded above by N requires $\log N$ bits of storage, and ignore what

rounding may occur.

MEMORY REQUIREMENT Different algorithms use different numbers of variables, each requiring run-time memory. In addition, the memory requirement can be increased with the use of recursive algorithms, which may require many sets of variables to be stored simultaneously. Note, however, that the value of every useful run-time variable needs to be computed at some stage, and that this requirement is already under consideration. Thus we do not consider the memory requirement to be as significant as the complexity and storage requirements already discussed, and do not consider it further.

When measuring the complexity or storage requirements of an algorithm, we adopt the convention of being concerned mainly with the order of magnitude of the quantity concerned, rather than its exact value. We give these measures in terms of the input variables best describing the 'size' of the problem under consideration; for the de Bruijn sequence decoding problem this is normally the alphabet size c and the span n. We measure these quantities using the 'big-O' notation. For functions f and g of n we write

(1) $f = O(g)$ when f/g is bounded above (intuitively, f is of order of magnitude at most g);

(2) $f = \Omega(g)$ when $g = O(f)$ (intuitively, f is of order of magnitude at least g), and

(3) $f = \Theta(g)$ when $f = O(g)$ and $g = O(f)$ (intuitively, f and g are of equal orders of magnitude).

This approach is broadly consistent with that presented for algorithms already

in the literature (see [42] or [51], for example). For a more detailed presentation of the theory of algorithm analysis, see [2].

1.3 Linear Recurring Sequences

Let $[s] = (s_i)_{i \in \mathbb{Z}}$ be a sequence over the finite field $\mathbb{F}_q$. We say that $[s]$ satisfies an *rth order homogenous linear recurrence relation* if and only if there exist constants $c_0, c_1, \ldots, c_r$ with $c_r = 1$ such that

$$\sum_{i=0}^{r} c_i s_{i+j} = 0 \quad \text{for all } j \in \mathbb{Z}. \tag{1.1}$$

The sequence $[s]$ is called an *rth order linear recurring sequence.* Notice that we consider only bi-infinite sequences, that is sequences satisfying equation (1.1) for all $j \in \mathbb{Z}$ and not just $j \geqslant 0$. Such sequences may be easily generated, as any r consecutive digits determine the next digit, and so the remainder of the sequence, uniquely. This process is easily implemented in software or in hardware. In the latter case, the circuit used to generate the sequence is called a *linear feedback shift register* or LFSR (see Figure 1.3). A full description of the operation of these devices may be found in, for example, [52, Chapter 7].

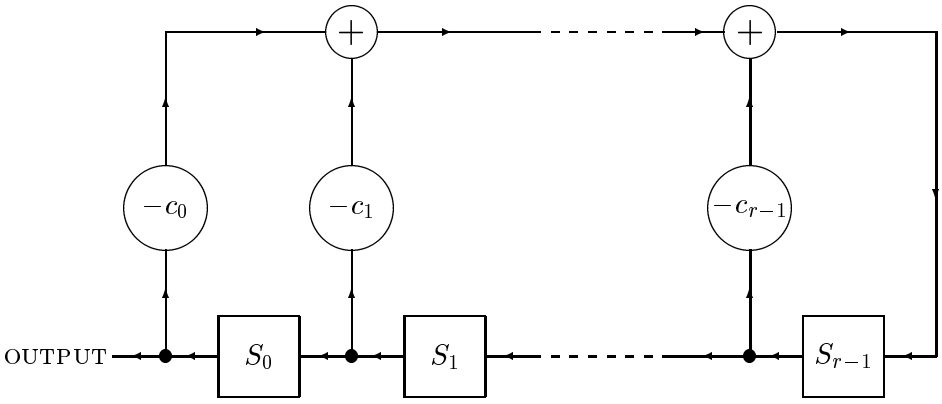

FIGURE 1.3. An r-stage linear feedback shift register.

If $[s]$ is a sequence with period p, then it satisfies the recurrence relation $s_{i+p} = s_i$ for all $i \in \mathbb{Z}$, and so is also a linear recurring sequence. Moreover, any linear recurring sequence is periodic. For, let $[s]$ be a sequence satisfying equation (1.1) for some r. There are only a finite number of possible strings of r consecutive digits. Thus, eventually, a string of r consecutive digits must occur in two different places in the sequence. Now if $c_0 \neq 0$ then any r consecutive digits determine not only the next digit of the sequence but also the previous digit. Thus the entire sequence is determined, and we deduce that the sequence $[s]$ is periodic. Finally, notice that if $c_0 = 0$ then since $[s]$ is bi-infinite, it satisfies a linear recurrence relation of lower order in which $c_0 \neq 0$, and the above argument may then be repeated.

Denote by E the *left shift operator* on sequences. That is, we write $[s] = E[t]$ if and only if $[s]$ and $[t]$ are sequences with $s_i = t_{i+1}$ for all $i \in \mathbb{Z}$. Defining $c(x) = \sum_{i=0}^{r} c_r x^r$, equation (1.1) may be written more succinctly as

$$c(E)[s] = [0].$$

We say that $c(x)$ is a *characteristic polynomial* for $[s]$. The set of all characteristic polynomials of $[s]$ form an ideal in $\mathbb{F}_q[x]$, and since $\mathbb{F}_q[x]$ is a Principal Ideal Domain, this ideal must be generated by a unique monic polynomial $m(x)$. We call this polynomial the *minimal polynomial* of $[s]$, and call the degree of $m(x)$ the *linear complexity* of $[s]$, denoted $\mathrm{lc}[s]$. By minimality and the bi-infinite nature of $[s]$ we have $x \nmid m(x)$. In other words, we have shown that for any periodic sequence $[s]$ there is a unique homogenous linear recurrence relation of least order generating $[s]$, and this order is called the linear complexity of $[s]$. Alternatively, the linear complexity of $[s]$ can be thought of as the number of

stages in the shortest LFSR generating $[s]$.

Another way to imagine the generation of the sequence $[s]$ is as follows. It is really an algebraic analogue of the action of the LFSR. Let $[s]$ be an rth order homogenous linear recurring sequence, and let $\mathbf{v}_i = [s]_i^r$. If $\mathbf{v}_i = (v_i^0, v_i^1, \ldots, v_i^{r-1})$, then the stage S_j of the LFSR contains $v_i^j = s_{i+j}$ at time i. The next state function of the LFSR can be described by the operation

$$\mathbf{v}_{i+1} = \mathbf{v}_i \mathbf{M}$$

where $\mathbf{M}$ is the *companion matrix* for the minimal polynomial $m(x) = x^r + m_{r-1}x^{r-1} + \cdots + m_0$ of $[s]$, described by

$$\mathbf{M} = \left(\begin{array}{ccccc|c} 0 & 0 & \cdots & 0 & 0 & -m_0 \\ 1 & 0 & \cdots & 0 & 0 & -m_1 \\ 0 & 1 & \cdots & 0 & 0 & -m_2 \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & -m_{r-2} \\ 0 & 0 & \cdots & 0 & 1 & -m_{r-1} \end{array}\right).$$

The output of the LFSR at time i is simply the contents of stage S_0 at time i, that is a projection of the vector $\mathbf{v}_i$ onto its first coordinate. Thus the sequence $[s]$ is given by

$$s_i = \mathbf{v}_0 \mathbf{M}^i \mathbf{e}_0^\top \quad \text{for all } i \in \mathbb{Z} \tag{1.2}$$

where $\mathbf{v}_0$ is the vector $(s_0, s_1, \ldots, s_{r-1})$ and $\mathbf{e}_0$ is the first standard basis vector. Notice that since $x \nmid m(x)$ we have $m_0 \neq 0$ and so the matrix $\mathbf{M}$ is invertible. Thus the least period of $[s]$ is also the order of $\mathbf{M}$ in the group $\mathrm{GL}(r, \mathbb{F}_q)$.

Let $\mathbf{R}$ be a $k \times k$ matrix over $\mathbb{F}_q$. $\mathbf{R}$ is *cyclic* if and only if there exists some $\mathbf{m} \in \mathbb{F}_q^k$ such that the set of vectors $\{\mathbf{m}_i \mid \mathbf{m}_i = \mathbf{m}\mathbf{R}^i \text{ for all } i \in \mathbb{Z}\}$ spans $\mathbb{F}_q^k$.

We call $\mathbf{m}$ a *cyclic vector for* $\mathbf{R}$. It may be shown that $\mathbf{R}$ is cyclic if and only if $\mathbf{R}$ has minimal polynomial of degree k. Since $(\mathbf{R}^i)^\top = (\mathbf{R}^\top)^i$, $\mathbf{R}$ and $\mathbf{R}^\top$ share the same minimal polynomial. Moreover, $\mathbf{R}$ has minimal polynomial $m(x)$ of degree k if and only if $\mathbf{R}$ is conjugate to the companion matrix $\mathbf{M}$ of $m(x)$. These results may be found in [12, Chapter 11].

The following results are of fundamental importance to us. They are well known, but for completeness we give proofs.

THEOREM 1.4. *Let* $[s]$ *be a sequence over* $\mathbb{F}_q$ *with linear complexity* r*, and let* V *be the vector space generated by the set of* n*-windows of* $[s]$*, for some fixed* $n > 0$*. Then* $\dim V = \min\{r, n\}$.

PROOF. Let $[s]$ have minimal polynomial $m(x)$ of degree r, and let $\mathbf{v}_i = [s]_i^r$, for all $i \in \mathbb{Z}$. Then the $\mathbf{v}_i$ are related by the next state map

$$\alpha : \mathbb{F}_q^r \to \mathbb{F}_q^r$$

$$\alpha(\mathbf{v}_i) = \mathbf{v}_{i+1} = \mathbf{v}_i\mathbf{M} \tag{1.3}$$

where $\mathbf{M}$ is the companion matrix of $m(x)$. Now let C be the space spanned by the r-windows of $[s]$. We claim that C has dimension r. For, if $\dim C = \ell < r$, then restricting to C, the map $\alpha : C \to C$ has minimal polynomial of degree $h \leqslant \ell$. Denote this polynomial by $f(x) = \sum_{i=0}^{h} f_i x^i$ (where $f_h = 1$). Consider the jth term of the sequence $f(E)(s)$: this is given by

$$\begin{aligned}
(f(E)(s))_j &= \sum_{i=0}^{h} f_i s_{i+j} \\
&= \sum_{i=0}^{h} f_i \mathbf{v}_{i+j} \mathbf{e}_0^\top \\
&= \left(\sum_{i=0}^{h} f_i \alpha^i(\mathbf{v}_j) \right) \mathbf{e}_0^\top
\end{aligned}$$

$$
\begin{aligned}
&= (f(\alpha)(\mathbf{v}_j))\,\mathbf{e}_0^\top \\
&= 0 \quad \text{for all } j \in \mathbb{Z},
\end{aligned}
$$

the last step following by the definition of $f(x)$. Thus $f(x)$ is a characteristic polynomial for $[s]$ having degree less than r, a contradiction. We deduce that C has dimension r.

Now let V be the space spanned by the n-windows of $[s]$. If $n > r$, notice that the leftmost r digits of each n-window determine the entire n-window uniquely. Moreover, the rightmost $n - r$ digits are linearly dependent on the leftmost r digits. Thus, $\dim V = \dim C = r$.

If $n < r$, suppose that $\dim V = n - a$ for some $a \geqslant 0$. By extending the n-windows to r-windows, observe that $\dim C \leqslant r - a$. As $\dim C = r$ we have $a = 0$ and $\dim V = n$ as required. □

The following result follows from Theorem 1.4 and the fact that if two sequences share a given characteristic polynomial, then so does their term-wise sum.

COROLLARY 1.5. *Let $m(x)$ be a monic polynomial of degree k, with $x \nmid m(x)$. The collection of all sequences with characteristic polynomial $m(x)$ forms a k-dimensional vector space.*

THEOREM 1.6. *Let $m(x)$ be a monic polynomial of degree k, with $x \nmid m(x)$. A periodic sequence $[s]$ has minimal polynomial $m(x)$ if and only if $[s]$ satisfies*

$$
s_i = \mathbf{t}\mathbf{R}^i\mathbf{x}^\top \quad \textit{for all } i \in \mathbb{Z}, \tag{1.4}
$$

for some $\mathbf{R}$, $\mathbf{t}$ and $\mathbf{x}$, where $\mathbf{R}$ is a $k \times k$ matrix with minimal polynomial $m(x)$, $\mathbf{t}$ is a cyclic vector for $\mathbf{R}$ and $\mathbf{x}$ is a cyclic vector for $\mathbf{R}^\top$.

PROOF. Let $[s]$ be a sequence with minimal polynomial $m(x)$. We have already shown that $[s]$ may be generated by the relation in equation (1.2). Clearly the companion matrix $\mathbf{M}$ of $m(x)$ has minimal polynomial $m(x)$ of degree k and so is cyclic. It remains to show that $\mathbf{v}_0 = [s]_0^n$ is a cyclic vector for the companion matrix $\mathbf{M}$ of $m(x)$, and that the first standard basis vector $\mathbf{e}_0$ is a cyclic vector for $\mathbf{M}^\top$. To see the former, note that for all $i \in \mathbb{Z}$ we have $[s]_i^n = \mathbf{v}_i = \mathbf{v}\mathbf{M}^i$, and that $[s]_i^n$ spans $\mathbb{F}_q^k$ by Theorem 1.4. For the latter, observe that for $0 \leqslant i < k$, the ith standard basis vector $\mathbf{e}_i$ is equal to $\mathbf{e}_0(\mathbf{M}^\top)^i$.

Conversely, let $\mathbf{R}$ be any $k \times k$ matrix with minimal polynomial $m(x)$. Let $\mathbf{t}$ be a cyclic vector for $\mathbf{R}$ and let $\mathbf{x}$ be a cyclic vector for $\mathbf{R}^\top$. Define the sequence $[s]$ by the formula of equation (1.4). For all $j \in \mathbb{Z}$ we have

$$\begin{aligned}
\sum_{i=0}^{i=k} m_i s_{i+j} &= \sum_{i=0}^{i=k} m_i \mathbf{t}\mathbf{R}^{i+j}\mathbf{x}^\top \\
&= \mathbf{t}\mathbf{R}^j \left(\sum_{i=0}^{i=k} m_i \mathbf{R}^i \right) \mathbf{x}^\top \\
&= 0,
\end{aligned}$$

and so $[s]$ has characteristic polynomial $m(x)$.

Since $\mathbf{x}$ is cyclic for $\mathbf{R}^\top$, there exists some $n > 0$ for which the set of vectors $\{\mathbf{b}_i \mid \mathbf{b}_i = \mathbf{x}(\mathbf{R}^\top)^i \text{ for all } 0 \leqslant i \leqslant n\}$ spans $\mathbb{F}_q^k$. Thus the matrix $\mathbf{B} = \left(\begin{array}{c|c|c|c} \mathbf{b}_0^\top & \mathbf{b}_1^\top & \cdots & \mathbf{b}_n^\top \end{array} \right)$ has rank k. Define $\mathbf{m}_i = \mathbf{t}\mathbf{R}^i$ for all $i \in \mathbb{Z}$. Now $[s]_i^n = \mathbf{m}_i\mathbf{B}$. Since $\mathbf{t}$ is a cyclic vector for $\mathbf{R}$, the $\mathbf{m}_i$ span $\mathbb{F}_q^k$. Thus $[s]_i^n$ spans a space of dimension k, and by Theorem 1.4 $[s]$ must have $\mathrm{lc}[s] \geqslant k$. Thus the minimal polynomial of $[s]$ must be $m(x)$. □

Let $[s]$ be a sequence over $\mathbb{F}_q$ with minimal polynomial $m(x)$. By the definition of a principal ideal, every characteristic polynomial $c(x)$ of $[s]$ must be

a multiple of $m(x)$. In particular, $[s]$ has period e if and only if $m(x) \mid x^e - 1$. Thus the least period of $[s]$ is the least e such that $m(x) \mid x^e - 1$. This value e is also known as the *order* of $m(x)$, and denoted $\mathrm{ord}(m(x))$.

Let α be an element of $\mathbb{F}_{q^r}^*$, the multiplicative group of $\mathbb{F}_{q^r}$. We denote by $m_\alpha(x) \in \mathbb{F}_q[x]$ the minimal polynomial of α, that is the unique monic polynomial generating the ideal $\{f(x) \in \mathbb{F}_q[x] \mid f(\alpha) = 0\}$. The order of α in $\mathbb{F}_{q^r}^*$ is the least e such that $\alpha^e = 1$. However, this can be so if and only if $m_\alpha(x) \mid x^e - 1$, and so $\mathrm{ord}(\alpha) = \mathrm{ord}(m_\alpha(x))$. Since $\mathbb{F}_{q^r}^*$ is a cyclic group with $q^r - 1$ elements, we have $e \mid q^r - 1$. Also, there exist $\phi(q^r - 1)$ distinct generators of $\mathbb{F}_{q^r}^*$. These are called *primitive elements*.

Let α be primitive in $\mathbb{F}_{q^r}$. Then $\alpha^q, \alpha^{q^2}, \ldots, \alpha^{q^{r-1}}$ are also primitive, and it may be shown that

$$m_\alpha(x) = (x - \alpha)(x - \alpha^q) \cdots (x - \alpha^{q^{r-1}}) \in \mathbb{F}_q[x] \tag{1.5}$$

and so $m_\alpha(x)$ has degree r. Thus for fixed q and r there exist $\phi(q^r - 1)/r$ distinct monic polynomials over $\mathbb{F}_q$ with maximal order $q^r - 1$. These are called *primitive polynomials*.

Fix $r > 0$, and let $\mathcal{S}$ be the set of all non-zero sequences $[s]$ over $\mathbb{F}_q$ satisfying a homogenous linear recurrence relation of the form of equation (1.1), where the c_i are taken to be the coefficients of a primitive polynomial $m(x)$ of degree r over $\mathbb{F}_q$. The least period of $[s]$ is $\mathrm{ord}(m(x)) = q^r - 1$. Since each $[s] \in \mathcal{S}$ satisfies a rth order linear recurrence relation, any r consecutive digits of such a sequence determines the remainder of the sequence uniquely. Thus the $q^r - 1$ r-windows of $[s]$ must consist of all possible non-zero r-tuples over $\mathbb{F}_q$, each

appearing exactly once. It follows that all the sequences in $\mathcal{S}$ are cyclic shifts of each other. We call such sequences *maximal length linear recurring sequences*, or *m-sequences*. (The terms pseudo-random sequence and pseudo-noise sequence or PN-sequence are also employed by some authors.) Due to their numerous useful properties, these sequences are widely used in a variety of mathematical and practical applications. We list those properties that will be relevant to us; for a more complete list see [37].

SHIFT PROPERTY If $[s] \in \mathcal{S}$, then $E[s] \in \mathcal{S}$.

WINDOW PROPERTY Every non-zero r-tuple over $\mathbb{F}_q$ appears as a window in each $[s] \in \mathcal{S}$.

ADDITION PROPERTY If $[s], [t] \in \mathcal{S}$, then so is their term-wise sum $[s] + [t]$.

BALANCE PROPERTY Each $[s] \in \mathcal{S}$ contains q^{r-1} instances of each of the digits of $\mathbb{F}_q$, except for the digit 0 which occurs $q^{r-1} - 1$ times.

SHIFT AND ADD PROPERTY If $[s] \in \mathcal{S}$, then $[s] + E^i[s] \in \mathcal{S}$ for all $i \not\equiv 0 \pmod{q^m - 1}$.

1.4 Error Control Codes

Coding Theory involves modelling the typical communications system as consisting of three main components: the *source*, the *channel* and the *receiver*. The source wishes to transmit a *message* to the receiver. To do so, he must send the message across the channel, where it may be subject to interference, or *noise*. The consequence of this noise is that the received message may be only an approximation to the original—it may contain *errors*. The object of

Coding Theory is to add extra data to the message, which the receiver may use to detect or correct these errors. This extra data, called *redundancy*, is necessarily dependent on the message, and as such cannot be used by the source to send any essentially new information to the receiver. To maximise efficiency, we wish to obtain the error control capacity we require with as little redundancy as possible.

Typically, a *block code* is used. In this, the message is encoded as strings of fixed length k over some alphabet. (In practice, $\mathbb{F}_2$ is normally used, although much of the theory holds over other alphabets.) In place of each message string, a *codeword* of $n > k$ digits is transmitted across the channel. Thus there are $n - k$ digits of redundancy with which to provide error control. Notice however that it is not necessary for each message to be contained explicitly within its corresponding codeword—we merely require that there be a 1-1 correspondence between the set $\mathcal{M}$ of all possible messages and the set $\mathcal{C}$ of all possible codewords. We call $\mathcal{C}$ a *code*, and the process of assigning a codeword in $\mathcal{C}$ to a given message in $\mathcal{M}$ we call *encoding*.

The *(Hamming) weight* of a codeword c, denoted $\mathrm{wt}\,(c)$, is defined to be the number of non-zero terms in c. The *(Hamming) distance* $\mathrm{d}(c_1, c_2)$ between two codewords c_1 and c_2 is the number of positions in which they differ. The *minimum (Hamming) distance* (or simply *distance*) of a code $\mathcal{C}$ is the least minimum distance between any two distinct codewords.

Let $\mathcal{H}_{q,n}$ be the graph with the set of all n-tuples over $\mathbb{F}_q$ as vertices, two vertices being adjacent if and only if they are at distance exactly 1. The words of a code may be thought of as representing a subset of the vertices of this graph.

The minimum distance of the code is the length of the shortest path between any two of these codeword vertices.

It is clear that in order to be able to detect any pattern of $d-1$ or fewer errors in the transmission of a codeword, it is necessary and sufficient that $\mathcal{C}$ have distance at least d. In this case, it is also clear that if $t \leqslant \lfloor \frac{d-1}{2} \rfloor$ errors have occurred in a received word r, then there is a unique codeword $c \in \mathcal{C}$ at distance at most $\lfloor \frac{d-1}{2} \rfloor$ from r. The process of finding the corresponding transmitted word given the received word is called *error correction*, and the process of transforming the transmitted word back into the original message is called *decoding*. (That this 'nearest neighbour' error correction technique (for $t \leqslant \lfloor \frac{d-1}{2} \rfloor$) gives the codeword most likely to have been transmitted given that r was received requires certain assumptions to be made regarding the type of errors encountered on the channel. Details may be found in [57, Chapter 2], but we affirm that channels in which all possible errors occur with equal, small probability, independently of each other and of the digit being transmitted, do behave in this fashion.)

Some systems do not use error correction—if the received word is a codeword, then it is decoded into a message, but otherwise the error is logged, or perhaps a request is sent from the receiver back to the source asking for re-transmission. These are called error detecting systems.

To allow as many different messages as possible we desire that the code $\mathcal{C}$ have as many codewords as possible. The maximum number of codewords in a length n code with distance d over an alphabet of size b is denoted $A_b(n, d)$.

Much of the work in Coding Theory is involved with one of two problems: finding codes $\mathcal{C}$ with as many words as possible given a fixed word length n and minimum distance d, and finding efficient error correcting algorithms (in terms of memory and computational requirements) for known codes. Most successful approaches to each of these problems require placing some extra conditions on the code $\mathcal{C}$, giving it a mathematical structure in order to facilitate analysis. The most widely used technique is to require $\mathcal{C}$ to be a linear code, that is a k-dimensional subspace of $\mathbb{F}_q^n$. In the remainder of this Section, we outline some basic results regarding these linear codes.

Suppose that $\mathcal{C}$ is a k-dimensional subspace of $\mathbb{F}_q^n$. We say that $\mathcal{C}$ is an $[n, k]$ code, or an $[n, k, d]$ code when $\mathcal{C}$ is known to have minimum distance exactly d. Since $\mathcal{C}$ has q^k words we may take $\mathcal{M} = \mathbb{F}_q^k$.

Let $\{\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{k-1}\}$ be any basis for $\mathcal{C}$. (Notice that since $\mathcal{C}$ is a vector space, we now write the codewords and messages as vectors.) Then the $k \times n$ matrix $\mathbf{G}$ formed by taking this basis as rows is called a *generator matrix* for $\mathcal{C}$. The mapping $\mathcal{M} \to \mathcal{C}$ given by $\mathbf{m} \mapsto \mathbf{mG}$ for all $\mathbf{m} \in \mathcal{M}$ is a 1-1 correspondence between $\mathcal{M}$ and $\mathcal{C}$. Thus each generator matrix induces a natural procedure for encoding messages into codewords.

For any non-singular $k \times k$ matrix $\mathbf{A}$, the map $\mathcal{M} \to \mathcal{M}$ given by $\mathbf{m} \mapsto \mathbf{mA}$ is a 1-1 correspondence—it is simply a change of basis in $\mathcal{M}$. Thus if $\mathbf{G}$ is a generator matrix of $\mathcal{C}$, so is $\mathbf{AG}$ for all non-singular $k \times k$ matrices $\mathcal{A}$. Each such generator matrix is distinct, and every generator matrix for $\mathcal{C}$ may be formed in this way.

Given a code $\mathcal{C}$ with q^k codewords, we says that $\mathcal{C}$ is *systematic* on a set of k distinct coordinate positions if and only if each k-tuple appears in those coordinates in exactly one codeword. These k coordinate positions can then be thought of as containing the message information, and the remaining positions as redundancy. When $\mathcal{C}$ is linear, this is equivalent to the square matrix obtained by considering only those columns of the generator matrix corresponding to chosen coordinate positions being non-singular.

In certain applications, such as voice telecommunications, it may be acceptable for a certain number of errors not to be corrected by the code. The probability with which this occurs may be computed from the weight distribution of a linear code, namely the number of codewords of each weight. However, in the position location scenario, undetected errors could have more serious consequences, and would not be tolerated. Thus it is of more interest for us to focus on the minimum distance of the code, since this determines what number of errors can be managed with complete certainty.

Since $\mathcal{C}$ is linear, the sum or difference of any two codewords is also a codeword. If two codewords are at distance d, then their difference is a codeword of weight d. Thus the minimum distance of $\mathcal{C}$ is equal to the least weight of the non-zero codewords in $\mathcal{C}$.

We denote the null space of $\mathcal{C}$ by $\mathcal{C}^\perp$. This is an $(n-k)$-dimensional subspace of $\mathbb{F}_q^n$, with the property that if $\mathbf{c} \in \mathcal{C}$ and $\mathbf{h} \in \mathcal{C}^\perp$ then $\mathbf{c} \cdot \mathbf{h} = 0$. Observe that since $\mathcal{C}^\perp$ determines $\mathcal{C}$ uniquely, words of $\mathcal{C}$ can be defined to be those words orthogonal to any basis $\{\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{n-k-1}\}$ of $\mathcal{C}^\perp$. Writing $\mathbf{H}$ to denote the

$(n-k) \times n$ matrix having the $\mathbf{h}_i$ as rows, we see that

$$\mathbf{c} \in \mathcal{C} \Longleftrightarrow \mathbf{c}\mathbf{H}^\top = \mathbf{0}. \tag{1.6}$$

Since in the binary case this is equivalent to a set of $n-k$ conditions each governing the parity of the sum of a subset of digits in $\mathbf{c}$, $\mathbf{H}$ is called a *parity check* matrix for $\mathcal{C}$.

We may also consider $\mathcal{C}^\perp$ as a code, with generator matrix $\mathbf{H}$ and parity check matrix $\mathbf{G}$. This code is called the *dual* of $\mathcal{C}$.

Notice from equation (1.6) that there exists a 1-1 correspondence between codewords of $\mathcal{C}$ and linear dependence relations on the columns of $\mathbf{H}$. We deduce the following well-known characterisation of the minimum distance of a linear code:

THEOREM 1.7. *Let* $\mathbf{H}$ *be a parity check matrix for a linear code* $\mathcal{C}$. *Then the minimum distance of* $\mathcal{C}$ *is equal to the least positive number of linearly dependent columns in* $\mathbf{H}$.

We shall also mention two popular techniques for reducing the length of a code. By *puncturing* a code, we refer to the technique of deleting one or more coordinate positions from all the codewords of the code, and identifying any equal codewords that result. When the coordinates chosen all lie at one end of the codewords, we say that the code is *truncated*. Alternatively, *shortening* a code is to again remove one or more coordinate positions from the code, but then to consider only those codewords that had a zero entry in all of the coordinates removed. Notice that puncturing may reduce the minimum distance of the code, whereas shortening leaves the minimum distance unchanged.

2 Decodable de Bruijn Sequences I

2.1 Introduction

We now describe the *difference operator*, D, first introduced to the study of de Bruijn sequences in the binary case by Lempel in 1970 ([32]). We shall define the difference operator on sequences over $\mathbb{Z}_c$, for arbitrary c, since this is the alphabet we shall most often use. Notice, however that we do not use the multiplication operation in $\mathbb{Z}_c$, and we could equally well have given these results over an arbitrary additive cyclic group. We shall later use the operator D over finite fields $\mathbb{F}_q$. Of course, these behave like $\mathbb{Z}_q$ when q is prime, which is the only case we shall require.

Let $[s] = (s_i)_{i\in\mathbb{Z}}$ be any sequence over $\mathbb{Z}_c$. We define the sequence $[t] = (t_i)_{i\in\mathbb{Z}}$ over $\mathbb{Z}_c$ by

$$t_i = s_{i+1} - s_i \quad \text{for all } i \in \mathbb{Z},$$

and write $[t] = D[s]$. With the notation of the left-shift operator E introduced in Section 1.3, we have that $D = E - 1$. We extend the operator D to act also as a map from $\mathbb{Z}_c^n$ to $\mathbb{Z}_c^{n-1}$, for any $n \geqslant 2$. Given $u = (u_0, u_1, \ldots, u_{n-1}) \in \mathbb{Z}_c^n$ and $v = (v_0, v_1, \ldots, v_{n-2}) \in \mathbb{Z}_c^{n-1}$, we write $v = Du$ in the event that

$$v_i = u_{i+1} - u_i \quad \text{for all } i \text{ in the range } 0 \leqslant i \leqslant n-2.$$

Clearly, for any $n \geqslant 2$, if the n-tuple u appears in the sequence $[s]$ at position $p \in \mathbb{Z}$, then Du appears in $D[s]$ at position p. The results in Lemmas 2.1–2.3 and Remark 2.4 are simple and well known generalisations of some of Lempel's original work.

LEMMA 2.1. *Let n and c be integers, with $n, c \geqslant 2$. Then the map $D : \mathbb{Z}_c^n \to \mathbb{Z}_c^{n-1}$ is a graph homomorphism from $\mathcal{G}_{c,n}$ to $\mathcal{G}_{c,n-1}$ (where the vertices of these de Bruijn graphs are considered to lie in $\mathbb{Z}_c^n$ or $\mathbb{Z}_c^{n-1}$ respectively).*

Thus, working over $\mathbb{Z}_c$, vertices u and v in $\mathcal{G}_{c,n}$ have $u \rightarrowtail v$ if and only if $Du \rightarrowtail Dv$ in $\mathcal{G}_{c,n-1}$.

LEMMA 2.2. *Let n and c be integers, with $n, c \geqslant 2$, and let $v = (v_0, v_1, \ldots, v_{n-2})$ be any element of $\mathbb{Z}_c^{n-1}$. Then v has c distinct pre-images under D, given by $u = (u_0, u_1, \ldots, u_{n-1}) \in \mathbb{Z}_c^n$ where*

$$u_i = \begin{cases} b & \text{if } i = 0, \text{ or} \\ b + \displaystyle\sum_{j=0}^{i-1} v_j & \text{if } 1 \leqslant i \leqslant n-1 \end{cases}$$

and where b is any element of $\mathbb{Z}_c$.

Let c be an integer, $c \geqslant 2$, and let $[s]$ be a sequence of least period ℓ over $\mathbb{Z}_c$. We define the *weight* of $[s]$, denoted $\mathrm{wt}([s])$, by $\mathrm{wt}([s]) = \sum_{i=0}^{\ell-1} s_i \in \mathbb{Z}_c$.

LEMMA 2.3. *Let $[s]$ be a sequence of least period ℓ and weight w over $\mathbb{Z}_c$. Then $[s]$ has c distinct pre-images under D, given by $[t] = (t_i)_{i \in \mathbb{Z}}$ where*

$$t_i = \begin{cases} b & \text{if } i = 0; \\ b + \displaystyle\sum_{j=0}^{i-1} s_j & \text{if } i > 0, \text{ or} \\ b - \displaystyle\sum_{j=i}^{-1} s_j & \text{if } i < 0, \end{cases}$$

and where b is any element of $\mathbb{Z}_c$. Moreover, the least period of $[t]$ is ℓ if $w = 0$ and $c\ell/\gcd(c, w)$ otherwise.

In each case, regardless of whether D is being considered as a map on n-tuples or sequences, there are c distinct pre-images under D, which are distinguished

by the choice of b. We write D_b^{-1} to denote the map which returns the pre-image identified by the given value b.

REMARK 2.4. We will be particularly concerned with pre-images of c-ary sequences $[s]$ of weight w and least period ℓ for which $\gcd(c, w) = 1$. In this case, each pre-image is a shift of $D_0^{-1}[s]$. For any $(n-1)$-tuple v appearing exactly once in $[s]$, the c pre-images of v under D each appear in $D_0^{-1}[s]$. Moreover they are equally spaced in $D_0^{-1}[s]$ and their order is determined by $\mathrm{wt}([s])$—explicitly, if $D_b^{-1}v$ appears in $D_0^{-1}[s]$ at position h then $D_{b+w}^{-1}v$ appears in $D_0^{-1}[s]$ at position $h+\ell$, $D_{b+2w}^{-1}v$ appears in $D_0^{-1}[s]$ at position $h+2\ell$, and so on. (If v occurs more than once in the minimal generating cycle of $[s]$ then the above still holds, with the exception that each repetition of v gives rise to a distinct family of pre-images of v in $D_0^{-1}[s]$, each family having the structure described above.) □

The following result is well known, but for completeness we give a proof.

LEMMA 2.5. *Let n and c be positive integers, with $n \geqslant 1$ and $c \geqslant 2$, and let $[s]$ be a span n de Bruijn sequence over $\mathbb{Z}_c$. Then $\mathrm{wt}([s]) = 0$ if and only if $n \geqslant 2$ or $n = 1$ and $2 \nmid c$.*

PROOF. Every possible n-tuple over $\mathbb{Z}_c$ appears exactly once in $[s]$, and each term of $[s]$ appears as the first digit of exactly one of these n-tuples. Thus each digit of $\mathbb{Z}_c$ must occur c^{n-1} times in the minimal generating cycle of $[s]$, giving $\mathrm{wt}([s]) = c^{n-1}\sum_{\ell=0}^{c-1}\ell \bmod c = c^n(c-1)/2 \bmod c$. If $n \geqslant 2$ this is 0. However, if $n = 1$ then it is clear that $\mathrm{wt}([s]) = 0$ if and only if $2 \nmid c$. □

We now describe the construction for de Bruijn sequences of arbitrary span n

given by Lempel ([32]). The construction is specific to binary alphabets, $c = 2$. We let X_r denote the r-tuple $(1, 0, 1, 0 \ldots)$, for all $r \geqslant 1$, and use $\overline{X_r}$ to denote its bit-wise complement.

CONSTRUCTION 2.6 (Lempel's 'cycle joining' Construction ([32])). To begin, let $[s] = (s_i)_{i \in \mathbb{Z}}$ be a binary de Bruijn sequence of span r, with $r \geqslant 2$. Construct the sequences $[u] = (u_i)_{i \in \mathbb{Z}}$ and $[v] = (v_i)_{i \in \mathbb{Z}}$ by $[u] = D_0^{-1}[s]$ and $[v] = D_1^{-1}[s]$, so $[u] = [\overline{v}]$. Notice that since $1_{|r|}$ appears in $[s]$, either X_{r+1} or $\overline{X_{r+1}}$ appears in $[u]$. We assume the former case, with X_{r+1} appearing in $[u]$ at position p, say; the latter case is similar. Then, $\overline{X_{r+1}}$ appears in $[v]$ at position p. The sequence with minimal generating cycle

$$[u_0, u_1, \ldots, u_{p+r}, v_{p+r}, v_{p+r+1}, \ldots, v_{2^r-1},$$

$$v_0, v_1, \ldots, v_{p+r-1}, u_{p+r+1}, u_{p+r+2}, \ldots, u_{2^r-1}]$$

is a de Bruijn sequence of span $r + 1$. □

The validity of this method stems from the fact that every $(r + 1)$-tuple occurs exactly once as an $(r+1)$-window in exactly one of $[u]$ or $[v]$, and that the minimal generating cycles of these sequences may be joined, in the manner given, into a single generating cycle, without affecting this window property (hence the name 'cycle-joining').

Given a de Bruijn sequence of span r, the cycle joining Construction may be applied repeatedly to yield a de Bruijn sequence of arbitrary span $n \geqslant r$. The resulting sequences have a structure that Paterson and Robshaw exploit to give a procedure for decoding these sequences ([51]). Their approach requires the storage of some tables of information, which are smaller than the amount of

information that would have to be stored in order to implement decoding by a direct look-up table.

Section 2.2 presents a generalisation of Lempel's construction to arbitrary alphabets. Section 2.3 then demonstrates that the Paterson-Robshaw decoding method may also be generalised to decode these sequences. Finally, Section 2.4 gives a technique by which the storage requirements for this decoding method may be further reduced, at the expense of a small amount of extra computation.

2.2 The Generalised Lempel Construction

Let c, k and ℓ be positive integers, with $c, \ell \geqslant 2$ and $0 \leqslant k \leqslant \ell - 1$, and let $[s] = (s_i)_{i \in \mathbb{Z}}$ be a sequence over $\mathbb{Z}_c$ of least period ℓ. We define the map χ_k on $[s]$ by $[t] = \chi_k[s]$ where $[t]$ is a sequence over $\mathbb{Z}_c$ of period $\ell - 1$ with generating cycle $[t_0, t_1, \ldots, t_{\ell-2}]$ given by

$$t_i = \begin{cases} s_i & \text{for } 0 \leqslant i \leqslant k-1, \text{ or} \\ s_{i+1} & \text{for } k \leqslant i \leqslant \ell - 2. \end{cases}$$

Thus a generating cycle for $[t]$ may be formed by deleting the digit in position k from the minimal generating cycle for $[s]$. For $[s]$, c, and ℓ as above, and for any k in the range $0 \leqslant k \leqslant \ell$ and for any $w \in \mathbb{Z}_c$, we also define the map $\rho_{k,w}$ on $[s]$ by $[r] = \rho_{k,w}[s]$, where $[r]$ is a sequence over $\mathbb{Z}_c$ of period $\ell + c$ with generating cycle $[r_0, r_1, \ldots, r_{\ell+c-1}]$ given by

$$r_i = \begin{cases} s_i & \text{for } 0 \leqslant i \leqslant k-1; \\ w + i - k \bmod c & \text{for } k \leqslant i \leqslant k+c-1, \text{ or} \\ s_{i-c} & \text{for } k+c \leqslant i \leqslant \ell+c-1. \end{cases} \tag{2.1}$$

Thus a generating cycle for the sequence $[r]$ may be obtained by inserting the c-tuple $(w, w+1, \ldots, w+c-1)$, with terms reduced modulo c, into the minimal generating cycle for $[s]$ at position k.

CONSTRUCTION 2.7 (The Generalised Lempel (GL) Construction). Let n and c be integers, with $n \geqslant 1$ and $c \geqslant 2$ and with $2 \nmid c$ if $n = 1$. Let $[s] = (s_i)_{i\in\mathbb{Z}}$ be a span n de Bruijn sequence over $\mathbb{Z}_c$. We define the sequence $[t]$ as follows:

(1) let $k = \mathrm{loc}([s], 1_{|n|})$;

(2) let $[s'] = (s'_i)_{i\in\mathbb{Z}} = \chi_k[s]$;

(3) let $[\hat{s}] = (\hat{s}_i)_{i\in\mathbb{Z}} = D_0^{-1}[s']$;

(4) let $p = (c-1)(c^n - 1) + k$ and let $w = \hat{s}_p$, and

(5) let $[t] = (t_i)_{i\in\mathbb{Z}} = \rho_{p,w}[\hat{s}]$.

□

REMARK 2.8. Since $[s]$ is a de Bruijn sequence we have that the least period of $[s']$ is $c^n - 1$. Also, since $s_k = 1$, applying Lemma 2.5 gives $\mathrm{wt}([s']) = c - 1$. By Lemma 2.3, together with the fact that for all $c \geqslant 2$ we have that $\gcd(c, c-1) = 1$, $[\hat{s}]$ has least period ℓ say, where $\ell = c(c^n - 1)$. Since $k \leqslant c^n - 1$ we have $p \leqslant \ell$, and so step (5) of Construction 2.7 is well defined. □

EXAMPLE 2.9. Let $[s] = (s_i)_{i\in\mathbb{Z}}$ be the span 3 de Bruijn sequence over $\mathbb{Z}_3$ given by

$$[s] = [00022212202112102012001110 1].$$

Notice that $k = \mathrm{loc}([s], 1_{|3|}) = 22$, and so $[s'] = \chi_k[s]$ is given by

$$[s'] = [00022212202112102012001101].$$

Then $[\hat{s}] = D_0^{-1}[s']$ is

$$\begin{aligned}[\hat{s}] \quad = \quad & [000021010221202002202220 11 \\ & 22221020211012122112111200 \\ & 11110212100201011001000\underline{0}122].\end{aligned} \tag{2.2}$$

We have $p = 2 \times 26 + 22 = 74$, and so $w = \hat{s}_p = 0$ (this digit is underlined in equation (2.2) above). Inserting the 3-tuple (012) at position 74 gives

$$\begin{aligned}[t] \quad = \quad & [000021010221202002202220 11 \\ & 22221020211012122112111200 \\ & 11110212100201011001000120122],\end{aligned}$$

a de Bruijn sequence of span 4 over $\mathbb{Z}_3$. □

LEMMA 2.10. *The sequence* $[t]$ *given by Construction 2.7 is a de Bruijn sequence of span* $n+1$ *over* $\mathbb{Z}_c$.

PROOF. By Remark 2.8, we have that $[\hat{s}]$ has least period $c(c^n - 1)$. Moreover, by Remark 2.4 every pre-image under D of each of the n-windows of $[s']$ appears exactly once as an $(n+1)$-window of $[\hat{s}]$. Since every n-tuple over $\mathbb{Z}_c$ except for $1_{|n|}$ appears as an n-window in $[s']$, the only $(n+1)$-tuples not appearing as $(n+1)$-windows of $[\hat{s}]$ are those whose image under D is $1_{|n|}$, namely the $(n+1)$-tuples $x_b = (b, b+1, \ldots, b+n)$ (with terms reduced modulo c) for each $b \in \mathbb{Z}_c$.

Since $1_{|n-1|}$ appears in $[s']$ at position k, we must have the n-tuple $(v, v+1, \ldots, v+n-1)$ (terms modulo c) appearing at position k in $[\hat{s}]$, where $v = \hat{s}_k$. By Remark 2.4 a similar n-tuple $(w, w+1, \ldots, w+n-1)$ (terms modulo c)

appears at position $p = (c-1)(c^n - 1) + k$ in $[\hat{s}]$, with $w = \hat{s}_p$. We now consider what happens to $[\hat{s}]$ when we apply the map $\rho_{p,w}$ to obtain the sequence $[t]$. Clearly, for $0 \leqslant j \leqslant p - n - 1$, we have $[\hat{s}]_j^{n+1} = [t]_j^{n+1}$. By choosing p and w so that x_w appears at position p, and by the definition of $\rho_{p,w}$, we have that $\hat{s}_\ell = t_\ell$ for $p \leqslant \ell \leqslant p + n - 1$. (It may be useful to consider the cases $n < c$, $n = c$ and $n > c$ separately.) Thus $[\hat{s}]_j^{n+1} = [t]_j^{n+1}$ for $p - n \leqslant j \leqslant p - 1$. Also, $[t]_j^{n+1} = x_{w+j-p \bmod c}$ for $p \leqslant j \leqslant p + c - 1$. Finally, since $[t]$ has the same generating cycle as $[\hat{s}]$ except for the insertion of a c-tuple at position p, we have that $[\hat{s}]_j^{n+1} = [t]_{j+c}^{n+1}$ for $p \leqslant j \leqslant c^{n+1} - c - 1$. Notice that every $(n+1)$-window of $[\hat{s}]$ appears exactly once in $[t]$, as does each $(n+1)$-tuple x_b. Thus $[t]$ is a span $n+1$ de Bruijn sequence as required. □

DEFINITION 2.11. We write $[t] = \Phi[s]$ in the event that $[t]$ is a span $n+1$ de Bruijn sequence obtained from a span n de Bruijn sequence $[s]$ using Construction 2.7. □

The value of p given ensures that $D[\hat{s}]_p^n = 1_{|n-1|}$; it is this property that is central to the Construction's validity. By Remark 2.4, any choice of $p = i(c^n - 1) + k$ with i in the range $0 \leqslant i \leqslant c - 1$ would also have this property. In Section 2.4 we shall impose the condition that $[s]_0^n = 0_{|n|}$. This will mean that the constant $(n+1)$-windows of $[\hat{s}]$ appear at positions $i(c^n - 1)$ for i in the range $0 \leqslant i \leqslant c - 1$. We have chosen the largest of all the possible values of p, since this ensures that the c-tuple inserted in Step (5) of Construction 2.7 does not affect the position of these constant $(n+1)$-windows. This makes some of our later calculations simpler, and in particular is central to allowing us to derive equation (2.6), using Lemma 2.14 (to follow).

The description of this construction is different to that of the cycle joining Construction described in Section 2.1. However, in the case $c = 2$ the sequences that result are extremely similar, and can be made identical if the choice of p discussed above is made slightly differently. In fact, there is no obstacle to generalising the cycle joining construction directly, given a little ingenuity, and the results obtained are again the same as those obtained by Construction 2.7, up to the choice of p. We therefore think of Construction 2.7 as a generalisation of the cycle joining technique presented in the binary case by Lempel, and hence call it the Generalised Lempel Construction (hereafter the GL construction).

However, Construction 2.7 has a number of advantages over the cycle joining method. Firstly, we believe that presented in this way, the construction is more easily understood. This is particularly true in the non-binary case, where the repeated joining of cycles, at positions that have already been involved in earlier cycle joinings, are especially difficult to envisage. In addition, we have already mentioned the benefit we shall obtain from our choice of p when we later condition that the maximal all-zero window always appears at position 0 in our sequences.

As with the cycle joining technique, the procedure described above for constructing a span $n + 1$ de Bruijn sequence from one of span n may be applied repeatedly. That is, let $c \geqslant 2$ be a fixed alphabet size, and let r be any integer, with $r \geqslant 1$ and $2 \nmid c$ if $r = 1$. Then let $[s^{(0)}]$ be a span r de Bruijn sequence over $\mathbb{Z}_c$. We define the sequences $[s^{(i+1)}]$ for all $i \geqslant 0$ iteratively, using

$$[s^{(i+1)}] = \Phi[s^{(i)}].$$

(We also define the sequences $[s'^{(i)}] = (s_j'^{(i)})_{j\in\mathbb{Z}}$ and $[\hat{s}^{(i)}] = (\hat{s}_j^{(i)})_{j\in\mathbb{Z}}$ for all $i \geqslant 0$

to be those intermediate sequences arising in the construction of $[s^{(i+1)}]$ from $[s^{(i)}]$.) By Lemma 2.10, for each $n > r$ we have constructed a span n de Bruijn sequence $[s^{(n-r)}]$ over $\mathbb{Z}_c$ by applying Φ to $[s^{(0)}]$ a total of $n-r$ times.

2.3 Decoding

We now present a procedure for decoding the span n de Bruijn sequence $[s^{(n-r)}]$ constructed at the end of Section 2.2. The method is based on Paterson and Robshaw's procedure for reduced-storage decoding of Lempel's original binary de Bruijn sequences ([51]).

Suppose we have stored the following information:

(1) c-ary digits $L^{(i)}(j) = s_j^{(i)}$ for each i and j in the range $1 \leqslant i \leqslant n-r$ and $0 \leqslant j \leqslant c^{i+r-1} - 2$;

(2) integers $k^{(i)} = \mathrm{loc}([s^{(i)}], 1_{|r+i|})$ for i in the range $0 \leqslant i \leqslant n-r$, and

(3) integers $T(w) = \mathrm{loc}([s^{(0)}], w)$ for all $w \in \mathbb{Z}_c^r$.

The set of integers $\{T(w)\}$ is simply a table allowing the sequence $[s^{(0)}]$ to be decoded immediately. (If available, this could be replaced by a decoding algorithm for the sequence $[s^{(0)}]$.) The integers $k^{(i)}$ are those required by step (1) of the GL construction.

For any $i \geqslant 1$, we have that the digits of $L^{(i)}(\cdot)$ are simply the first $c^{r+i-1} - 1$ digits of $[\hat{s}^{(i-1)}]$. From Remarks 2.4 and 2.8, any element of $[s^{(i)}]$ may be found from a single reference to $L^{(i)}(\cdot)$. Explicitly,

$$\hat{s}_j^{(i-1)} = L^{(i)}(j \bmod (c^{r+i-1} - 1)) + (c-1)\left\lfloor \frac{j}{c^{r+i-1}-1} \right\rfloor \quad \text{for all } j \in \mathbb{Z}. \quad (2.3)$$

It is then simple to compute the terms of $[s^{(i)}]$ from those of $[\hat{s}^{(i-1)}]$ by a relationship like that given in equation (2.1).

Given this stored information, it is possible to implement a recursive decoding procedure for $[s^{(n-r)}]$. The idea behind this procedure is as follows. Suppose we wish to compute $j = \mathrm{loc}([s^{(n-r)}], w)$ for some $w \in \mathbb{Z}_c^n$. Firstly, compute $v = Dw \in \mathbb{Z}_c^{n-1}$. If $v \neq 1_{|n-1|}$ then v appears in $[s'^{(n-r-1)}]$, at position f, say. This is easily computed from the value of $k^{(n-r-1)}$ and the position of v in $[s^{(n-r-1)}]$, which may in turn be found by applying this decoding algorithm recursively. Since $[s'^{(n-r-1)}]$ is a window sequence with span $n-1$, v occurs at a unique position within a minimal generating cycle for $[s'^{(n-r-1)}]$. Since w is one of the c possible pre-images of v under D, by Remark 2.4 we are left with c possible locations for w in $[\hat{s}^{(n-r-1)}]$, which may be distinguished and located by examining the first digit of w and comparing it with $L^{(n-r)}(f)$. Once the position of w in $[\hat{s}^{(n-r-1)}]$ is known, the final position of w in $[s^{(n-r)}]$ is easily computed, again using the value of $k^{(n-r-1)}$. Otherwise, if $v = 1_{|n-1|}$ then w is one of the n-tuples introduced to $[s^{(n-r)}]$ in step (5) of the GL construction, and its location may be found accordingly. The algorithm is described more fully below.

ALGORITHM 2.12. (This is an outline only, although sufficient detail for full implementation should be clear.)

`Input` $c, L^{(i)}, T, k^{(i)}, n, r, w = (w_0, w_1, \ldots, w_{n-1})$, `for` $1 \leqslant i \leqslant n-r$

`Output` $j = \mathrm{loc}([s^{(n-r)}], w)$

`If` $n = r$

    `Return` $T(w)$

```
Else
```

Let $p = (c-1)(c^{n-1}-1) + k^{(n-r-1)}$

Compute $v = Dw$

If $v = 1_{|n-1|}$

Let $e = L^{(n-r)}(k^{(n-r-1)}) + (c-1)^2$

Return $p + ((w_0 - e) \bmod c)$

Else

Find $f = \mathrm{loc}([s^{(n-r-1)}], v)$ (use this algorithm recursively)

If $f > k^{(n-r-1)}$

$f = f - 1$

End if

Let $e = L^{(n-r)}(f)$

Let $j = f + (c^{n-1}-1)((w_0 - e) \bmod c)$

If $0 \leqslant j \leqslant p-1$

Return j

Else

Return $j + c$

End if

End if

End if

□

LEMMA 2.13. *Algorithm 2.12 correctly computes* $j = \mathrm{loc}([s^{(n-r)}], w)$ *for all* $w \in \mathbb{Z}_c^n$ *and all* $n \geqslant r$.

PROOF. It is clear that each step of the algorithm is well defined, it being easy to check that the variables used lie within appropriate ranges (such as f lying in the correct range for finding $L^{(n-r)}(f)$). Termination of the algorithm is assured by the fact that no looping operations occur except for that required, if necessary, by the recursive step. This step calls the Algorithm again, with the value of r the same and the value of n reduced by one. Thus after a finite number of recursions the algorithm must be called in a state when another recursion will not occur, either because $n = r$ or because v will have been computed as being all ones. An answer is always returned directly in these cases, so the recursion cannot continue indefinitely and termination with an answer is guaranteed.

We prove the correctness of the Algorithm by induction upon n. If $n = r$ then the algorithm outputs $T(w)$, which by definition is the correct decoding of w in $[s^{(0)}]$. We consider the case $n = N+1$, assuming by the inductive hypothesis that correct decoding occurs in the case $n = N$, where $N \geqslant r$. We suppose that we wish to find $j = \mathrm{loc}([s^{(N+1-r)}], w)$ for some $w \in \mathbb{Z}_c^{N+1}$. Letting $v = Dw$, suppose initially that the case $v = 1_{|N|}$ applies. By the same argument as used in the proof of Lemma 2.10, we know that $w = (w_0, w_0+1, \ldots, w_0+N)$ (with terms reduced modulo c), and that this $(N+1)$-tuple was not present in $[\hat{s}^{(N-r)}]$, being introduced into $[s^{(N+1-r)}]$ at position $p = (c-1)(c^N-1)+k^{(N-r-1)}$ by step (5) of the GL construction. By equation (2.3), the value $e = L^{(N+1-r)}(k^{(N-r)})+(c-1)^2$ is equal to $\hat{s}_p^{(N-r)}$. We deduce that w occurs in $[s^{(N+1-r)}]$ starting at a position $(w_0 - e) \bmod c$ digits after position p, as returned by the Algorithm.

Alternatively, suppose that $v \neq 1_{|N|}$. Firstly, the algorithm computes $f = \mathrm{loc}([s^{(N-r)}], v)$. If $f > k^{(N-r)}$ this is reduced by one, so now $f = \mathrm{loc}([s'^{(N-r)}], v)$.

We also store $e = \hat{s}_f^{(N-r)}$. By Remark 2.4, since $\mathrm{wt}([\hat{s}]) = c-1$ the pre-images of v under D lie in $[\hat{s}^{(N-r)}]$ at positions $f, f+c^N-1, f+2(c^N-1), \ldots, f+(c-1)(c^N-1)$. The first of these pre-images has its first digit equal to e, the next has its first digit equal to $e+(c-1)$, and so on. We require the pre-image whose first digit is equal to w_0; this must lie at position $j = f + (c^N-1)((w_0 - e) \bmod c)$ in $[\hat{s}^{(N-r)}]$. Finally, the computation of $[s^{(N+1-r)}]$ from $[\hat{s}^{(N-r)}]$ involves the insertion of c digits at position p, so if j is as much as this then c is added to it before the value is returned. □

We now address first the complexity and then the storage requirements of the above procedure. Excluding the recursion, the algorithm uses no looping procedures. Each iteration of the algorithm as described above requires $\Theta(n)$ integer operations, the most expensive step being the computation of v from w. Considering the recursion, we see that each decoding of $[s^{(n-r)}]$ requires additionally at most one decoding of each of the sequences $[s^{(n-r-1)}], [s^{(n-r-2)}], \ldots, [s^{(r+1)}]$, and possibly one look-up in the table T (to perform a decoding of $[s^{(r)}]$). Thus the algorithm has an overall complexity of $O(n(n-r))$ integer operations.

The storage requirements for Algorithm 2.12 are as follows:

(1) the values $L^{(i)}(\cdot)$, which require the storage of $c^{r+i-1}-1$ c-ary digits for each i in the range $1 \leqslant i \leqslant n-r$;

(2) the values $k^{(i)}$, which require the storage of one c^{r+i}-ary digit for each i in the range $0 \leqslant i \leqslant n-r$, and

(3) the values $T(w)$, which require the storage of c^r c^r-ary digits.

The total storage requirement demanded by the above is for $\Gamma(n,c,r)$ bits, where

$$\begin{aligned}
\Gamma(n,c,r) &= \sum_{i=1}^{n-r}(c^{r+i-1}-1)\log c + \sum_{i=0}^{n-r}\log(c^{r+i}) + c^r\log(c^r) \\
&= c^r\log c\cdot\sum_{i=0}^{n-r-1}c^i - (n-r)\log c + \log c\cdot\sum_{i=0}^{n-r}(r+i) + rc^r\log c \\
&= c^r\log c\cdot\frac{c^{n-r}-1}{c-1} - (n-r)\log c + \\
&\qquad \log c\cdot\left(r(n-r+1)+\frac{(n-r)(n-r+1)}{2}\right) + rc^r\log c \\
&= \log c\cdot\left[c^r\left(\frac{c^{n-r}-1}{c-1}+r\right)+\frac{n^2-n+3r-r^2}{2}\right] \qquad (2.4) \\
&= O(\log c\cdot(c^n+rc^r)) \qquad (2.5)
\end{aligned}$$

When $c=2$, the value given in equation (2.5) agrees with that stated in [51].

We should compare this requirement with the storage needed to implement decoding using a naïve look-up table. Such a table requires c^n c^n-ary digits, and so uses $O(\log c\cdot nc^n)$ bits of storage. For r small, the above method reduces this by a factor of n. (Notice that for $r=n$, of course, these two methods are identical.)

2.4 Reduced Storage Requirements

We present various means by which these storage requirements may be reduced, at the expense of a small extra amount of computation. These refinements may be applied for every value of c and n (except for some trivial examples, such as $n=r$), including the case $c=2$. Thus they are a refinement of the methods of Paterson and Robshaw.

Firstly, suppose that we specify that $[s^{(0)}]_0^r = 0_{|r|}$, a condition not required by Lempel or by Paterson and Robshaw. By our construction method, this

ensures that $[s^{(i)}]_0^{r+i} = 0_{|r+i|}$ for all $i \geqslant 0$. Keeping track of the position of the longest run of zeroes in each sequence allows us to keep track of its pre-images in the next sequences, namely the maximal runs of a constant digit. This idea is made precise in the following Lemma.

LEMMA 2.14. *Let $[s]$ be a span n de Bruijn sequence and let $[t] = \Phi[s]$. If $[s]$ has $[s]_0^n = 0_{|n|}$ then $[t]$ has $[t]_{i(c^n-1)}^{n+1} = (i(c-1) \bmod c)_{|n+1|}$, for all i in the range $0 \leqslant i \leqslant c-1$.*

PROOF. Let $[s']$ and $[\hat{s}]$ be defined as in the intermediate stages of the GL construction. Since $[s']_0^n = 0_{|n|}$, we have $[\hat{s}]_0^{n+1} = 0_{|n+1|}$. By Remarks 2.4 and 2.8 we have that $[\hat{s}]_{i(c^n-1)}^{n+1} = (i(c-1) \bmod c)_{|n+1|}$, for all i in the range $0 \leqslant i \leqslant c-1$.

Finally, since $[s]_0^n = 0_{|n|}$ we have that $k = \mathrm{loc}([s], 1_{|n|}) \geqslant n$, and thus that $p = (c-1)(c^n-1) + k \geqslant (c-1)(c^n-1) + n$. This shows that $\hat{s}_i = t_i$ for all i in the range $0 \leqslant i \leqslant (c-1)(c^n-1) + n - 1$. However, since $[t] = \rho_{p,w}[\hat{s}]$ with $w = \hat{s}_p$, we have that $t_i = \hat{s}_i$ for $i = (c-1)(c^n-1) + n$ also. Thus $[t]_{i(c^n-1)}^{n+1} = [\hat{s}]_{i(c^n-1)}^{n+1} = (i(c-1) \bmod c)_{|n+1|}$ for all i in the range $0 \leqslant i \leqslant c-1$, as required. □

We may use Lemma 2.14 to deduce that when the condition $[s^{(0)}]_0^r = 0_{|r|}$ is added to the GL construction, we have

$$k^{(i)} = (c-1)(c^{r+i-1} - 1) \tag{2.6}$$

for each $i \geqslant 1$, and so for these values of i the values $k^{(i)}$ may be calculated rather than stored. Also, the value of $k^{(0)}$ may be found from the table T, since $k^{(0)} = T(1_{|r|})$. Thus by using the GL construction with $[s^{(0)}]_0^r = 0_{|r|}$ we have avoided having to store the values $k^{(i)}$. This is one advantage our GL construction

has over sequences obtained by generalising the cycle joining approach directly.

However, storing the $k^{(i)}$ is a relatively minor concern, considering the other information that Algorithm 2.12 requires. By far the largest storage requirement is for the elements $L^{(i)}(j) = s_j^{(i)} = \hat{s}_j^{(i-1)}$ for each i and j in the ranges $1 \leqslant i \leqslant n - r$ and $0 \leqslant j \leqslant c^{r+i-1} - 2$. It is this requirement that we attempt to reduce in the remainder of this Section. For $i \geqslant 2$ we demonstrate a method by which these $L^{(i)}(j)$ may computed from another, significantly smaller storage table. By employing this procedure, the storage requirements for the tables $L^{(i)}$ may be replaced by the need to store these smaller tables.

For all $i \geqslant 0$ and all j in the range $1 \leqslant j \leqslant c^{r+i+1} - 1$ we define

$$N^{(i)}(j) = \sum_{\ell=0}^{j-1} \hat{s}_\ell^{(i)}.$$

Of course, all expressions that we give for $N^{(i)}(j)$ should always be evaluated modulo c. For all $i \geqslant 1$ we define

$$w^{(i)} = \hat{s}_p^{(i-1)},$$

where $p = (c-1)(c^{r+i-1} - 1) + k^{(i-1)}$. Thus $w^{(i)}$ is the value w obtained in step (4) of the construction of $[s^{(i)}]$ from $[s^{(i-1)}]$ using the GL construction.

LEMMA 2.15. *Let i be any integer, $i \geqslant 2$, and let j be any integer in the range $0 \leqslant j \leqslant c^{r+i-1} - 2$. Then it is possible to compute the value of $L^{(i)}(j)$, given the values of $w^{(i-2)}$, $N^{(i-2)}(c^{r+i-2} - 1)$ and $N^{(i-2)}(j''')$, for some j''' in the range $1 \leqslant j''' \leqslant c^{r+i-2} - 1$.*

PROOF. Firstly, we define

$$M^{(i)}(j) = \sum_{\ell=0}^{j-1} s_\ell^{(i)}$$

for all $i \geqslant 0$ and all j in the range $1 \leqslant j \leqslant c^{r+i} - 1$. Again, expressions for $M^{(i)}(j)$ are to be evaluated modulo c.

Suppose we are given $i \geqslant 2$ and j in the range $0 \leqslant j \leqslant c^{r+i-1} - 2$, and are required to find $L^{(i)}(j)$. We show how this may be achieved in three stages.

(1) Firstly, we show how for any j in the range above, the value of $L^{(i)}(j)$ may be found using (perhaps) a single value $M^{(i-1)}(j')$, for some j' in the range $1 \leqslant j' \leqslant c^{r+i-1} - 1$.

(2) Next, we show how for any j' in the range above, the value of $M^{(i-1)}(j')$ may be computed from the value of $w^{(i-2)}$ and a single value $N^{(i-2)}(j'')$, where j'' lies in the range $1 \leqslant j'' \leqslant c^{r+i-1} - c - 1$.

(3) Lastly, we show how for any such j'', the value of $N^{(i-2)}(j'')$ may be computed from the value $N^{(i-2)}(c^{r+i-2} - 1)$ and a single value $N^{(i-2)}(j''')$, where j''' lies in the range $1 \leqslant j''' \leqslant c^{r+i-2} - 1$.

Together, these three stages provide the means to achieve the computation required, and the result then follows.

STAGE (1). Notice that

$$L^{(i)}(j) = s^{(i)}(j) = \hat{s}^{(i-1)}(j) \tag{2.7}$$

when j lies in the range $0 \leqslant j \leqslant c^{r+i-1} - 1$, since the computation of $[s^{(i)}]$ from $[\hat{s}^{(i-1)}]$ does not affect the terms in this range. Notice also that since

$$\hat{s}^{(i-1)}(j) = \begin{cases} 0 & \text{if } j = 0, \text{ or} \\ \displaystyle\sum_{\ell=0}^{j-1} s'^{(i-1)}_{\ell} & \text{if } 1 \leqslant j \leqslant c^{r+i-1} - 1. \end{cases}$$

and since $[s'^{(i-1)}]$ is formed from $[s^{(i-1)}]$ simply by deleting the digit in position $k^{(i-1)}$, it is simple to express $L^{(i)}(j)$ as

$$L^{(i)}(j) = \begin{cases} 0 & \text{if } j = 0; \\ M^{(i-1)}(j) & \text{if } 1 \leqslant j \leqslant k^{(i-1)} - 1, \text{ or} \\ M^{(i-1)}(j+1) - 1 \bmod c & \text{if } k^{(i-1)} \leqslant j \leqslant c^{r+i-1} - 2. \end{cases}$$

Our proof of Stage (1) is thus complete.

STAGE (2). By the construction method, we have that $[s^{(i-1)}] = \rho_{p,w}[\hat{s}^{(i-2)}]$, where $p = (c-1)(c^{r+i-2} - 1) + k^{(i-2)}$ and $w = w^{(i-1)}$.

We claim that for any j' in the range $1 \leqslant j' \leqslant c^{r+i-1} - 1$ we have

$$M^{(i-1)}(j') = \begin{cases} N^{(i-2)}(j') & \text{for } 1 \leqslant j' \leqslant p; \\ N^{(i-2)}(p) + (j'-p)w & \text{for } p+1 \leqslant j' \leqslant p+c, \text{ or} \\ \quad + \dfrac{(j'-p-1)(j'-p)}{2} & \\ N^{(i-2)}(j'-c) + \dfrac{c(c-1)}{2} & \text{for } p+c+1 \leqslant j' \leqslant c^{r+i-1} - 1. \end{cases}$$

For $1 \leqslant j' \leqslant p$ this is immediate. For $p+1 \leqslant j' \leqslant p+c$ we have

$$\begin{aligned} M^{(i-1)}(j') &= \sum_{\ell=0}^{j'-1} s_\ell^{(i-1)} \\ &= N^{(i-2)}(p) + \sum_{\ell=p}^{j'-1} s_\ell^{(i-1)} \\ &= N^{(i-2)}(p) + w + (w+1) + \cdots + (w+j'-p-1) \\ &= N^{(i-2)}(p) + (j'-p)w + \frac{(j'-p-1)(j'-p)}{2} \end{aligned}$$

as required. Finally, for $p+c+1 \leqslant j' \leqslant c^{r+i-1} - 1$ we have that each of the terms $w, w+1, \ldots, w+c-1$ has been inserted into $[\hat{s}^{(i-2)}]$ to form $[s^{(i-2)}]$, and that all of these terms have occurred in the sum $M^{(i-1)}(j')$. Thus

$$M^{(i-1)}(j') = \sum_{\ell=0}^{j'-1} s_\ell^{(i-1)}$$

$$
\begin{aligned}
&= N^{(i-2)}(j'-c) + (0+1+\cdots+(c-1)) \bmod c \quad \text{(re-ordering)} \\
&= N^{(i-2)}(j'-c) + \frac{c(c-1)}{2} \bmod c.
\end{aligned}
$$

In each case, a single value $N^{(i-2)}(j'')$ is required. Since $k^{(i-2)}$ is the starting position of an $(r+i-2)$-tuple in $[s^{(i-2)}]$, we have $k^{(i-2)} \leqslant c^{r+i-2} - (r+i-2)$. Thus we have

$$
\begin{aligned}
p &= (c-1)(c^{r+i-2}-1) + k^{(i-2)} \\
&\leqslant (c-1)(c^{r+i-2}-1) + c^{r+i-2} - (r+i-2) \\
&= c^{r+i-1} - c - r - i + 3 \\
&\leqslant c^{r+i-1} - c,
\end{aligned}
$$

the last inequality following from the fact that $r \geqslant 1$ and $i \geqslant 2$. From this bound on p, and the range of j' used, we have shown that in each case above the value $N^{(i-2)}(j'')$ required has j'' in the range $1 \leqslant j'' \leqslant c^{r+i-1} - c$. Stage (2) is complete.

Stage (3). For brevity, we write P in place of $c^{r+i-2} - 1$. We are concerned with finding $N^{(i-2)}(j'')$ for any j'' in the range $1 \leqslant j'' \leqslant cP$. We divide this range into c equal sections, $tP + 1 \leqslant j'' \leqslant (t+1)P$, with t taking each value in the range $0 \leqslant t \leqslant c-1$. Each section contains P terms. This is motivated by the fact that $N^{(i-2)}(j) = \sum_{\ell=0}^{j-1} \hat{s}_\ell^{(i-2)}$, where $[\hat{s}^{(i-2)}] = D_0^{-1}[s'^{(i-2)}]$, the sequence $[s'^{(i-2)}]$ having weight $c-1$ and least period P.

Let $t = 0$. The required result is immediate, as j'' lies in the range $1 \leqslant j'' \leqslant P$. Next, suppose $t = 1$, so j'' lies in the range $P + 1 \leqslant j'' \leqslant 2P$. From the

above remarks on the structure of $[\hat{s}^{(i-2)}]$ we have

$$\begin{aligned}
N^{(i-2)}(j'') &= \sum_{\ell=0}^{j''-1} \hat{s}_\ell^{(i-2)} \\
&= \sum_{\ell=0}^{P-1} \hat{s}_\ell^{(i-2)} + \sum_{\ell=P}^{j''-1} \hat{s}_\ell^{(i-2)} \\
&= N^{(i-2)}(P) + \sum_{\ell=P}^{j''-1} \sum_{m=0}^{\ell-1} s_m'^{(i-2)} \\
&= N^{(i-2)}(P) + \sum_{\ell=P}^{j''-1} \left(\mathrm{wt}\left([s'^{(i-2)}]\right) + \sum_{m=P}^{\ell-1} s_{m-P}'^{(i-2)} \right) \\
&= N^{(i-2)}(P) + (j''-P)\mathrm{wt}\left([s'^{(i-2)}]\right) + \sum_{\ell=P}^{j''-1} \hat{s}_{\ell-P}^{(i-2)} \\
&= N^{(i-2)}(P) + (j''-P)(c-1) + N^{(i-2)}(j''-P).
\end{aligned}$$

Similarly, for $t = 2$ we have $2P + 1 \leqslant j'' \leqslant 3P$, and

$$\begin{aligned}
N^{(i-2)}(j'') &= \sum_{\ell=0}^{j''-1} \hat{s}_\ell^{(i-2)} \\
&= \sum_{\ell=0}^{P-1} \hat{s}_\ell^{(i-2)} + \sum_{\ell=P}^{2P-1} \hat{s}_\ell^{(i-2)} + \sum_{\ell=2P}^{j''-1} \hat{s}_\ell^{(i-2)} \\
&= N^{(i-2)}(P) + \left(P\mathrm{wt}\left([s'^{(i-2)}]\right) + N^{(i-2)}(P) \right) \\
&\qquad + \left(2(j''-2P)\mathrm{wt}\left([s'^{(i-2)}]\right) + N^{(i-2)}(j''-2P) \right) \\
&= 2N^{(i-2)}(P) + P(c-1) \\
&\qquad + \left(2(j''-2P)(c-1) + N^{(i-2)}(j''-2P) \right).
\end{aligned}$$

By the same methods, for any t in the range $0 \leqslant t \leqslant c-1$ and j'' in the range $tP + 1 \leqslant j'' \leqslant (t+1)P$ we have

$$\begin{aligned}
N^{(i-2)}(j'') &= \sum_{\ell=0}^{j''-1} \hat{s}_\ell^{(i-2)} \\
&= \sum_{\ell=0}^{P-1} \hat{s}_\ell^{(i-2)} + \sum_{\ell=P}^{2P-1} \hat{s}_\ell^{(i-2)} + \cdots + \sum_{\ell=(t-1)P}^{tP-1} \hat{s}_\ell^{(i-2)} + \sum_{\ell=tP}^{j''-1} \hat{s}_\ell^{(i-2)}
\end{aligned}$$

$$
\begin{aligned}
&= N^{(i-2)}(P) + \left(P\mathrm{wt}\left([s'^{(i-2)}]\right) + N^{(i-2)}(P)\right) \\
&\quad + \left(2P\mathrm{wt}\left([s'^{(i-2)}]\right) + N^{(i-2)}(P)\right) + \cdots \\
&\quad + \left((t-1)P\mathrm{wt}\left([s'^{(i-2)}]\right) + N^{(i-2)}(P)\right) \\
&\quad + \left(t(j''-tP)\mathrm{wt}\left([s'^{(i-2)}]\right) + N^{(i-2)}(j''-tP)\right) \\
&= tN^{(i-2)}(P) + \left(\frac{t(t-1)}{2}P + t(j''-tP)\right)(c-1) \\
&\quad + N^{(i-2)}(j''-tP).
\end{aligned} \tag{2.8}
$$

We may substitute

$$t = \left\lfloor \frac{j''-1}{P} \right\rfloor$$

into equation (2.8) to obtain a general expression for $N^{(i-2)}(j'')$. This expression requires the value of $N^{(i-2)}(P)$ and the value of $N^{(i-2)}(j''')$ for some j''' in the range $0 \leqslant j''' \leqslant P$. This completes the final stage of the proof. □

LEMMA 2.16. *For $i \geqslant 2$ the value of $w^{(i)}$ is given by*

$$w^{(i)} = (c-1)\left[N^{(i-2)}(c^{r+i-2}-1) + (c-1)\left(1 + \frac{(c-2)(c^{r+i-2}-1)}{2}\right)\right].$$

PROOF. We have

$$
\begin{aligned}
w^{(i)} &= \hat{s}_p^{(i-1)} \\
&= s_{k^{(i-1)}}^{(i)} + (c-1)^2 \\
&= \hat{s}_{k^{(i-1)}}^{(i-1)} + (c-1)^2.
\end{aligned} \tag{2.9}
$$

where the first line follows by definition, the second by substituting $p = (c-1)(c^{r+i-1}-1) + k^{(i-1)}$ and equation (2.3) and the third line follows from equation (2.7) and the fact that $k^{(i-1)} = (c-1)(c^{r+i-2}-1) < c^{r+i-1}$. Since $[\hat{s}^{(i-1)}] = D_0^{-1}[s'^{(i-1)}]$, and since the formation of $[s'^{(i-1)}]$ from $[s^{(i-1)}]$ and the formation of $[s^{(i-1)}]$ from $[\hat{s}^{(i-2)}]$ involves changing only terms whose position

lies after $k^{(i-1)} - 1$, we have

$$\begin{aligned}
\hat{s}^{(i-1)}_{k^{(i-1)}} &= \sum_{\ell=0}^{k^{(i-1)}-1} s'^{(i-1)}_{\ell} \\
&= \sum_{\ell=0}^{k^{(i-1)}-1} s^{(i-1)}_{\ell} \\
&= \sum_{\ell=0}^{k^{(i-1)}-1} \hat{s}^{(i-2)}_{\ell} \\
&= N^{(i-2)}(k^{(i-1)}).
\end{aligned}$$

Applying equation (2.8) to this with $t = \left\lfloor k^{(i-1)} - 1/(c^{r+i-2} - 1) \right\rfloor = c - 2$ and $P = c^{r+i-2} - 1$, and then substituting $k^{(i-1)} = (c-1)(c^{r+i-2} - 1) = (c-1)P$ from equation (2.6) we obtain

$$\begin{aligned}
\hat{s}^{(i-1)}_{k^{(i-1)}} &= N^{(i-2)}(k^{(i-1)}) \\
&= (c-2)N^{(i-2)}(P) + \left(\frac{(c-2)(c-3)}{2} P + (c-2)(k^{(i-1)} - (c-2)P) \right) \\
&\quad \times (c-1) + N^{(i-2)}(k^{(i-1)} - (c-2)P) \\
&= (c-2)N^{(i-2)}(P) + \left(\frac{(c-2)(c-3)}{2} P + (c-2)P \right)(c-1) \\
&\quad + N^{(i-2)}(P) \\
&= (c-1)\left[N^{(i-2)}(P) + P\left(\frac{(c-2)(c-3)}{2} + c - 2 \right) \right] \\
&= (c-1)\left[N^{(i-2)}(P) + P\frac{(c-1)(c-2)}{2} \right]
\end{aligned}$$

Substituting into equation (2.9) and rearranging gives the required result. □

We may now give our main result of this Chapter.

THEOREM 2.17. *Let n and c be positive integers, with $c \geqslant 2$ and $n \geqslant 3$. Then there exists a de Bruijn sequence of span n over $\mathbb{Z}_c$, whose decoding procedure has a computational complexity of $O(n^2)$ integer operations, and which requires $O(c^{n-1} \log c)$ bits of storage.*

PROOF. Let r be any positive integer, with $n - r \geqslant 2$. Let $[s^{(0)}]$ be any de Bruijn sequence of span r over $\mathbb{Z}_c$, and apply the GL Construction $n - r$ times, as discussed previously, to obtain a de Bruijn sequence $[s^{(n-r)}]$ of span n.

Decoding for this sequence may be achieved using Algorithm 2.12, as discussed in Section 2.3. By employing the techniques described in Section 2.4, the storage requirements for decoding the sequence $[s^{(n-r)}]$ are:

(1) the values $L^{(1)}(\cdot)$, consisting of $c^r - 1$ c-ary values;

(2) one c-ary value $N^{(i)}(j)$ for each i in the range $0 \leqslant i \leqslant n - r - 2$ and each j in the range $1 \leqslant j \leqslant c^{r+i} - 1$, and

(3) the table T, requiring c^r c^r-ary digits.

By Lemma 2.16, the values $w^{(i)}$ may be easily computed from the values $N^{(i)}(j)$, and need not be stored. Similarly, we have seen how Lemma 2.14 gives rise to an expression allowing us to avoid storing the $k^{(i)}$. Thus the total requirement is for $\Lambda(n, c, r)$ bits, where

$$\begin{aligned}
\Lambda(n,c,r) &= (c^r - 1)\log c + \sum_{i=0}^{n-r-2} (c^{r+i} - 1)\log c + c^r \log(c^r) \\
&= (c^r - 1)\log c + c^r \log c \cdot \sum_{i=0}^{n-r-2} (c^i) - (n - r - 1)\log c + rc^r \log c \\
&= \log c \left[c^r \left(1 + r + \frac{c^{n-r-1} - 1}{c - 1} \right) - n + r \right] \qquad (2.10) \\
&= O(\log c\ (c^{n-1} + rc^r)). \qquad (2.11)
\end{aligned}$$

By letting $r = 2$, which the GL construction allows for any alphabet size c, we obtain the storage requirement described in the statement of the Theorem.

The version of Algorithm 2.12 described in Section 2.3 had a computational complexity of $O(n(n-r))$ integer operations. The cost of employing the storage-

reduction techniques given in this Section is the extra computation required to perform the necessary calculations. However, note that the method described in the proof of Lemma 2.15 for computing $L^{(i)}(j)$, whilst being complicated, involves no looping procedures. The same holds for the computation of the values $k^{(i)}$ and $w^{(i)}$. Thus the number of extra operations performed in computing $L^{(i)}(j)$ is bounded above by a constant. Since Algorithm 2.12 makes just one reference to the tables $L^{(i)}(j)$ for each recursion, the total extra computation involved is $O(n)$ integer operations. Thus the asymptotic complexity of the decoding algorithm remains unaffected by these changes. Letting $r = 2$ gives the $O(n^2)$ complexity stated in the Theorem. □

Comparing equation (2.11) with the original storage requirement of $\Gamma(n, c, r) = O(\log c\,(c^n + rc^r))$ bits given in equation (2.5), the techniques developed in this Section have reduced the previous storage requirement for Algorithm 2.12 by a factor of c (approximately, assuming r is small). This improvement is valid for all values of c, including the case $c = 2$ studied in [51].

3 Decodable de Bruijn Sequences II

3.1 Introduction

We shall present a method for constructing de Bruijn sequences, based on a combination of two existing techniques. The first is the GL construction presented in Chapter 2. The second technique, due to Mitchell, Etzion and Paterson ([42]), we shall call the interleaving construction; details are given later in this Section.

The GL construction has the advantage of constructing sequences over arbitrary alphabets and of arbitrary span, which have a decoding algorithm of low computational complexity. It has the disadvantage that its decoding algorithm requires the storage of relatively large tables of information. The interleaving method also allows the construction of de Bruijn sequences over arbitrary alphabets, with the advantage of allowing a decoding algorithm of low computational complexity that requires only very small look-up tables. However, it is applicable only for a small subset of the possible values of the span n.

We shall combine these two methods, and so obtain a compromise construction. This allows for the construction of de Bruijn sequences of arbitrary span over arbitrary alphabets. The decoding procedure is faster than that for sequences constructed using the GL construction alone, and has similar storage requirements. This approach thus has significant advantages over either the GL or interleaving constructions used in isolation.

Finally, we consider a restriction of the above technique to binary alphabets, the most important case for practical applications. The two underlying

construction methods are shown to exhibit a useful relationship to each other, which is then exploited to dramatically reduce the amount of storage required for decoding these sequences. The result is a construction for binary de Bruijn sequences of arbitrary span n that admits a computationally efficient decoding algorithm that requires storage tables of only a small, constant size.

The remainder of this Section establishes some notation, and presents the interleaving construction of [42]. Notice that the notation and methods of Chapter 2 shall also be employed in this Chapter.

Given two sequences $[b] = (b_i)_{i\in\mathbb{Z}}$ and $[a] = (a_i)_{i\in\mathbb{Z}}$, we define the *interleaving* of $[b]$ and $[a]$, denoted $\mathcal{I}([b],[a])$, to be the sequence $[s] = (s_i)_{i\in\mathbb{Z}}$ given by

$$s_i = \begin{cases} b_{i/2} & \text{if } i \text{ is even, or} \\ a_{(i-1)/2} & \text{if } i \text{ is odd.} \end{cases}$$

We now describe the interleaving construction of [42]. Let $[t]$ be a span n de Bruijn sequence over $\mathbb{Z}_c$, where n and c are any integers with $n, c \geqslant 2$. The construction method uses $[t]$ to form another sequence $[s]$ which is a span $2n$ de Bruijn sequence, also over $\mathbb{Z}_c$. There are two separate constructions, according as to whether c is even or odd. We present the former, slightly more complex case, as this is the case we shall consider in more detail when considering binary alphabets later. The latter case is similar ([42, Parts IIA and IIB]). Recall that X_r denotes the r-tuple $(1, 0, 1, 0 \ldots)$, for all $r \geqslant 1$.

CONSTRUCTION 3.1 ([42, Section III]). Let n and c be integers, with $n, c \geqslant 2$. Suppose that c is even and that $[t]$ is a span n de Bruijn sequence over $\mathbb{Z}_c$, in which $0_{|n|}$ occurs at position 0 and $1_{|n|}$ occurs at position k. We form the sequence $[b]$ of least period $c^n + 2$ by inserting an extra '1' at position k of the

minimal generating cycle of $[t]$ and an extra '0' at position 0. We also define $[a]$ of period $c^n - 2$ by 'doubly puncturing' $[t]$, that is by removing a '1' from position k and a '0' from position 0 of the minimal generating cycle of $[t]$. Next, we form the sequence $[d]$ of least period $(c^n - 2)(c^n + 2) = c^{2n} - 4$ by letting

$$[d] = \mathcal{I}([b], [a]).$$

We note that the $(2n-1)$-tuples X_{2n-1} and $1_{|2n-1|}$ must each occur exactly once in $[d]$, and denote their respective positions by $v = \mathrm{loc}([d], X_{2n-1})$ and $m = \mathrm{loc}([d], 1_{|2n-1|})$. Finally, we let $[s]$ be the sequence whose minimal generating cycle is formed by (simultaneously) inserting an extra '10' into the minimal generating cycle of $[d]$ at position v, inserting a '1' at position m and a '0' at position 0. □

We write $[s] = \Upsilon[t]$ when $[s]$ is a span $2n$ de Bruijn sequence formed using interleaving from a span n de Bruijn sequence $[t]$, by the above method when the alphabet size c is even and by the related method when c is odd. We illustrate the interleaving construction with an example.

EXAMPLE 3.2. Let $[t] = [00011101]$, a binary de Bruijn sequence of span 3, so $[a] = [001101]$ and $[b] = [0000111101]$. Interleaving $[b]$ and $[a]$, and writing the digits of the former in bold, we get

$$\begin{aligned}[d] \;=\; [&\underline{\mathbf{00000}}\mathbf{1}\mathbf{0}\mathbf{1}\mathbf{1}\mathbf{0}\mathbf{1}\mathbf{1}\mathbf{1}\mathbf{0}\mathbf{1}\mathbf{0}\mathbf{0}\mathbf{1}\mathbf{1}\mathbf{1}\mathbf{0}\mathbf{0}\mathbf{0}\mathbf{1}\mathbf{0}\mathbf{0}\mathbf{0}\mathbf{0}\underline{\mathbf{11111}}\\ &\mathbf{0}\mathbf{1}\mathbf{1}\mathbf{0}\mathbf{0}\mathbf{1}\mathbf{0}\mathbf{0}\mathbf{1}\mathbf{0}\mathbf{1}\mathbf{0}\mathbf{0}\mathbf{0}\mathbf{1}\underline{\mathbf{10101}}\mathbf{1}\mathbf{1}\mathbf{1}\mathbf{0}\mathbf{0}\mathbf{1}\mathbf{1}].\end{aligned}$$

Notice that X_5 appears at position 48, $1_{|5|}$ appears at position 28 and that $0_{|5|}$ appears at position 0. These 5-tuples have been underlined. Simultaneously

inserting the digits '10', '1' and '0' at these positions, respectively, gives

$$[s] = [00000010110111010011100010000111$$
$$11101100100101000110101011110011],$$

a de Bruijn sequence of span 6. □

RESULT 3.3 (from [42, Sections IIE and IIIE]). *Let $[t]$ be a span n de Bruijn sequence, and let $[s] = \Upsilon[t]$. Then there exists a decoding procedure for $[s]$ that requires:*

(1) at most two decodings of $[t]$;

(2) the solution modulo $c^{2n} - \epsilon$ of a pair of simultaneous congruences, given modulo $c^n - \epsilon$ and $c^n + \epsilon$ (where ϵ represents some small, variable integer), and

(3) a small, constant amount of computation and storage.

Suppose that a de Bruijn sequence of span n is required, where $n = a2^b$ where a is odd. Given a starting sequence $[t]$ of span a, applying the interleaving construction b times produces a de Bruijn sequence of span n, whose decoding problem may be reduced to 2^b decodings of the sequence $[t]$ using Result 3.3. Thus when used in isolation the interleaving technique is effective only when a is small. In particular, it is of no use when the required span n is odd.

3.2 CONSTRUCTION

In this section we give a construction for de Bruijn sequences of arbitrary span n and arbitrary alphabet size c, that combines the GL construction and the interleaving construction.

The idea behind the Construction may be summarised as follows. The interleaving method gives a span $2n$ sequence from a span n sequence. Considering the binary representation of the span (with the most significant digit to the left), applying this method gives a de Bruijn sequence whose span has a binary representation equal to that of the previous sequence, except that it has been shifted left by one place and the digit zero inserted at the right hand end. By contrast, the Lempel method takes a span n de Bruijn sequence and gives a span $n+1$ sequence. If this is applied after interleaving, the newly inserted zero at the right hand end of the binary representation of n is changed into a one. By applying the interleaving method repeatedly, and alternating with the Lempel method where necessary, any binary sequence may be obtained as the binary representation of n, and so any span n may be achieved. The method is presented formally below. (The reason why any span n may be obtained in this way is in fact the same as the reason why the well known 'square and multiply' algorithm for exponentiation may be used for any exponent.)

CONSTRUCTION 3.4. Suppose that we wish to construct a c-ary de Bruijn sequence of span n over $\mathbb{Z}_c$, where $n, c \geqslant 2$. Firstly, let $[s^{(2)}] = (s_i^{(2)})_{i\in\mathbb{Z}}$ be a span 2 de Bruijn sequence over $\mathbb{Z}_c$ with $[s^{(2)}]_0^2 = 0_{|2|}$.

Let $d = \lfloor 1 + \log_2 n \rfloor$ be the number of digits in the binary representation of the desired span n, and denote this representation by $n_1 n_2 \cdots n_d$. (We position the most significant digit to the left, so $n = \sum_{i=1}^{d} n_i 2^{d-i}$.) Necessarily, $n_1 = 1$ and $d \geqslant 2$.

Inductively define the sequences $[s^{(i)}] = (s_i^{(i)})_{i\in\mathbb{Z}}$ for i in the range $3 \leqslant i \leqslant d$

and $[t^{(i)}] = (t_i^{(i)})_{i\in\mathbb{Z}}$ for $2 \leqslant i \leqslant d$ by

$$[s^{(i)}] = \Upsilon[t^{(i-1)}] \quad \text{and} \tag{3.1}$$

$$[t^{(i)}] = \begin{cases} [s^{(i)}] & \text{if } n_i = 0, \text{ or} \\ \Phi[s^{(i)}] & \text{if } n_i = 1. \end{cases} \tag{3.2}$$

□

Note that the necessary conditions for the application of the GL and interleaving constructions are always satisfied, since they are only applied to sequences of span at least 2.

LEMMA 3.5. *For each integer j in the range $1 \leqslant j \leqslant d$, let m_j denote the integer whose binary representation is given by $n_1 n_2 \cdots n_j$, so $m_j = \sum_{i=1}^{j} n_i 2^{j-i}$. Then for each i in the range $2 \leqslant i \leqslant d$ the sequence $[t^{(i)}]$ is a c-ary de Bruijn sequence of span m_i.*

PROOF. From the definition of $[t^{(2)}]$, the Lemma holds when $i = 2$. We shall prove the result by induction upon i, thus we assume that the Lemma holds for some value $i = r$ and consider the case $i = r + 1$.

It is clear that since $[t^{(r)}]$ is a de Bruijn sequence of span m_r, and since $[s^{(r+1)}] = \Upsilon[t^{(r)}]$, we have that $[s^{(r+1)}]$ is a de Bruijn sequence with span $2m_r$. If $n_r = 0$ we have $[t^{(r+1)}] = [s^{(r+1)}]$ and $2m_r = m_{r+1}$, as required. Otherwise, if $n_r = 1$ then $[t^{(r+1)}] = \Phi[s^{(r+1)}]$, and so $[t^{(r+1)}]$ has span $2m_r + 1$ which is equal to m_{r+1} in this case, as required. □

COROLLARY 3.6. *The sequence $[t^{(d)}]$ is a c-ary de Bruijn sequence of span n.*

3.3 Decoding, Complexity and Storage

We make the following observation on the two underlying constructions used: each is a means of forming a higher span sequence from a lower span one, in such a way that the decoding problem for the higher span sequence may easily be solved assuming it is possible to decode the lower span sequence.

Previously, each construction has been applied iteratively, usually starting with a sequence of such low span that it is trivial to decode. The above property then leads to a recursive decoding algorithm for the final high-span sequence. The crucial observation is to notice that since *both* construction methods share this property, sequences formed by a combination of these methods also allow for recursive decoding, using an appropriate combination of the decoding methods for the separate constructions.

We do not state the algorithm for decoding $[t^{(d)}]$ formally. From the above it is clear how the overall algorithm would work, given procedures for decoding sequences formed by a single application of either of the underlying constructions. Instead, we give a Theorem presenting the storage requirements and complexity of the proposed procedure.

Theorem 3.7. *Let n and c be arbitrary integers, with $n, c \geqslant 2$, and let $d = \lfloor 1 + \log_2 n \rfloor$. Let $[t^{(d)}]$ be the c-ary de Bruijn sequence of span n arising from Construction 3.4. Then the decoding procedure for $[t^{(d)}]$ described above has a computational complexity of $O(n \log n(1+\log c))$ integer operations, and requires $O((c^{n/2^\ell} + n) \log c)$ bits of storage, where ℓ is the number of factors of 2 in n.*

Proof. We adopt the notation of Construction 3.4 and Lemma 3.5. We shall

need to decode each sequence $[t^{(i)}]$ and each sequence $[s^{(i)}]$, for each i in the range $2 \leqslant i \leqslant d$. (The sequence $[t^{(1)}]$ is trivial to decode if it is chosen as suggested in Construction 3.4). We assumed $n \geqslant 2$ and so $d \geqslant 2$.

Suppose that we wish to decode the sequence $[t^{(i)}]$. If $n_i = 0$ this is simply a decoding of $[s^{(i)}]$. Otherwise, if $n_i = 1$ the decoding may be performed by a version of Algorithm 2.12, with the recursive step modified to call the decoding procedure for the previous sequence $[s^{(i)}]$ rather than to call Algorithm 2.12 again. Since by Lemma 3.5 the span of $[t^{(i)}]$ is m_i, decoding $[t^{(i)}]$ in this way requires one decoding of $[s^{(i)}]$ together with $O(m_i)$ integer operations (there being a constant number of operations for all stages of a single recursion of Algorithm 2.12 except for the computation of Dw given the input m_i-tuple w, which requires $O(m_i)$ operations).

Each sequence $[s^{(i)}]$ is formed using the interleaving construction, and by Result 3.3 its decoding requires two decodings of $[t^{(i-1)}]$. Thus, to decode $[t^{(d)}]$ requires one decoding of $[s^{(d)}]$, two decodings of $[s^{(d-1)}]$, four decodings of $[s^{(d-2)}]$, and so forth, up to 2^{d-2} decodings of $[s^{(2)}]$. In total, P decodings of interleaved sequences are required, where

$$\begin{aligned} P &= \sum_{i=2}^{d} 2^{d-i} \\ &= 2^{d-1} - 1 \\ &= O(n). \end{aligned}$$

By Result 3.3, each of these $O(n)$ decodings also requires the solution of a pair of congruences (using the Chinese Remainder Theorem) and a small, constant amount of computation. The latter results in a total of $O(n)$ integer operations;

the former we consider in more detail later.

The number of applications of a single recursion of Algorithm 2.12 for recursively decoding $[t^{(d)}]$ depends on the number of values i for which $n_i = 1$. In the worst case, $n_i = 1$ for all i, so $n = 2^d - 1$, and every decoding of a sequence $[t^{(i)}]$ requires the application of the Algorithm. To recursively decode $[t^{(d)}]$, the required number of decodings of each sequence $[t^{(i)}]$ for i in the range $2 \leqslant i \leqslant d$ is the same as the number of decodings of the sequence $[s^{(i)}]$ above, namely 2^{d-i}. However, the complexity of each decoding is dependent on the window size, which in turn varies with the depth of recursion. As seen above, decoding $[t^{(i)}]$ requires $O(m_i)$ operations. Let Q denote the total number of operations involved in the application of single recursion of Algorithm 2.12 in order to decode $[t^{(d)}]$. Then Q is given by

$$\begin{aligned} Q &= O\left(\sum_{i=2}^{d} 2^{d-i} m_i\right) \\ &= O\left(\sum_{i=2}^{d} 2^{d-i}(2^i - 1)\right) \\ &= O\left(\sum_{i=2}^{d} 2^d\right) \\ &= O\left(n \log n\right). \end{aligned}$$

We have shown that decoding $[t^{(d)}]$ requires $O\left(n \log n\right)$ integer operations, together with the solution of $O(n)$ pairs of simultaneous congruences. The size of the moduli involved in each of the latter pairs varies. Since the span of $[t^{(i-1)}]$ is at most $2^i - 1$, decoding $[s^{(i)}]$ results in 2^{d+1-i} pairs of simultaneous congruences, each of which has its least modulus of size at most $c^{2^i - 1}$. By applying the well known result that a pair of simultaneous congruences whose least modulus has size N can be solved using Euclid's Algorithm in $O(N)$ integer operations, the

number of integer operations required to solve the above congruences is given by R, where

$$\begin{aligned} R &= O\left(\sum_{i=2}^{d} 2^{d+1-i} \log c^{2^i}\right) \\ &= O\left(\sum_{i=2}^{d} 2^{d+1} \log c\right) \\ &= O(n \log n \log c). \end{aligned}$$

Thus the total computation requirement for decoding the c-ary de Bruijn sequences of span n arising from Construction 3.4 is

$$O(n \log n(1 + \log c)) \tag{3.3}$$

integer operations.

It remains to consider the storage requirements of the algorithm. The storage required for each decoding of a sequence formed using the interleaving construction is described in [42, Section IIIE]. This states that, with the notation of Construction 3.1, it is necessary to store three integers, corresponding to the position of the digits inserted into $[d]$ to form $[t]$. By always having the all-zero n-tuple at position 0 in all our sequences we have already removed the need to store one of these values. The other two values, denoted v and m in Construction 3.1, must be stored for each of the $O(d)$ applications of the Construction used to form $[t^{(d)}]$. These values require $O(n \log c)$ bits of storage.

The application of a single recursion of Algorithm 2.12 to decode $[t^{(i)}]$ requires the storage of certain tables of information, namely

(1) c-ary digits $L^{(i)}(j) = t_j^{(i)}$ for each j in the range $0 \leqslant j \leqslant c^{m_i - 1} - 2$, and

(2) a c^{m_i}-ary digit $k^{(i)} = \mathrm{loc}([t^{(i)}], 1_{|m_i|})$.

These correspond to items (1) and (2) on page 52. Item (3) is unnecessary, since the GL construction is only applied to sequences arising from the interleaving construction, and never from sequences whose decoding is achieved directly from a table. However, these values need be stored only for the values of i in the range $2 \leqslant i \leqslant d$ for which $n_i = 1$. Notice that the techniques developed in Section 2.4 to reduce the space required to store the values $L^{(i)}(\cdot)$ are not applicable in this case, since the sequences $[t^{(i)}]$ are formed by a combination of methods rather than by the generalised Lempel method alone. These storage requirements are dominated by the largest table $L^{(i)}(\cdot)$, which corresponds to the largest value i for which $n_i = 1$. Denoting this value of i by ℓ', we require storage for $O(c^{m_{\ell'}}) = O(c^{n/2^{\ell}})$ c-ary digits, where $\ell = d - \ell'$ is the number of factors of 2 in n. The result follows. □

For large n, this represents a significant reduction in complexity when compared with the $O(n^2)$ operations required to decode sequences formed using the GL construction of Chapter 2 alone.

For odd n, where $\ell = 0$, this storage requirement is asymptotically equal to that required for decoding sequences constructed with the generalised Lempel method of Chapter 2 (the comparable case having $r = 1$ in equation (2.5)). The storage efficiency is markedly improved when n is even, and further improved with each extra factor of 2 in n. In these cases the sequence $[t^{(d)}]$ is effectively formed by repeatedly applying the interleaving method to an initial sequence of span $n' = n/2^{\ell}$ formed using the construction of this Chapter. The benefits of having these extra factors of 2 in n are due to the well documented properties of the interleaving construction, rather than any new technique in this Chapter.

3.4 Preliminary Lemmas

For the remainder of this Chapter, we consider only the binary alphabet. It is our intention to demonstrate how, given some stored tables of small, constant size, the storage requirement described in Theorem 3.7 may be replaced by some calculations of low computational complexity—this result will be presented in Section 3.5. In this Section we give a number of preliminary results that will aid us in achieving this goal.

Given any sequence $[r] = (r_i)_{i \in \mathbb{Z}}$ we define the sums $\sigma_j[r] \in \mathbb{Z}_2$ for all $j \geqslant 1$ by

$$\sigma_j[r] = \sum_{i=0}^{j-1} r_i.$$

We also define the double sums $\eta_j[r] \in \mathbb{Z}_2$ for all $j \geqslant 2$ by

$$\eta_j[r] = \sum_{i=1}^{j-1} \sigma_i[r].$$

We will find sums of alternate elements from sequences useful, and so define $\xi_e([r], j) \in \mathbb{Z}_2$ and $\xi_o([r], j) \in \mathbb{Z}_2$ for all $j \geqslant 1$ by

$$\begin{aligned} \xi_e([r], j) &= \sum_{i=0}^{j-1} r_{2i} \\ \xi_o([r], j) &= \sum_{i=0}^{j-1} r_{2i+1}. \end{aligned}$$

The subscripts 'e' and 'o' refer to the summations being over the terms whose indices are even or odd, respectively. The role of j in the expressions $\sigma_j[r]$, $\xi_e([r], j)$ and $\xi_o([r], j)$ is to denote the number of terms in each of these summations, thus we define $\sigma_0[r] = \xi_e([r], 0) = \xi_o([r], 0) = 0$. Notice that any expressions for each of the sums σ_j, η_j, ξ_e and ξ_o described above should be taken modulo 2. The following Remark is of central importance in what shall follow.

REMARK 3.8. For any sequence $[r] = (r_i)_{i\in\mathbb{Z}}$ and any $j \geqslant 2$ we have that, since our alphabet is binary,

$$\begin{aligned}
\eta_j[r] &= \sum_{i=1}^{j-1} \sigma_i[r] \\
&= \sum_{i=1}^{j-1}\sum_{\ell=0}^{i-1} r_\ell \\
&= \sum_{i=0}^{j-2} (j-1-i) r_i \\
&= \begin{cases} \xi_e([r], j/2) & \text{for } j \text{ even, or} \\ \xi_o([r], (j-1)/2) & \text{for } j \text{ odd.} \end{cases}
\end{aligned}$$

□

We now give an outline of the strategy we shall adopt for eliminating the storage requirement of Theorem 3.7. Imagine for the moment that the interleaving construction consists purely of interleaving a de Bruijn sequence with a copy of itself, and ignore the extra stages involving the insertion and deletion of other digits. Similarly, imagine that the GL construction simply involves applying the D_0^{-1} operator (that is, forming a new sequence by taking partial sums of terms from the previous sequence), again ignoring the stages involving the insertion and removal of other digits.

Now suppose for each sequence $[s^{(i)}]$ arising in Construction 3.4 that the sums of terms and double sums of terms, as represented by the notation σ and η established above, are known. For each i, the sequence $[t^{(i)}]$ is formed either by copying $[s^{(i)}]$ directly, or by applying the GL construction. In the former case, clearly the terms and sums of terms of $[t^{(i)}]$ are known. In the latter case, the terms of $[t^{(i)}]$ are also known, since they are formed by summing terms from $[s^{(i)}]$, and the sums of terms of $[t^{(i)}]$ correspond to the double sums of terms

of $[s^{(i)}]$, which are also known. Thus in each case the terms and sums of terms in $[t^{(i)}]$ are known.

Now consider applying the interleaving construction to $[t^{(i)}]$, to yield $[s^{(i+1)}]$. By the nature of interleaving, a sum of terms in $[s^{(i+1)}]$ can be obtained from two expressions in sums of terms in $[t^{(i)}]$. The key to our method is to observe that the double sums of terms in $[s^{(i+1)}]$ may also be found, since by Remark 3.8 these correspond to sums of alternate terms in $[s^{(i+1)}]$, which are easy to find from sums of terms in $[t^{(i)}]$ since $[s^{(i+1)}]$ is formed by interleaving.

The main storage requirement described in the proof of Theorem 3.7 is for the tables $L^{(i)}(j)$, containing the terms $t_j^{(i)}$ of the sequences $[t^{(i)}]$ formed at the GL stage of the construction. The above argument outlines a scheme by which these terms could be recursively computed rather than stored. However, in the above we have greatly simplified the two constructions used, by ignoring the stages in the GL and interleaving constructions in which terms are inserted and removed from the sequences involved. It is necessary to form the sums and double sums above using the unsimplified forms of the two constructions. This is the purpose of Lemmas 3.11, 3.12, 3.14 and 3.15 below.

Before we begin those Lemmas, we prove two other Lemmas concerning the positions of certain windows in sequences arising from the interleaving construction. Expressions for these positions are required in some of the later Lemmas. Also, some of the other stored information in the proof of Theorem 3.7 concerned the positions of certain windows in the sequences arising during Construction 3.4; the following results shall enable us later to avoid this storage requirement as well.

LEMMA 3.9. *Let* $[s]$, $[t]$, $[a]$, $[b]$, $[d]$, k, v, n *and* m *be as in Construction 3.1. Assuming* k *is odd and that* $n \geqslant 3$, *then*

$$\begin{aligned} m &= 2k - 2 + 2^{n-1}(2^n - 2) && (3.4)\\ &= 2k + 2 + (2^{n-1} - 2)(2^n + 2) && (3.5)\\ v &= \begin{cases} (2^n + 1 - k)(2^{n-1} - 1) & \text{if } k \equiv 1 \pmod 4, \text{ or} \\ (2^{n+1} + 3 - k)(2^{n-1} - 1) & \text{if } k \equiv 3 \pmod 4. \end{cases} && (3.6) \end{aligned}$$

PROOF. Notice that $[a]_{k-1}^{n-1} = 1_{|n-1|}$ and $[b]_{k+1}^{n+1} = 1_{|n+1|}$, and that the unique position m of $1_{|2n-1|}$ in $[d]$ occurs when these windows of $[b]$ and $[a]$ are interleaved (uniqueness is assured by the proof of Construction 3.1). Since k is odd, $k+1$ and $k-1$ are both even, and we observe that digits with an even position in $[b]$ are always followed in $[d]$ by digits with an even position in $[a]$, thus the first digit of the window $1_{|2n-1|}$ in $[d]$ is obtained from the first digit of an instance of $1_{|n+1|}$ in $[b]$. Noting that $[a]$ and $[b]$ have least periods $2^n - 2$ and $2^n + 2$ respectively, we deduce that m is the unique solution modulo $2^{2n} - 4$ of the simultaneous congruences

$$\begin{aligned} \frac{m}{2} &\equiv k + 1 \pmod{2^n + 2} \quad \text{and} \\ \frac{m}{2} &\equiv k - 1 \pmod{2^n - 2}. \end{aligned}$$

It is now simple to verify that the two equivalent formulae (3.4) and (3.5) are correct.

To determine v, we argue as follows. Notice first that $[a]_{i(2^n-2)}^{n-1} = 0_{|n-1|}$ for all $i \in \mathbb{Z}$. We consider where these $(n-1)$-windows occur in $[d]$, and with which n-windows of $[b]$ they are interleaved. (Digits of $[b]$ are to be taken as the first

and last digits of the resulting interleaved $(2n-1)$-tuple.) Now:

$$[a]_{0}^{n-1} \text{ is interleaved with } [b]_{0}^{n};$$

$$[a]_{2^n-2}^{n-1} \text{ is interleaved with } [b]_{2^n-2}^{n} = [b]_{-4+2^n+2}^{n};$$

$$[a]_{2(2^n-2)}^{n-1} \text{ is interleaved with } [b]_{2(2^n-2)}^{n} = [b]_{-4\times 2+2^n+2}^{n};$$

$$\vdots$$

$$[a]_{2^{n-2}(2^n-2)}^{n-1} \text{ is interleaved with } [b]_{2^{n-2}(2^n-2)}^{n} = [b]_{-4(2^{n-2})+2^n+2}^{n};$$

$$[a]_{(2^{n-2}+1)(2^n-2)}^{n-1} \text{ is interleaved with } [b]_{(2^{n-2}+1)(2^n-2)}^{n} = [b]_{-2+2^n+2}^{n};$$

$$[a]_{(2^{n-2}+2)(2^n-2)}^{n-1} \text{ is interleaved with } [b]_{(2^{n-2}+2)(2^n-2)}^{n} = [b]_{-6+2^n+2}^{n};$$

$$\vdots$$

$$[a]_{2^{n-1}(2^n-2)}^{n-1} \text{ is interleaved with } [b]_{2^{n-1}(2^n-2)}^{n} = [b]_{-2^n+2+2^n+2}^{n};$$

$$[a]_{(2^{n-1}+1)(2^n-2)}^{n-1} \text{ is interleaved with } [b]_{(2^{n-1}+1)(2^n-2)}^{n} = [b]_{0}^{n},$$

and so the pattern repeats. By a similar argument to that used to find m, since $[b]_{k+1}^{n+1} = 1_{|n+1|}$ for all $i \in \mathbb{Z}$, we are interested in where $[b]_{k+1}^{n}$ appears in the right hand column in the above list. If $k \equiv 1 \pmod 4$ then this will be in the first half of the list, corresponding to $[a]_{\frac{2^n+1-k}{4}(2^n-2)}^{n-1}$. This gives $v = (2^n+1-k)(2^{n-1}-1)$. Similarly, if $k \equiv 3 \pmod 4$ then $[b]_{k+1}^{n}$ occurs in the second half of the list, matched with $[a]_{(\frac{2^n+3-k}{4}+2^{n-2})(2^n-2)}^{n-1}$. This gives $v = (2^{n+1}+3-k)(2^{n-1}-1)$, and so equation (3.6) is proved. □

LEMMA 3.10. *Let $[s]$, k, v, n and m be as in Construction 3.1, and assume again that k is odd and that $n \geqslant 3$. Let $w = \mathrm{loc}([s], 1_{|2n|})$. Then*

(1) if $v < m$ then $w = m+3$ and $\sigma_w[s] = 0$, or

(2) if $v > m$ then $w = m+1$ and $\sigma_w[s] = 1$.

PROOF. We adopt the notation of Lemma 3.9. Consider the process of constructing $[s]$ from $[d]$. If $v < m$, then the digits '01' are inserted into $[d]$ at a position before $1_{|2n-1|}$ appears in $[d]$. Together with the digit '0' being inserted into $[d]$ at position 0, this gives $w = m+3$. That $w = m+1$ when $v > m$ is also clear.

Notice from equation (3.4) that m is even. We have

$$\begin{aligned} \sigma_m[d] &= \xi_e([d], m/2) + \xi_o([d], m/2) \\ &= \sigma_{m/2}[b] + \sigma_{m/2}[a]. \end{aligned} \tag{3.7}$$

Since $[a]$ has least period $2^n - 2$ and weight 1, and from its relation to $[t]$, we have

$$\sigma_\ell[a] = \begin{cases} \sigma_{\ell+1}[t] & \text{if } 1 \leqslant \ell \leqslant k-1; \\ \sigma_{\ell+2}[t] - 1 & \text{if } k \leqslant \ell \leqslant 2^n - 2, \text{ or} \\ \sigma_{\ell \bmod (2^n-2)}[a] + \left\lfloor \frac{\ell}{2^n-2} \right\rfloor & \text{if } \ell \geqslant 2^n - 1. \end{cases} \tag{3.8}$$

Taking m from equation (3.4), and substituting $m/2$ in place of ℓ in (3.8) gives

$$\begin{aligned} \sigma_{m/2}[a] &= \sigma_{k-1}[a] + 2^{n-2} \\ &= \sigma_{k-1}[a] \\ &= \sigma_k[t], \end{aligned} \tag{3.9}$$

using the condition $n \geqslant 3$. Similarly,

$$\sigma_\ell[b] = \begin{cases} \sigma_{\ell-1}[t] & \text{if } 1 \leqslant \ell \leqslant k+1; \\ \sigma_{\ell-2}[t] + 1 & \text{if } k+2 \leqslant \ell \leqslant 2^n + 2, \text{ or} \\ \sigma_{\ell \bmod (2^n+2)}[b] + \left\lfloor \frac{\ell}{2^n+2} \right\rfloor & \text{if } \ell \geqslant 2^n + 3. \end{cases} \tag{3.10}$$

Taking m from equation (3.5) and substituting $m/2$ for ℓ in the above gives

$$\sigma_{m/2}[b] = \sigma_{k+1}[b] + 2^{n-2} - 1$$

$$= \quad \sigma_k[t] + 1. \tag{3.11}$$

Combining equations (3.7), (3.9) and (3.11) gives $\sigma_m[d] = 1$. It is then simple to see that if $v > m$ then $\sigma_w[s] = \sigma_m[d] = 1$ and that if $v < m$ then $\sigma_w[s] = \sigma_m[d] + 1 = 0$, as required. □

The following two Lemmas each concern relations between summations of terms of de Bruijn sequences formed using the interleaving construction. We adopt the notation employed in Lemmas 3.9 and 3.10.

LEMMA 3.11. *For any i in the range $1 \leqslant i \leqslant 2^{2n}$ the value of $\sigma_i[s]$ can be found using k and at most two values $\sigma_j[t]$ and $\sigma_{j'}[t]$, where j and j' lie in the range $1 \leqslant j, j' \leqslant 2^n$.*

PROOF. If $v < m$ we have

$$\sigma_i[s] = \begin{cases} \sigma_{i-1}[d] & \text{if } 1 \leqslant i \leqslant v+1; \\ \sigma_v[d] + 1 & \text{if } i = v+2, v+3; \\ \sigma_{i-3}[d] + 1 & \text{if } v+4 \leqslant i \leqslant m+3; \\ \sigma_m[d] + 2 & \text{if } i = m+4, \text{ or} \\ \sigma_{i-4}[d] + 2 & \text{if } m+5 \leqslant i \leqslant 2^{2n}. \end{cases}$$

Alternatively, if $v > m$ then

$$\sigma_i[s] = \begin{cases} \sigma_{i-1}[d] & \text{if } 1 \leqslant i \leqslant m+1; \\ \sigma_m[d] + 1 & \text{if } i = m+2; \\ \sigma_{i-2}[d] + 1 & \text{if } m+3 \leqslant i \leqslant v+2; \\ \sigma_v[d] + 2 & \text{if } i = v+3, v+4, \text{ or} \\ \sigma_{i-4}[d] + 2 & \text{if } v+5 \leqslant i \leqslant 2^{2n}. \end{cases}$$

(In the above we leave the '+2' to facilitate understanding of the argument; of course, it serves no purpose as the results should be taken modulo 2.)

From the construction of $[d]$, the value $\sigma_\ell[d]$ for all $\ell \geqslant 1$ may be computed from the information described in the statement of the Lemma, by using equations (3.8) and (3.10) together with

$$\sigma_\ell[d] = \sigma_{\lceil \ell/2 \rceil}[b] + \sigma_{\lfloor \ell/2 \rfloor}[a].$$

□

LEMMA 3.12. *Assume that k is odd. For any i in the range $2 \leqslant i \leqslant 2^{2n}$ the value of $\eta_i[s]$ can be found using k and at most one value $\sigma_j[t]$, where j lies in the range $1 \leqslant j \leqslant 2^n$.*

PROOF. Applying Remark 3.8 gives

$$\eta_i[s] \quad = \quad \begin{cases} \xi_e([s], i/2) & \text{for } i \text{ even, or} \\ \xi_o([s], (i-1)/2) & \text{for } i \text{ odd.} \end{cases}$$

Notice from Lemma 3.9 that v and m are both even, and easily computed from k. If $v < m$ then

$$\xi_e([s], \ell) \quad = \quad \begin{cases} \xi_o([d], \ell - 1) & \text{if } 1 \leqslant \ell \leqslant v/2 + 1; \\ \xi_o([d], v/2) & \text{if } \ell = v/2 + 2; \\ \xi_o([d], \ell - 2) & \text{if } v/2 + 3 \leqslant \ell \leqslant m/2 + 2, \text{ or} \\ \xi_o([d], m/2) + \xi_e([d], \ell - 2) & \\ \quad -\xi_e([d], m/2) & \text{if } m/2 + 3 \leqslant \ell \leqslant 2^{2n-1}. \end{cases}$$

Similarly,

$$\xi_o([s],\ell) = \begin{cases} \xi_e([d],\ell) & \text{if } 1 \leqslant \ell \leqslant v/2; \\ \xi_e([d],v/2)+1 & \text{if } \ell = v/2+1; \\ \xi_e([d],\ell-1)+1 & \text{if } v/2+2 \leqslant \ell \leqslant m/2+1; \\ \xi_e([d],\ell-2)+2 & \text{if } \ell = m/2+2, \text{ or} \\ \xi_e([d],m/2)+2+\xi_o([d],\ell-2) & \\ \quad -\xi_o([d],m/2) & \text{if } m/2+3 \leqslant \ell \leqslant 2^{2n-1}. \end{cases}$$

If $v > m$ then by similar methods

$$\xi_e([s],\ell) = \begin{cases} \xi_o([d],\ell-1) & \text{if } 1 \leqslant \ell \leqslant m/2+1; \\ \xi_o([d],m/2)+\xi_e([d],\ell-1) & \\ \quad -\xi_e([d],m/2) & \text{if } m/2+2 \leqslant \ell \leqslant v/2+1; \\ \xi_o([d],m/2)+\xi_e([d],v/2) & \\ \quad +1-\xi_e([d],m/2) & \text{if } \ell = v/2+2, \text{ or} \\ \xi_o([d],m/2)+\xi_e([d],\ell-2) & \\ \quad +1-\xi_e([d],m/2) & \text{if } v/2+3 \leqslant \ell \leqslant 2^{2n-1} \end{cases}$$

and

$$\xi_o([s],\ell) = \begin{cases} \xi_e([d],\ell) & \text{if } 1 \leqslant \ell \leqslant m/2; \\ \xi_e([d],\ell-1)+1 & \text{if } \ell = m/2+1; \\ \xi_e([d],m/2)+1+\xi_o([d],\ell-1) & \\ \quad -\xi_o([d],m/2) & \text{if } m/2+2 \leqslant \ell \leqslant v/2+1; \\ \xi_e([d],m/2)+1+\xi_o([d],\ell-2) & \\ \quad -\xi_o([d],m/2) & \text{if } \ell = v/2+2, \text{ or} \\ \xi_e([d],m/2)+1+\xi_o([d],\ell-2) & \\ \quad -\xi_o([d],m/2) & \text{if } v/2+3 \leqslant \ell \leqslant 2^{2n-1}. \end{cases}$$

Expressions concerning $\xi_e([d], m/2)$ and $\xi_o([d], m/2)$ in each of the above equations can be simplified by substituting

$$\begin{aligned} \xi_o([d], m/2) &= \sigma_{m/2}[a] \\ &= \sigma_k[t] \quad \text{(applying equation (3.9))} \end{aligned}$$

and

$$\begin{aligned} \xi_e([d], m/2) &= \sigma_{m/2}[b] \\ &= \sigma_k[t] + 1 \quad \text{(applying equation (3.11)).} \end{aligned}$$

Since $\sigma_k[t]$ appears either not at all or twice in each of the expressions for $\xi_o([s], \ell)$ and $\xi_e([s], \ell)$ thus resulting, and since we are working in binary, the result is not dependent on $\sigma_k[t]$. In each case we are left with an expression for $\eta_i[s]$ in terms of at most one of $\xi_o([d], \ell)$ or $\xi_e([d], \ell)$, for some ℓ in the range $1 \leqslant \ell \leqslant 2^{2n-1} - 1$. From the definition of $[d]$ we deduce that

$$\xi_o([d], \ell) = \sigma_\ell[a]$$

and

$$\xi_e([d], \ell) = \sigma_\ell[b].$$

By applying either equation (3.8) or (3.10), as appropriate, we obtain $\eta_i[s]$ in terms of k and $\sigma_j[t]$, where j lies in the range $1 \leqslant j \leqslant 2^n$. □

We now prove a more specific result that we later use to eliminate the need to perform a particular calculation.

COROLLARY 3.13. *With the notation of Lemma 3.12, we have that*

$$\eta_{2^{2n}}[s] = \begin{cases} 0 & \text{if } v < m, \text{ or} \\ 1 & \text{if } v > m. \end{cases}$$

PROOF. Let m, k, $[b]$ and $[d]$ be as given in Construction 3.1. Following the proof of Lemma 3.12, for $v < m$ we have

$$\begin{aligned}
\eta_{2^{2n}}[s] &= \xi_e([s], 2^{2n-1}) \\
&= \xi_o([d], m/2) + \xi_e([d], 2^{2n-1} - 2) - \xi_e([d], m/2) \\
&= 1 + \xi_e([d], 2^{2n-1} - 2) \\
&= 1 + \sigma_{2^{2n-1}-2}[b].
\end{aligned}$$

Now, since $2^{2n-1} - 2 = (2^n + 2)(2^{n-1} - 1)$, applying equation (3.10) gives

$$\sigma_{2^{2n-1}-2}[b] = \sigma_0[b] + 2^{n-1} - 1.$$

The result follows for $v < m$, and the case $v > m$ is similar. □

We now give two Lemmas concerning summations of terms in sequences arising from the GL construction. Let n be an integer, $n \geqslant 2$, let $[s] = (s_i)_{i\in\mathbb{Z}}$ be a de Bruijn sequence of span n, let $k = \mathrm{loc}([s], 1_{|n|})$ and let $[t] = (t_i)_{i\in\mathbb{Z}}$ be given by $[t] = \Phi[s]$. Let $[s'] = (s'_i)_{i\in\mathbb{Z}}$ and $[\hat{s}] = (\hat{s}_i)_{i\in\mathbb{Z}}$ be the intermediate sequences formed during the construction of $[t]$ from $[s]$ by the GL construction.

LEMMA 3.14. *For any i in the range $0 \leqslant i \leqslant 2^{n+1}-1$ the value of t_i can be found using k and at most one value $\sigma_j[s]$, for some j in the range $1 \leqslant j \leqslant 2^n - 1$.*

PROOF. We have $[s'] = \chi_k[s]$, $s_k = 1$ and $[\hat{s}] = D_0^{-1}[s']$, thus

$$\hat{s}_i = \begin{cases} \sigma_i[s] & \text{if } 0 \leqslant i \leqslant k; \\ \sigma_{i+1}[s] - 1 & \text{if } k+1 \leqslant i \leqslant 2^n - 2, \text{ or} \\ \hat{s}_{i-(2^n-1)} + 1 & \text{if } 2^n - 1 \leqslant i \leqslant 2^{n+1} - 3. \end{cases}$$

Considering the final stage of Construction 2.7, since the alphabet size 2 is at

most equal to the span of $[s]$, letting $p = 2^n - 1 + k$, we have

$$t_i = \begin{cases} \hat{s}_i & \text{if } 0 \leqslant i \leqslant p+1, \text{ or} \\ \hat{s}_{i-2} & \text{if } p+2 \leqslant i \leqslant 2^{n+1}-1. \end{cases}$$

This argument gives an explicit means of computing t_i from the information in the statement of the Lemma. □

LEMMA 3.15. *For any i in the range $1 \leqslant i \leqslant 2^{n+1}$ the value of $\sigma_i[t]$ can be found using k, $\eta_{2^n}[s]$ and at most one value $\eta_j[s]$, for some j in the range $2 \leqslant j \leqslant 2^n$.*

PROOF. Letting $p = 2^n - 1 + k$, from the relation of $[t]$ to $[\hat{s}]$ we have

$$\sigma_i[t] = \begin{cases} \sigma_i[\hat{s}] & \text{if } 1 \leqslant i \leqslant p+2, \text{ or} \\ \sigma_{i-2}[\hat{s}] + 1 & \text{if } p+3 \leqslant i \leqslant 2^{n+1}. \end{cases}$$

It is clear that since $[\hat{s}] = D_0^{-1}[s']$, we have $\sigma_\ell[\hat{s}] = \eta_\ell[s']$ for all ℓ in the range $2 \leqslant \ell \leqslant 2^{n+1} - 2$, and that $\sigma_1[\hat{s}] = 0$. Next, since $[s'] = \chi_k[s]$ and $[s]$ has least period 2^n we have

$$\eta_\ell[s'] = \begin{cases} \eta_\ell[s] & \text{if } 2 \leqslant \ell \leqslant k+1; \\ \eta_{\ell+1}[s] - (\ell - k - 1) & \text{if } k+2 \leqslant \ell \leqslant 2^n + k + 1, \text{ or} \\ \eta_{\ell+2}[s] - (\ell - k - 1) & \\ \quad -(\ell - k - 1 - 2^n) & \text{if } 2^n + k + 2 \leqslant \ell \leqslant 2^{n+1} - 2. \end{cases}$$

Finally, if $2^n + 2 \leqslant \ell \leqslant 2^{n+1}$ then $\eta_\ell[s] = \eta_{\ell-2^n}[s] + \eta_{2^n+1}[s] + (\ell - 2^n)\mathrm{wt}\,([s])$. However, notice that the last term here is zero since $\mathrm{wt}\,([s]) = 0$ by Lemma 2.5. Also, since $[s]$ has its maximal run of consecutive zeroes at position 0, we must have $s_{2^n-1} = 1$. Thus we may substitute $\eta_{2^n+1}[s] = \eta_{2^n}[s] + \sigma_{2^n}[s] = \eta_{2^n}[s] + \mathrm{wt}\,([s]) - 1 = \eta_{2^n}[s] + 1$. Together, these relationships show how the computation described in the statement of the Lemma may be performed, as required. □

3.5 The Binary Case

We continue to consider only the binary alphabet. In this Section we shall demonstrate formally the storage reduction technique outlined in the previous Section. Before we begin, we make a slight change to our Construction, involving the initial sequence that the construction uses. The main reason for doing this is to ensure that the conditions $n \geqslant 3$ and that k is odd required by Lemmas 3.9 and 3.10 are satisfied when we employ these Lemmas later. The new construction is given below.

Construction 3.16. Firstly, we define two fixed binary de Bruijn sequences $[\kappa^1]$ and $[\kappa^2]$, with spans 3 and 4 respectively, by

$$\begin{aligned} [\kappa^1] &= [00011101] \\ [\kappa^2] &= [0000100111101011]. \end{aligned}$$

Suppose that we wish to construct a span n de Bruijn sequence, where n is any integer greater than or equal to 4. Let $d_n = \lfloor 1 + \log_2 n \rfloor$ be the number of digits in the binary representation of n and denote this representation by $n_1 n_2 \cdots n_d$, where the most significant digit is to the left (so $n = \sum_{i=1}^{d} n_i 2^{d-i}$). Necessarily, $n_1 = 1$ and $d \geqslant 3$.

We define the sequence $[s^{(3)}]$ according to the value of n_2, by

$$[s^{(3)}] = \begin{cases} \Upsilon[\kappa^1] & \text{if } n_2 = 1, \text{ or} \\ [\kappa^2] & \text{if } n_2 = 0. \end{cases}$$

We then inductively define the sequences $[t^{(i)}]$ for $3 \leqslant i \leqslant d$ and $[s^{(i)}]$ for

$4 \leqslant i \leqslant d$ by

$$[t^{(i)}] = \begin{cases} [s^{(i)}] & \text{if } n_i = 0, \text{ or} \\ \Phi[s^{(i)}] & \text{if } n_i = 1, \text{ and} \end{cases} \tag{3.12}$$

$$[s^{(i+1)}] = \Upsilon[t^{(i)}]. \tag{3.13}$$

□

As in Lemma 3.5, define $m_j = \sum_{i=1}^{j} n_i 2^{j-i}$ for all j in the range $1 \leqslant j \leqslant d$, so that m_j is the integer whose binary representation agrees with the first j digits of that of n. By a similar method to the proof of Lemma 3.5, it may be shown that the sequences $[t^{(i)}]$ and $[s^{(i)}]$ defined in Construction 3.16 are de Bruijn sequences of span m_i and $2m_{i-1}$ respectively, for each i in the range $3 \leqslant i \leqslant d$. Thus Construction 3.16 may be used to construct binary de Bruijn sequences of any span n with $n \geqslant 3$.

For each i in the range $3 \leqslant i \leqslant d$ we define integers $k^{(i)}$ and $k'^{(i)}$ by

$$k'^{(i)} = \mathrm{loc}([s^{(i)}], 1_{|2m_{i-1}|}) \quad \text{and}$$

$$k^{(i)} = \mathrm{loc}([t^{(i)}], 1_{|m_i|}).$$

We also define $[a^{(i)}] = (a_j^{(i)})_{j \in \mathbb{Z}}$, $[b^{(i)}] = (b_j^{(i)})_{j \in \mathbb{Z}}$ and $[d^{(i)}] = (d_j^{(i)})_{j \in \mathbb{Z}}$ for each i in the range $3 \leqslant i \leqslant d-1$ to be those intermediate sequences obtained when applying Construction 3.1 to $[t^{(i)}]$ to obtain $[s^{(i+1)}]$, and define integers $v^{(i)}$ and $m^{(i)}$ by

$$v^{(i)} = \mathrm{loc}([d^{(i)}], X_{2m_i-2}) \quad \text{and} \tag{3.14}$$

$$m^{(i)} = \mathrm{loc}([d^{(i)}], 1_{|2m_i-1|}). \tag{3.15}$$

LEMMA 3.17. *With the above notation*

$$k'^{(3)} = \begin{cases} 7 & \text{if } n_2 = 0, \text{ or} \\ 29 & \text{if } n_2 = 1, \end{cases}$$

and the remaining values of $k'^{(i)}$*, together with the values of* $v^{(i)}$*,* $m^{(i)}$ *and* $k^{(i)}$ *for appropriate ranges of* i *may be found from the following recurrences:*

$$\begin{aligned}
k^{(i)} &= \begin{cases} k'^{(i)} & \text{if } n_i = 0, \text{ or} \\ 2^{2m_{i-1}} - 1 & \text{if } n_i = 1 \end{cases} \\
m^{(i)} &= 2k^{(i)} - 2 + 2^{2m_i - 1}(2^{2m_i} - 2) \\
v^{(i)} &= \begin{cases} (2^{m_i} + 1 - k^{(i)})(2^{2m_i - 1} - 1) & \text{if } k^{(i)} \equiv 1 \pmod 4, \text{ or} \\ (2^{m_i + 1} + 3 - k^{(i)})(2^{2m_i - 1} - 1) & \text{if } k^{(i)} \equiv 3 \pmod 4 \end{cases} \\
k'^{(i+1)} &= \begin{cases} m^{(i)} + 3 & \text{if } v^{(i)} < m^{(i)}, \text{ or} \\ m^{(i)} + 1 & \text{if } v^{(i)} > m^{(i)}. \end{cases}
\end{aligned}$$

Moreover, $k^{(i)}$ *and* $k'^{(i)}$ *are always odd, and* $m^{(i)}$ *and* $v^{(i)}$ *are always even.*

PROOF. Clearly, if $n_2 = 0$ then $k'^{(3)} = \mathrm{loc}([\kappa^2], 1_{|4|}) = 7$. If $n_2 = 1$ then that $k'^{(3)} = 29$ follows from Example 3.2, or from the fact that $\mathrm{loc}([\kappa^1], 1_{|3|}) = 3$ by applying Lemmas 3.9 and 3.10. Notice that both values for $k^{(3)}$ are odd (this was one of the reasons for changing the initial sequence in Construction 3.16 from that used in Construction 3.4).

The formula for $k^{(i)}$ is trivial when $n_i = 0$. When $n_i = 1$ it follows from Lemma 2.14. Notice that these values too are odd, provided that $k'^{(i)}$ is odd.

The formulae for $m^{(i)}$, $v^{(i)}$ and $k'^{(i+1)}$ follow from Lemmas 3.9 and 3.10, provided that $k^{(i)}$ is always odd. Notice that the expressions for $m^{(i)}$ and $v^{(i)}$ are even, implying that $k'^{(i+1)}$ is odd. The result follows for all i by induction.

□

THEOREM 3.18. *Let n be any integer, with $n \geqslant 4$. Then there exists a binary de Bruijn sequence of span n, formed using Construction 3.16, that may be decoded using an algorithm that has a computational complexity of $O(n \log n)$ integer operations and requires $O(1)$ bits of storage.*

PROOF. The decoding technique is like that described in Section 3.3 for sequences over arbitrary alphabets. It suffices to show that all of the information which had previously to be stored may now be computed instead, given tables of size $O(1)$ bits, and to consider the computational complexity of the resulting procedure.

Recall that the decoding procedure involved the combination of an interleaving decoder and a decoder for the GL construction. The interleaving decoder required only the storage of the integers m and v arising from each application of the interleaving construction in the formation of $[t^{(d)}]$. These are the values $m^{(i)}$ and $v^{(i)}$ as defined in equations (3.14) and (3.15) respectively. The GL decoder required the storage of the integers $k^{(i)}$ and the integers $L^{(i)}(j)$. Lemma 3.17 gives relations that allow the $m^{(i)}$, $v^{(i)}$ and $k^{(i)}$ to be computed inductively, rather than them having to be stored in advance. Note that the values of $k'^{(i)}$ are also found in this way. As far as the storage requirements are concerned, it remains to show that the values $L^{(i)}(j) = t_j^{(i)}$, for each j in the range $0 \leqslant j \leqslant 2^{m_i - 1} - 2$ and for each i in the range $3 \leqslant i \leqslant d$ with $n_i = 1$, may also be computed.

We prove the following proposition: for each i in the range $4 \leqslant i \leqslant d$ we may compute the value of $\sigma_j[t^{(i)}]$, for any j in the range $1 \leqslant j \leqslant 2^{m_i}$, using at most two values $\sigma_\ell[t^{(i-1)}]$ and $\sigma_{\ell'}[t^{(i-1)}]$ for some ℓ and ℓ' in the range $1 \leqslant \ell, \ell' \leqslant 2^{m_{i-1}}$.

Suppose that $n_i = 0$. Then $[t^{(i)}] = [s^{(i)}]$, and so the required sum $\sigma_j[t^{(i)}]$ is simply equal to $\sigma_j[s^{(i)}]$. Since $[s^{(i)}] = \Upsilon[t^{(i-1)}]$, Lemma 3.11 shows how this sum may be expressed in terms of at most two sums $\sigma_\ell[t^{(i-1)}]$ and $\sigma_{\ell'}[t^{(i-1)}]$.

Now suppose that $n_i = 1$. Then we have $[t^{(i)}] = \Phi[s^{(i)}]$, and so we may use Lemma 3.15 to express the required sum $\sigma_j[t^{(i)}]$ in terms of at most two double sums $\eta_{2^{2m_{i-1}}}[s^{(i)}]$ and $\eta_{j'}[s^{(i)}]$. Corollary 3.13 shows how the former of these may be found, and Lemma 3.12 expresses the latter in terms of a sum $\sigma_\ell[t^{(i-1)}]$, as required.

The proof of the proposition is complete. By storing the values $\sigma_j[t^{(3)}]$ in a table (of size $O(1)$ bits), the proposition gives rise to an inductive proof of the fact that we may calculate any expression of the form $\sigma_j[t^{(i)}]$ recursively.

We need now remark only that Lemma 3.14 shows how the required values $L^{(i)}(j) = t^{(i)}_j$ may be computed from a sum $\sigma_\ell[s^{(i)}]$, and that this sum may be computed from at most two sums $\sigma_\ell[t^{(i-1)}]$ and $\sigma_{\ell'}[t^{(i-1)}]$ using Lemma 3.11. These sums may be computed recursively by the method just described, and so the need to store the values $L^{(i)}(j)$ has been eliminated.

Theorem 3.7 established that the computational complexity of the simpler form of the decoding procedure, in which storage tables were still required, is $O(n \log n(1 + \log c))$ integer operations, where c represents the alphabet size. Fixing $c = 2$ gives a complexity of $O(n \log n)$ integer operations. We must assess the extra computational requirement introduced by replacing the look-up tables used previously with the calculations described above.

Fix i in the range $3 \leqslant i \leqslant d$. It is clear that the inductive procedure of

Lemma 3.17 for finding $v^{(i)}$, $m^{(i)}$, $k^{(i)}$ and $k'^{(i)}$ requires $O(i)$ integer operations. (Note that when repeating this procedure for another value of i a great deal of calculation is repeated. We accept this repetition, but remark that these values could be calculated just once and stored at very little expense.)

We now consider the computational complexity of the recursive procedure described above for computing the value of $L^{(i)}(j) = t_j^{(i)}$. By the proposition, in the worst case the recursive step doubles the number of expressions that need to be calculated at the next recursive step. Also, from the descriptions of the individual calculations in Section 3.4, the number of operations in each individual recursive step may be bounded above by a constant. Thus computing $L^{(i)}(j)$ requires $O(1 + 2 + 4 + \cdots + 2^{i-3})$ integer operations.

In the worst case, when $n_i = 1$ for all i in the range $3 \leqslant i \leqslant d$, the GL decoder is used for each i in this range. By summing the computational requirements described above over this range of i, we see that in the worst case the total computational requirement for these calculations is for

$$\begin{aligned} O\left(\sum_{i=3}^{d}\left[i + \sum_{\ell=0}^{i-3} 2^{\ell}\right]\right) &= O\left(\sum_{i=3}^{d} i + 2^{i-2} - 1\right) \\ &= O\left(d^2 + 2^{d-1}\right) \\ &= O(n) \end{aligned}$$

integer operations. Comparing this with the $O(n \log n)$ operations already required, we see that the asymptotic computational requirements of the decoding algorithm are not affected by the replacement of the storage tables with the alternative computations we have described. □

Notice that the storage requirements for the above procedure, as well as having

an asymptotic size of $O(1)$ bits, are very small in absolute terms.

For $n < 4$, any binary de Bruijn sequence is trivial to decode using a look-up table. In fact, were such a scheme incorporated into the general procedure for constructing and decoding de Bruijn sequences described in Theorem 3.18, the asymptotic storage requirements would remain at $O(1)$ bits. Thus for any n we have established a construction for binary de Bruijn sequences of span n, with a decoding algorithm requiring just $O(n \log n)$ integer operations and only a small, constant (independent of n) amount of storage. It is common for a small, fixed storage requirement such as this to be considered an integral part of the decoding algorithm, and for the algorithm then to be classed as 'storage-free'.

This is the first time, for any fixed alphabet size, that a construction for de Bruijn sequences of arbitrary span has been exhibited, that allows computationally efficient, storage-free decoding.

4 Decodable de Bruijn Sequences III

4.1 Introduction

The following technique is useful for constructing sequences over alphabets with composite size.

Definition 4.1. Let y and z be integers, with $y, z \geqslant 2$, and let $[r] = (r_i)_{i\in\mathbb{Z}}$ and $[s] = (s_i)_{i\in\mathbb{Z}}$ be sequences of least periods p and q over y-ary and z-ary alphabets $\mathcal{A}$ and $\mathcal{B}$, respectively. Define the sequence $[t] = (t_i)_{i\in\mathbb{Z}}$ of least period $\mathrm{lcm}(p, q)$ over the alphabet $\mathcal{A} \times \mathcal{B}$ of size yz by

$$t_i \quad = \quad (r_i, s_i) \quad \text{for all } i \in \mathbb{Z}.$$

We call $[t]$ the *product* of the sequences $[r]$ and $[s]$, and write $[t] = [r] \times [s]$. □

We now define a class of sequence which we then use with the product operation above. These sequences are a special case of the perfect multi-factors of Mitchell ([40, Definition 5], [39, Definition 2.1]). We shall give this general form later (Definition 4.23), but for now the simpler special case possesses all the properties that we require.

Definition 4.2. Let c, m and n be positive integers, with $c \geqslant 2$. A c-ary sequence $[s]$ of least period mc^n is an (n, m) *spaced de Bruijn sequence* (hereafter an (n, m)-SDB sequence) if and only if for each k in the range $0 \leqslant k \leqslant m - 1$, every possible n-tuple appears in $[s]$ at a unique position i with $i \equiv k \pmod{m}$.

□

The following result is a specialisation of [39, Theorem 5.2]. Again, we shall give

the full result later (Theorem 4.24).

THEOREM 4.3. *Let $[r]$ be a span n de Bruijn sequence over a y-ary alphabet $\mathcal{A}$ and let $[s]$ be a (n, y^n)-SDB sequence over a z-ary alphabet $\mathcal{B}$. Then the sequence $[t] = [r] \times [s]$ is a yz-ary de Bruijn sequence of span n over $\mathcal{A} \times \mathcal{B}$.*

We now consider the decoding problem for sequences formed using the above technique. Let $[r]$, $[s]$, $[t]$, $\mathcal{A}$, $\mathcal{B}$, y, z and n be as defined in Theorem 4.3. Given $w \in (\mathcal{A} \times \mathcal{B})^n$ we wish to find $h = \mathrm{loc}([t], w)$.

An obvious first step is to project w onto $\mathcal{A}$ and $\mathcal{B}$. That is, if $w = (w_0, w_1, \ldots, w_{n-1})$ where $w_i = (u_i, v_i) \in \mathcal{A} \times \mathcal{B}$ for all $0 \leqslant i \leqslant n-1$, then let $w|_{\mathcal{A}}$ denote the n-tuple $u = (u_0, u_1, \ldots, u_{n-1}) \in \mathcal{A}^n$ and let $w|_{\mathcal{B}}$ denote $v = (v_0, v_1, \ldots, v_{n-1}) \in \mathcal{B}^n$. Suppose that $[r]$ is decoded by some other technique, so we may find $f = \mathrm{loc}([r], u)$. Since $[r]$ has least period y^n and by the structure of $[t]$ we have that $h \equiv f \pmod{y^n}$. Thus it remains to find the unique integer g in the range $0 \leqslant g \leqslant z^n - 1$ such that $h = gy^n + f$. This motivates the following definition.

PROBLEM 4.4 (The decoding problem for spaced de Bruijn sequences). Let n and m be positive integers, and let $[s]$ be an (n, m)-SDB sequence over a z-ary alphabet $\mathcal{A}$. Given any $v \in \mathcal{A}^n$ and any integer f in the range $0 \leqslant f \leqslant m-1$, the decoding problem for $[s]$ is to determine the unique integer g in the range $0 \leqslant g \leqslant z^n - 1$ such that $[s]^n_{gm+f} = v$. We write $g = \mathrm{loc}([s], v, f)$. □

By the definition of a spaced de Bruijn sequence, the value g described always exists and is unique.

In summary: let $[t] = [r] \times [s]$, as above. Finding $h = \mathrm{loc}([t], w)$ may be

achieved by first decoding $[r]$ to find $f = \mathrm{loc}([r], u)$ and then decoding $[s]$ to find $g = \mathrm{loc}([s], v, f)$. Then

$$h = gy^n + f. \tag{4.1}$$

In Section 4.2 we present a simple construction for a narrow class of spaced de Bruijn sequences. We generalise this construction in Section 4.3, and discuss the range of values of the parameters for which these sequences may be used to form de Bruijn sequences by the manner of Theorem 4.3 above. In Section 4.4 we assess the complexity and storage requirements of a procedure for solving the decoding problem for our new class of spaced de Bruijn sequences, and consider the implications of our results for the complexity of decoding the de Bruijn sequences thus obtained. We argue that in any practical application in which a decodable de Bruijn sequence over a composite alphabet is required, the sequences we construct are the most readily decodable sequences yet known.

Lastly, in Section 4.5, we adapt our work to the more general scenario alluded to earlier in this introduction. This yields easily decodable classes of what are known as perfect multi-factors and perfect factors. We describe an unusual feature of the perfect factors that result.

4.2 Decodable Spaced de Bruijn Sequences

Let n and z be positive integers, with $z \geqslant 2$. We define the map $\psi_{z,n} : \mathbb{Z}_{z^n} \to \mathbb{Z}_z^n$ by

$$\psi_{z,n}(x) = (w_{n-1}, w_{n-2}, \ldots, w_0)$$

for each $x \in \mathbb{Z}_{z^n}$ where, considered as integers, the w_i and x satisfy $x = \sum_{i=0}^{n-1} w_i z^i$. In other words, $\psi_{z,n}(x)$ is the n-tuple corresponding to the n digit base z representation of the integer x, with the least significant digit positioned to the right. Notice that $\psi_{z,n}$ is a bijection, with inverse $\psi_{z,n}^{-1}$.

Let $w = (w_{n-1}, w_{n-2}, \ldots, w_0)$ be an n-tuple over an arbitrary alphabet. We extend the definition of the left-shift operator E to apply to arbitrary n-tuples as well as sequences, defining

$$E(w) = (w_{n-2}, w_{n-3}, \ldots, w_0, w_{n-1}).$$

We also extend our definition of the [ ] symbol. We have used these brackets to define a sequence from a generating cycle, for example writing $[s] = [012]$. For an n-tuple w as defined above we write $[w]$ to denote the sequence defined by $[w] = [w_{n-1}, w_{n-2}, \ldots, w_0]$.

Let $\mathcal{A}$ denote any alphabet. Following the notation of Mitchell ([41]), let p and q be positive integers and let $u = (u_{p-1}, u_{p-2}, \ldots, u_0) \in \mathcal{A}^p$ and $v = (v_{q-1}, v_{q-2}, \ldots, v_0) \in \mathcal{A}^q$. We define the *concatenation* of u and v, denoted $u\,|\,v$, by

$$u\,|\,v = (u_{p-1}, u_{p-2}, \ldots, u_0, v_{q-1}, v_{q-2}, \ldots, v_0) \in \mathcal{A}^{p+q}.$$

Since this operator is associative, we do not need parentheses in expressions with multiple concatenations. This allows us to to define u^t as denoting the t-fold concatenation of u with itself, for all positive integers t. For any n-tuple u, we define u^0 to be the 0-tuple (the identity with respect to concatenation).

LEMMA 4.5. *Let n and z be integers, with $n \geqslant 1$ and $z \geqslant 2$. Let x be any*

element of $\mathbb{Z}_{z^n}$. *Then for each* i *in the range* $0 \leqslant i < n$ *the functions*

$$A_i : x \mapsto [\psi_{z,n}(x) \,|\, \psi_{z,n}(x)]_i^n \quad \textit{and}$$

$$B_i : x \mapsto [\psi_{z,n}(x) \,|\, \psi_{z,n}(x+1)]_i^n$$

are each bijections from $\mathbb{Z}_{z^n}$ *to* $\mathbb{Z}_z^n$.

(Notice that since the domain of B_i is $\mathbb{Z}_{z^n}$, the addition of 1 to x is performed modulo z^n.)

PROOF. $A_i(x)$ is simply $E^i\psi_{z,n}(x)$, which is a bijection.

Clearly B_i is well defined on $\mathbb{Z}_{z^n}$ and gives an image in $\mathbb{Z}_z^n$. $B_i(x)$ consists of an n-tuple over $\mathbb{Z}_z$, whose leftmost $n-i$ digits are the $n-i$ least significant digits of $\psi_{z,n}(x)$, and whose rightmost i digits are the i most significant digits of $\psi_{z,n}(x+1)$.

Suppose additionally that the leftmost $n-i$ digits of $B_i(x)$ are not all equal to $z-1$. Then the addition of 1 to x does not affect the most significant i digits of the n digit base z representation of x. That is, the i most significant digits of $\psi_{z,n}(x)$ and $\psi_{z,n}(x+1)$ are equal. Thus $B_i(x) = E^i\psi_{z,n}(x)$.

Otherwise, suppose that the leftmost $n-i$ digits of $B_i(x)$ are all equal to $z-1$. Then the addition of 1 to x involves a carry into the i most significant digits of the n digit base z representation of x. Thus $B_i(x) = E^i\psi_{z,n}(x + z^i \bmod z^n)$.

In each case, B_i is described by a composition of simple bijections: addition of a constant is a bijection on $\mathbb{Z}_{z^n}$; $\psi_{z,n}$ is a bijection from $\mathbb{Z}_{z^n}$ to $\mathbb{Z}_z^n$; and E^i is a bijection on $\mathbb{Z}_z^n$. This alone does not imply that B_i is a bijection, since a different composition of functions applies in the two different cases. However, given $B_i(x)$ and i these cases are trivial to distinguish by simply comparing the

leftmost $n-i$ digits of $B_i(x)$ with $z-1$. We deduce that B_i has an inverse, B_i^{-1}. Since $|\mathbb{Z}_{z^n}| = |\mathbb{Z}_z^n|$, B_i is a bijection as claimed. □

We now present our first construction for spaced de Bruijn sequences.

CONSTRUCTION 4.6. Let n, z and k be integers, with $n, k \geqslant 1$ and $z \geqslant 2$. We construct the sequence $[s]$ by

$$[s] = [(\psi_{z,n}(0))^k \,|\, (\psi_{z,n}(1))^k \,|\, \cdots \,|\, (\psi_{z,n}(z^n - 1))^k]. \tag{4.2}$$

□

LEMMA 4.7. *The sequence* $[s]$ *constructed in the manner of Construction 4.6 is a* (n, nk)-SDB *sequence.*

PROOF. It is clear that $[s]$ has least period nkz^n. Let f be any integer in the range $0 \leqslant f < nk$, and consider the following two cases.

CASE 1: $0 \leqslant f < n(k-1)$. Let g be any integer with $0 \leqslant g < z^n$. By the form of $[s]$, we have $[s]^n_{gnk+f} = A_{f \bmod n}(g)$. Recall that $A_{f \bmod n}$ is a bijection by Lemma 4.5. Allowing g to vary, we see that every possible n-tuple over $\mathbb{Z}_z$ appears precisely once as an n-window of $[s]$, starting at a position congruent to f modulo nk.

CASE 2: $n(k-1) \leqslant f < nk$. Again, let g be any integer, $0 \leqslant g < z^n$. The form of $[s]$ gives $[s]^n_{gnk+f} = B_{f \bmod n}(g)$. Similarly to above, $B_{f \bmod n}$ is a bijection by Lemma 4.5, thus every n-tuple over $\mathbb{Z}_z$ appears exactly once in $[s]$ at a position congruent to f modulo nk.

Thus $[s]$ is a (n, nk)-SDB sequence as claimed. □

By allowing k to vary, the above construction can be used to make an (n, m)-SDB

sequence $[s]$ over an alphabet $\mathbb{Z}_z$ for any $z \geqslant 2$, and for any values of n and m with $n \mid m$. Suppose now that we wish to employ Theorem 4.3 to construct a c-ary de Bruijn sequence $[t]$ of span n, where c is composite. We are required to choose a non-trivial factorisation of c, say $c = y \times z$, and construct a y-ary de Bruijn sequence $[r]$ of span n and a z-ary (n, y^n)-SDB sequence $[s]$. If $[s]$ is to be constructed using Construction 4.6 then we require $n \mid y^n$. In other words, Construction 4.6 may only be used to construct a spaced de Bruijn sequence for use in Theorem 4.3 if and only if the desired de Bruijn sequence $[t]$ has an alphabet size c and span n, with the property that c has a non-trivial factor y such that $n \mid y^n$. Moreover, since it is our intention to present an extremely efficient decoding procedure for $[s]$, the decoding of $[r]$ will probably be the most difficult stage in decoding the new sequence $[t]$. To simplify this process, it is therefore desirable that y be as small as possible—ideally, the smallest non-trivial factor of c. Thus the scope for employing Construction 4.6 in practice is somewhat limited. We are motivated to try to generalise Construction 4.6, to improve the range of the parameter m for which we can construct (n, m)-SDB sequences.

4.3 A More General Construction

Fix integers n and z, with $n \geqslant 1$ and $z \geqslant 2$. Let $\mathcal{C}$ denote the $[n+1, n, 2]$ generalised parity check code over $\mathbb{Z}_z$, formed by taking those $(n+1)$-tuples whose coordinates sum to zero in $\mathbb{Z}_z$. That is, $c = (c_n, c_{n-1}, \ldots, c_0)$ is a codeword of $\mathcal{C}$ if and only if $\sum_{i=0}^{n} c_i = 0$.

Remark 4.8. Let $w = (w_n, w_{n-1}, \ldots, w_0)$ be any codeword in $\mathcal{C}$. Then for

any i in the range $0 \leqslant i \leqslant n$, the ith term of w may be found from the other terms by the formula

$$w_i = - \sum_{\substack{0 \leqslant j \leqslant n \\ j \neq i}} w_j. \tag{4.3}$$

□

Let n and z be integers, with $z \geqslant 2$. We define the map $\theta_{z,n} : \mathbb{Z}_{z^n} \to \mathcal{C}$ by defining, for all $x \in \mathbb{Z}_{z^n}$, $\theta_{z,n}(x)$ as the unique codeword in $\mathcal{C}$ whose rightmost n digits are equal to $\psi_{z,n}(x)$. Alternatively, we may think of $\theta_{z,n}(x)$ as being the codeword formed by adding a parity check digit to the left hand end of the n digit base z representation of x.

LEMMA 4.9. *Let n and z be integers, with $n \geqslant 1$ and $z \geqslant 2$. Let x be any element of $\mathbb{Z}_{z^n}$. Then for each i in the range indicated, the functions*

$$C_i : x \mapsto [\psi_{z,n}(x) \,|\, \theta_{z,n}(x)]^n_i \qquad \text{for } 0 \leqslant i < n;$$

$$D_i : x \mapsto [\theta_{z,n}(x) \,|\, \psi_{z,n}(x+1)]^n_i \qquad \text{for } 0 \leqslant i \leqslant n;$$

$$E_i : x \mapsto [\theta_{z,n}(x) \,|\, \theta_{z,n}(x)]^n_i \qquad \text{for } 0 \leqslant i \leqslant n, \text{ and}$$

$$F_i : x \mapsto [\theta_{z,n}(x) \,|\, \theta_{z,n}(x+1)]^n_i \qquad \text{for } 0 \leqslant i \leqslant n$$

are each bijections from $\mathbb{Z}_{z^n}$ to $\mathbb{Z}^n_z$.

PROOF. For each of the functions listed, we will describe a sequence of operations to find x given its image. Since $|\mathbb{Z}_{z^n}| = |\mathbb{Z}^n_z|$, this implies that the function concerned is a bijection.

We begin by considering the function C_i. In the case $i = 0$ we have $C_i(x) = \psi_{z,n}(x)$, so $C_i^{-1} = \psi_{z,n}^{-1}$.

If $i > 0$, pick $x \in \mathbb{Z}_{z^n}$, let $v = (v_{n-1}, v_{n-2}, \ldots, v_0) = C_i(x)$ and let $w = (w_n, w_{n-1}, \ldots, w_0) = \theta_{z,n}(x)$. We have $(w_n, w_{n-1}, \ldots, w_{n-i+1}) = (v_{i-1}, v_{i-2}, \ldots,$

v_0) and $(w_{n-i-1}, w_{n-i-2}, \ldots, w_0) = (v_{n-1}, v_{n-2}, \ldots, v_i)$. Thus the only term of w we lack is w_{n-i}, which is easily found from equation (4.3). It is simple to apply $\psi_{z,n}^{-1}$ to the rightmost n digits of w to recover x, thus C_i is invertible.

Now consider D_i. If $i > 0$ then $D_i(x) = B_{i-1}(x)$ and so by Lemma 4.5 D_i is invertible.

Suppose that $i = 0$. We wish to form $\theta_{z,n}(x) = (w_n, w_{n-1}, \ldots, w_0)$ from $D_i(x) = (v_{n-1}, v_{n-2}, \ldots, v_0)$. We have $w_j = v_{j-1}$ for all j in the range $0 < j \leqslant n$, so by equation (4.3) we may find w_0. As with C_i above, apply $\psi_{z,n}^{-1}$ to the rightmost n digits of w to invert D_i.

Consider E_i. If $i = 0$ then $E_i = D_i$ and is invertible by the method just described. If $i > 0$ then $E_i = C_{i-1}$, which has also already been considered.

Lastly we consider F_i. If $i = 0$ or 1 then $F_i = E_i$ and we are done. Therefore, assume $i \geqslant 2$. For any $x \in \mathbb{Z}_{z^n}$, let $v = (v_{n-1}, v_{n-2}, \ldots, v_0) = F_i(x)$ and let $w = (w_n, w_{n-1}, \ldots, w_0) = \theta_{z,n}(x+1)$. The $(n-i+1)$-tuple $u = (v_{n-1}, v_{n-2}, \ldots, v_{i-1})$ represents the $n-i+1$ least significant digits in $\psi_{z,n}(x)$. It is simple to compute $u' = \psi_{z,n-i+1}(\psi_{z,n-i+1}^{-1}(u) + 1)$—this is simply adding 1 to the value represented in base z by u, modulo z^{n-i+1}. Then u' consists of the $n - i + 1$ least significant digits of $\psi_{z,n}(x + 1)$, namely $(w_{n-i}, w_{n-i-1}, \ldots, w_0)$. Together with $(v_{i-2}, v_{i-3}, \ldots v_0) = (w_n, w_{n-1}, \ldots, w_{n-i+2})$, the only digit of w we do not have is w_{n-i+1}, which may be found via equation (4.3). We find $x + 1$ and thus x in the manner of C_i above. □

The following construction is a generalisation of Construction 4.6.

CONSTRUCTION 4.10. Let n, z, k and ℓ be integers, with $k, \ell \geqslant 0$, $n \geqslant 1$, $z \geqslant 2$

and k, ℓ not both zero. For each $x \in \mathbb{Z}_{z^n}$, we define the $(nk + (n+1)\ell)$-tuple d_x by

$$d_x = (\psi_{z,n}(x))^k \,|\, (\theta_{z,n}(x))^\ell. \tag{4.4}$$

We construct the sequence $[s]$ by

$$[s] = [d_0 \,|\, d_1 \,|\, \cdots \,|\, d_{z^n-1}] \tag{4.5}$$

□

LEMMA 4.11. *The sequence $[s]$ constructed in the manner of Construction 4.10 is a (n, m)-SDB sequence, where $m = nk + (n+1)\ell$.*

PROOF. Suppose that $\ell = 0$. Then Construction 4.10 is identical to Construction 4.6, so the result holds by Lemma 4.7. Henceforth we assume $\ell \neq 0$.

Since there are z^n of the m-tuples d_x in equation (4.5), $[s]$ has period mz^n as required. Let f be any integer in the range $0 \leqslant f < m$, and let v be any element of $\mathbb{Z}_z^n$. We are required to show that for each such f there exists some unique integer g in the range $0 \leqslant g < z^n$ such that $v = [s]^n_{gm+f}$. We consider a number of distinct cases. These are distinguished by f, and for fixed f apply for all g. In each, we shall express $[s]^n_{gm+f}$ in terms of g, using one of the functions described in Lemmas 4.5 and 4.9. Since these functions are bijections from $\mathbb{Z}_{z^n}$ to $\mathbb{Z}_z^n$, this will imply that the integer g described above exists and is unique.

CASE 1: $k \neq 0$ and $f < n(k-1)$. For each g in the range $0 \leqslant g < z^n$ we have $[s]^n_{gm+f} = A_{f \bmod n}(g)$. Thus $g = \mathrm{loc}([s], v, f) = A^{-1}_{f \bmod n}(v)$.

CASE 2: $k \neq 0$ and $n(k-1) \leqslant f < nk$. For each g in the range $0 \leqslant g < z^n$ we have $[s]^n_{gm+f} = C_{f \bmod n}(g)$. Thus $g = \mathrm{loc}([s], v, f) = C^{-1}_{f \bmod n}(v)$.

CASE 3: $k \neq 0$ and $nk \leqslant f < nk + (n+1)(\ell - 1)$. For each g in the range $0 \leqslant g < z^n$ we have $[s]^n_{gm+f} = E_{(f-nk) \bmod (n+1)}(g)$. Thus $g = \mathrm{loc}([s], v, f) = E^{-1}_{(f-nk) \bmod (n+1)}(v)$.

CASE 4: $k \neq 0$ and $nk + (n+1)(\ell - 1) \leqslant f < m$. For each g in the range $0 \leqslant g < z^n$ we have $[s]^n_{gm+f} = D_{(f-nk) \bmod (n+1)}(g)$. Thus $g = \mathrm{loc}([s], v, f) = D^{-1}_{(f-nk) \bmod (n+1)}(v)$.

CASE 5: $k = 0$ and $f < (n+1)(\ell - 1)$. For each g in the range $0 \leqslant g < z^n$ we have $[s]^n_{gm+f} = E_{f \bmod (n+1)}(g)$. Thus $g = \mathrm{loc}([s], v, f) = E^{-1}_{f \bmod (n+1)}(v)$.

CASE 6: $k = 0$ and $(n+1)(\ell - 1) \leqslant f$. For each g in the range $0 \leqslant g < z^n$ we have $[s]^n_{gm+f} = F_{f \bmod (n+1)}(g)$. Thus $g = \mathrm{loc}([s], v, f) = F^{-1}_{f \bmod (n+1)}(v)$.

□

We now address the following naturally occurring question: given n and z, for what values m does Construction 4.10 allow the construction of a z-ary (n, m)-SDB sequence?

LEMMA 4.12. *Let n and m be positive integers. Then Construction 4.10 can be used to form an (n, m)-SDB sequence over an alphabet of arbitrary size if and only if*

$$m \bmod n \leqslant \left\lfloor \frac{m}{n} \right\rfloor . \tag{4.6}$$

PROOF. Suppose that n and m satisfy the inequality (4.6). Let $\ell = m \bmod n$ and let $k = \lfloor m/n \rfloor - \ell$. The hypothesis ensures that $k \geqslant 0$. Notice that

$$\begin{aligned} m &= n \times \left\lfloor \frac{m}{n} \right\rfloor + (m \bmod n) \\ &= n(k + \ell) + \ell \end{aligned}$$

$$= \quad nk + (n+1)\ell,$$

so by Lemma 4.11 the sequence $[s]$ constructed using these values of k and ℓ in Construction 4.10 is an (n, m)-SDB sequence as required.

Now suppose that $[s]$ is an (n, m)-SDB sequence formed using Construction 4.10. For some integers $k, \ell \geqslant 0$, not both zero, we have $m = nk + (n+1)\ell = n(k + \ell) + \ell$. Thus

$$\begin{aligned} m \bmod n &= \ell \bmod n \\ &\leqslant k + \ell \\ &\leqslant \left\lfloor \frac{m}{n} \right\rfloor, \end{aligned}$$

as required. The last stage follows since $m = nk + (n+1)\ell$ implies that $k + \ell \leqslant m/n$ and from the fact that k and ℓ are integers. □

LEMMA 4.13. *Let n and c be integers, with $n, c \geqslant 2$. Then*

$$c^n \bmod n \leqslant \left\lfloor \frac{c^n}{n} \right\rfloor.$$

PROOF. Since $c^n \bmod n \leqslant n - 1$ it suffices to show that

$$n - 1 \leqslant \left\lfloor \frac{c^n}{n} \right\rfloor. \tag{4.7}$$

To do this, we demonstrate that

$$n(n-1) \leqslant c^n; \tag{4.8}$$

the inequality (4.7) follows on dividing each side by n and from the fact that $n - 1$ is an integer. For c fixed, the inequality (4.8) is clearly true for large n since the left hand side is a polynomial function in n and the right hand side is

exponential in n. Also, equation (4.8) is clearly true for a given n and all $c \geqslant 3$ if it is true for that value n with $c = 2$. Thus it suffices to show that for all $n \geqslant 2$ we have

$$n(n-1) \leqslant 2^n. \tag{4.9}$$

This holds for $n = 2$ and $n = 3$, and since

$$\frac{n(n-1)/(n-1)(n-2)}{2^n/2^{n-1}} = \frac{n/(n-2)}{2} \leqslant 1$$

for all $n \geqslant 4$, the result follows by induction on n. □

These preliminary results allow us now to analyse the scope of Construction 4.10.

THEOREM 4.14. *Let n and c be integers, with $n, c \geqslant 2$ and with c composite. Let y be any non-trivial factor of c, and let $[r]$ be any y-ary span n de Bruijn sequence. Define $z = c/y$. Then there exists a c-ary span n de Bruijn sequence $[t] = [r] \times [s]$, where $[s]$ is a z-ary (n, y^n)-SDB sequence constructed in the manner of Construction 4.10.*

PROOF. Since $[r]$ is a y-ary de Bruijn sequence of span n, we know that it has least period y^n. By Lemma 4.13 we have $y^n \bmod n \leqslant \lfloor y^n/n \rfloor$, so by Lemmas 4.11 and 4.12 we may use Construction 4.10 to form a (n, y^n)-SDB sequence $[s]$. By Theorem 4.3, the sequence $[t] = [r] \times [s]$ is a span n de Bruijn sequence over an alphabet of size $c = yz$. □

EXAMPLE 4.15. Suppose we wish to construct a c-ary de Bruijn sequence of span n, with $c = 15$ and $n = 2$. Factoring c, let $y = 3$ and $z = 5$. Let $[r] = [002212011]$, a de Bruijn sequence of span 2 over $\mathbb{Z}_3$. Since the least period of $[r]$ is $y^n = 9$, we seek to construct a $(2, 9)$-SDB sequence $[s]$ over $\mathbb{Z}_5$. Following

Lemma 4.11, we require integers k and ℓ such that $9 = 2k + 3\ell$. We may choose either $k = 0$ and $\ell = 3$, or $k = 3$ and $\ell = 1$; in this Example we opt for the latter. From equation (4.5), $[s]$ is formed by concatenating the 9-tuples d_x for x running from 0 to $z^n - 1 = 24$, where d_x is as given in equation (4.4). That is,

$$
\begin{aligned}
[s] = [&00\ 00\ 00\ 000\ 01\ 01\ 01\ 401\ 02\ 02\ 02\ 302\ 03\ 03\ 03\ 203\ 04\ 04\ 04\ 104\\
&10\ 10\ 10\ 410\ 11\ 11\ 11\ 311\ 12\ 12\ 12\ 212\ 13\ 13\ 13\ 113\ 14\ 14\ 14\ 014\\
&20\ 20\ 20\ 320\ 21\ 21\ 21\ 221\ 22\ 22\ 22\ 122\ 23\ 23\ 23\ 023\ 24\ 24\ 24\ 424\\
&30\ 30\ 30\ 230\ 31\ 31\ 31\ 131\ 32\ 32\ 32\ 032\ 33\ 33\ 33\ 433\ 34\ 34\ 34\ 334\\
&40\ 40\ 40\ 140\ 41\ 41\ 41\ 041\ 42\ 42\ 42\ 442\ 43\ 43\ 43\ 343\ 44\ 44\ 44\ 244].
\end{aligned}
$$

(The spaces serve no purpose other than to clarify the structure.) The product of $[r]$ and $[s]$ is then a span 2 de Bruijn sequence over $\mathbb{Z}_3 \times \mathbb{Z}_5$. □

4.4 Decoding

We now demonstrate how the structure of the spaced de Bruijn sequences arising from Construction 4.10 gives rise to an efficient decoding procedure.

We take the sequence $[s]$ to be as defined in Construction 4.10, that is an (n, m)-SDB sequence over $\mathbb{Z}_z$, where $n \geqslant 1$, $z \geqslant 2$ and $m = nk + (n+1)\ell$, where $k, \ell \geqslant 0$ and k and ℓ are not both zero. Thus n, z, k and ℓ are parameters for our algorithm. The other input to our algorithm is $v \in \mathbb{Z}_z^n$, and an integer f in the range $0 \leqslant f < m$. The output shall be $g = \mathrm{loc}([s], v, f)$, that is the unique integer g in the range $0 \leqslant g < z^n$ with $[s]_{gm+f}^n = v$. The algorithm is as follows.

Algorithm 4.16.

`Input` v, f, n, z, k, ℓ

`Output` $g = \mathrm{loc}([s], v, f)$

```
If ℓ = 0
    Let t = f mod n
    If f < n(k − 1)
        Return A_t^{-1}(v)                    (Lemma 4.7, case 1)
    Else
        Return B_t^{-1}(v)                    (Lemma 4.7, case 2)
    End if
Else
    If k = 0
        Let t = f mod (n + 1)
        If f < (n + 1)(ℓ − 1)
            Return E_t^{-1}(v)                (Lemma 4.11, case 5)
        Else
            Return F_t^{-1}(v)                (Lemma 4.11, case 6)
        End if
    Else
        If f < nk
            Let t = f mod n
            If f < n(k − 1)
                Return A_t^{-1}(v)            (Lemma 4.11, case 1)
            Else
                Return C_t^{-1}(v)            (Lemma 4.11, case 2)
            End if
        Else
```

```
                Let t = (f − nk) mod (n + 1)
                If f ⩽ nk + (n + 1)(ℓ − 1)
                        Return E_t^{-1}(v)          (Lemma 4.11, case 3)
                Else
                        Return D_t^{-1}(v)          (Lemma 4.11, case 4)
                End if
            End if
        End if
End if
```

□

The algorithm determines which case in the proofs of Lemmas 4.7 and 4.11 is applicable and computes the decoding of the given n-tuple accordingly. These cases are indicated on the right. Thus the correctness of Algorithm 4.16 follows directly from the proofs of Lemmas 4.7 and 4.11.

THEOREM 4.17. *Let $[s]$ be an (n, m)-*SDB *sequence formed by means of Construction 4.10. Then $[s]$ may be decoded using Algorithm 4.16, this procedure having a computational complexity of $O(n)$ integer operations and requiring $O(1)$ bits of storage.*

PROOF. We note that the Algorithm does not require the storage of any tables of information, and move on to consider its computational complexity. It is no longer clear how to interpret our assumption of Section 1.2 regarding the maximum value of the integers that need to be stored in a single computer register. We take the conservative view, and assume only that the maximum value that may reasonably arise is bounded above by mz^n, the least period

of $[s]$. As before, we note that many computations will involve only much smaller values.

The operations performed by Algorithm 4.16 are as follows:

(1) a sequence of comparisons, concerning f, n, k and ℓ to determine which of the cases in which of the proofs of Lemmas 4.7 and 4.11 applies, and

(2) finding one of the inverses $A_t^{-1}(v), B_t^{-1}(v), \ldots, F_t^{-1}(v)$. The proofs of Lemmas 4.5 and 4.9 show that each of these inverse functions may be computed using the following three step process:

   (a) computing some n-tuple v' (say) from v and t;

   (b) finding $g' = \psi_{z,n}^{-1}(v')$, and

   (c) finding the output g from g'.

Note that the structure of the algorithm involves simply distinguishing between a fixed, small number of cases and then computing the desired result accordingly, with no possibility of 'looping'. Thus it is clear that the number of integer operations required to complete Step (1) above is upper bounded by a constant. (It may also be worth noting that all of these operations involve integers bounded above by m.)

We analyse the three steps of stage (2). The operations described below are taken directly from the procedures given for inverting the functions $A_t, B_t, \ldots, F_t$ in the proofs of Lemmas 4.5 and 4.9.

Step (2)(a) involves operations on n-tuples of digits over $\mathbb{Z}_z$. In the case of F_t^{-1} only, the first stage is to add 1 to the value represented in base z by v, modulo z^{n-t+1}. This may be performed using at most $n - t + 1$ operations.

Next, in almost every case, a comparison (or two, in the case of F_t^{-1}) involving t is required. In some cases, such as for B_t^{-1}, a further $n - t$ comparisons of digits of v may also be performed. At this stage, in each case at least $n - 1$ of the digits of v' may be obtained by copying appropriate digits of v. Lastly, in the cases of $C_t^{-1}, D_t^{-1}, \ldots, F_t^{-1}$ the last digit of v' may need to be computed via equation (4.3). Considering each of the above stages, we conclude that step (2)(a) of Algorithm 4.16 may be completed in $O(n)$ integer operations. (We remark again that the parameters in most of these operations are bounded by integers far smaller than the mz^n we assumed previously.)

Step (2)(b) is the evaluation of the integer g' in the range $0 \leqslant g' < z^n$ represented in base z by the n-tuple v'. All the integers concerned in this process are bounded by z^n, and so certainly by mz^n. A straightforward approach uses $O(n)$ integer operations.

Finally, step (2)(c) involves possibly subtracting z^i from g' modulo z^n, for some integer i in the range $0 \leqslant i < n$. This may be best achieved by performing step (2)(c) before step (2)(b), that is by working on v', the n-digit base z representation of g'. This requires $O(n)$ integer operations (with integers bounded by z).

The above argument shows that Algorithm 4.16 requires just $O(n)$ integer operations. □

Now suppose $[t] = [r] \times [s]$ is a span n c-ary de Bruijn sequence, constructed in the manner of Theorem 4.14. As discussed in Section 4.1, decoding $[t]$ requires one decoding of $[r]$, one decoding of $[s]$ and a simple calculation as given by

equation (4.1). The latter requires at most a constant number of mz^n-ary operations. We conclude that the decoding problem for $[t]$ may be solved using $O(n)$ integer operations, plus the computation required for a single decoding of $[r]$.

Decoding any span n de Bruijn sequence will always require $\Omega(n)$ operations, since this is the number of operations required to process the input. Thus the decoding procedure for $[t]$ described above has asymptotically optimal complexity, except for the effort required to perform a single decoding of the de Bruijn sequence $[r]$. Also, we impose no restriction on the techniques used to construct and decode $[r]$—any current or future technique may be employed. Thus we argue that the methods of this Chapter represent an extremely powerful set of tools for constructing a wide class of easily decodable de Bruijn sequences. This idea is presented formally below. DEFINITION 4.18. We write $\mathcal{D}(n, c)$ to denote the least possible complexity, in terms of operations on integers bounded by c^n, for an algorithm for decoding a c-ary de Bruijn sequence of span n that uses $O(1)$ bits of storage. □

REMARK 4.19. We should demonstrate that $\mathcal{D}(n, c)$ is well defined. Firstly, note that if c is a prime power, then there exists a c-ary de Bruijn sequence of span n formed from an m-sequence over $\mathbb{F}_c$. Decoding for this sequence may be achieved by an algorithm requiring just $O(1)$ bits of storage, albeit one that is exponential in its computational complexity, by simply generating the m-sequence term-by-term until the required n-window is reached. Thus $\mathcal{D}(n, c)$ is well defined in this case.

Now suppose that c is not a prime power. We may always express c in terms of its factorisation, say $c = \prod_{i=1}^{k} p_i^{r_i}$, where the p_i are distinct primes and the r_i

and k are positive integers. We may form $p_i^{r_i}$-ary de Bruijn sequences $[s^{(i)}]$ from m-sequences over $\mathbb{F}_{p_i^{r_i}}$, for each $i = 1, 2, \ldots, k$. By the above argument, each of the sequences $[s^{(i)}]$ may be decoded using $O(1)$ bits of storage. Now Theorems 4.14 and 4.17 show that any one of the sequences $[s^{(i)}]$ may be extended to a c-ary de Bruijn sequence, and that the resulting sequence may be decoded using $O(1)$ bits of storage. This shows that $\mathcal{D}(n, c)$ is well defined for all n and c.

This result, that algorithms for decoding de Bruijn sequences using $O(1)$ bits of storage exist for arbitrary n and c, and not just those values c which are prime powers, is not new to this thesis. For example, the sequences $[s^{(i)}]$ given above may be combined into the product $[s^{(1)}] \times [s^{(2)}] \times \cdots \times [s^{(k)}]$ which, since the sequences $[s^{(i)}]$ have pair-wise co-prime least periods, is also a de Bruijn sequence and has alphabet size c. Decoding for this sequence may be achieved using $O(1)$ bits of storage by decoding each of the sequences $[s^{(i)}]$ separately, in the manner described previously. □

The following result summarises the significance of the results in this Chapter to the complexity of algorithms for decoding de Bruijn sequences, as measured by the expression $\mathcal{D}(n, c)$ defined above.

THEOREM 4.20. *Let n and c be positive integers, with $c \geqslant 2$. If c is composite with non-trivial factor y, then $\mathcal{D}(n, c) = O(\mathcal{D}(n, y))$.*

PROOF. Theorems 4.14 and 4.17 have shown that $\mathcal{D}(n, c) = \mathcal{D}(n, y) + O(n)$. Since it requires $\Omega(n)$ operations to process the input to Problem 1.3, we have $\mathcal{D}(n, y) = \Omega(n)$. The result follows. □

As a final comment for our work on decoding de Bruijn sequences, we note

that the sequence $[r]$ above may be constructed by the means described in Chapters 2 or 3. Employing those results together with the results of this Chapter and those of [42] gives the following Theorem.

THEOREM 4.21. *Let n and c be positive integers, with $c \geqslant 2$, and let a and b be positive integers such that $n = a2^b$ with a odd. Then there exists a c-ary de Bruijn sequence $[t]$ possessing a decoding algorithm with the following requirements:*

(1) for c even, the storage requirement is bounded by a constant and the computational requirement by $O(n \log n)$ integer operations;

(2) for c odd and composite, letting $c = yz$ be any non-trivial factorisation of c, the storage requirement is for $O(y^a)$ bits and the computational requirement is for $O(n \log n(1 + \log y))$ integer operations, and finally

(3) for c an odd prime, the storage requirements are for $O(c^a)$ bits and the computational requirement is for $O(n \log n(1 + \log c))$ integer operations.

PROOF. For case (1), we construct a binary de Bruijn sequence of span n using Construction 3.16, and (for $c \neq 2$) combine this with a $(c/2)$-ary $(n, 2^n)$-SDB sequence formed using Construction 4.10. The decoding is achieved using the separate decoding algorithms for those two sequences as presented in Chapter 3 and this Chapter.

In case (2) we do exactly as in case (1) above, except that the de Bruijn sequence formed has an odd alphabet size of y rather than being binary. Therefore we form this y-ary sequence using Construction 3.4. The decoding of this latter sequence dominates the overall complexity and storage requirements, and

so their values are as obtained in Theorem 3.7.

Case (3) is achieved using Construction 3.4 and its associated decoding algorithm alone, the techniques of this Chapter not being applicable. The result follows from Theorem 3.7. □

4.5 Perfect Multi-Factors

First, we describe the structures called perfect factors and perfect multi-factors that were mentioned in Section 4.1.

Definition 4.22 ([39, Definition 1.1]). A set of sequences $\mathbb{S} = \{[s^{(0)}], [s^{(1)}], \ldots, [s^{(\ell-1)}]\}$ over a c-ary alphabet $\mathcal{A}$, each having least period t, is an *(n, t, c) perfect factor* (hereafter an (n, t, c)-PF) if and only if every n-tuple over $\mathcal{A}$ appears at a unique position (modulo t) in exactly one of the sequences $[s^{(i)}]$. □

Notice that we must have $t\ell = c^n$, and hence the notation is able to suppress dependence on ℓ. Whilst a de Bruijn sequence corresponds to a Hamiltonian cycle in the de Bruijn graph (see Section 1.2), an (n, t, c)-PF corresponds to a set of $\ell = c^n/t$ vertex disjoint cycles (called factors), each of length t. De Bruijn sequences are special cases of perfect factors, in which $\ell = 1$ and $t = c^n$.

Perfect Factors were first studied in the binary case by Etzion ([17]), and later for arbitrary alphabets by Paterson ([48]) and Mitchell ([39]). They have a particular application in the construction of two-dimensional arrays with an analogous window property to the window sequences we have been studying ([17], [48]). A variation of this array construction is considered in Section 5.9 of this thesis.

DEFINITION 4.23 ([40, Definition 5], [39, Definition 2.1]). Let n, t, m, c and ℓ be integers, such that $n, t, m, \ell \geqslant 1$ and $c \geqslant 2$. An (n, t, m, c) *perfect multi-factor* (hereafter (n, t, m, c)-PMF) is a set $\{[s^{(0)}], [s^{(1)}], \ldots, [s^{(\ell-1)}]\}$ of sequences over a c-ary alphabet $\mathcal{A}$ such that for every $w \in \mathcal{A}^n$ and every k in the range $0 \leqslant k < m$ there exists a unique pair of integers i and j with $0 \leqslant i < \ell$, $0 \leqslant j < t$ such that $w = [s^{(i)}]^n_{jm+k}$. □

Again, we have $\ell = c^n/t$. Notice that an (n, c^n, m, c)-PMF is a c-ary (n, m)-SDB sequence, an $(n, t, 1, c)$-PMF is an (n, t, c)-PF, and that an $(n, c^n, 1, c)$-PMF (or (n, c^n, c)-PF) is a de Bruijn sequence. (To avoid some notational clumsiness, we may call $[s]$ a perfect (multi-)factor in place of $\{[s]\}$ in the case $\ell = 1$.)

Theorem 4.3 is a special case of Theorem 4.24 below.

THEOREM 4.24 ([39, Theorem 5.2]). *Let* $R = \{[r^{(0)}], [r^{(1)}], \ldots, [r^{(\ell-1)}]\}$ *be an* (n, m, y)-PF *over* $\mathcal{A}$, *and let* $S = \{[s^{(0)}], [s^{(1)}], \ldots, [s^{(\ell'-1)}]\}$ *be an* (n, t, m, z)-PMF *over* $\mathcal{B}$. *Define*

$$T = \{[r^{(i)}] \times [s^{(j)}] \,|\, 0 \leqslant i < \ell \text{ and } 0 \leqslant j < \ell'\}.$$

Then T *is an* (n, mt, yz)-PF *over* $\mathcal{A} \times \mathcal{B}$.

For there to exist an (n, t, c)-PF we require that $t \,|\, c^n$ and either $n = t = 1$ or $n + 1 \leqslant t$. Similarly, the existence of an (n, t, m, c)-PMF requires that $t \,|\, c^n$ and either $t = 1$ and $n \leqslant mt$ or $t > 1$ and $n < mt$. These conditions were established by Mitchell ([39]), who also conjectured that they suffice to guarantee existence of perfect (multi-)factors with the parameters as specified. The general case of this conjecture remains open, although considerable progress has been made ([44]).

In this Section we present a generalisation of Construction 4.10, and then demonstrate how it may be used to construct a class of perfect multi-factors. We define the decoding problem for perfect multi-factors, and discuss the efficiency with which our perfect multi-factors may be decoded. We also show that our perfect multi-factors exhibit an unusual relationship to each other.

We shall make use of the following function: let j and z be integers, with $j \geqslant 0$ and $z \geqslant 2$. We define the function $f_{z,j} : \mathbb{Z}^+ \to \mathbb{Z}^+$ by

$$f_{z,j}(x) = x - (x \bmod z^j) + (x + 1 \bmod z^j)$$

for all $x \in \mathbb{Z}$. The operation performed by $f_{z,j}$ is to replace the least significant j digits in the base z representation of x by the least significant j digits in the base z representation of $x + 1$. We also allow this function to be used in rings $\mathbb{Z}_{z^n}$, and remark that it is clearly a bijection on $\mathbb{Z}_{z^n}$ for all $n \geqslant 0$.

Recall that a code $\mathcal{C}$ is systematic on a particular set of n coordinate positions if and only if every possible n-tuple over the alphabet appears in those coordinate positions in exactly one codeword.

DEFINITION 4.25. Let m, n and z be positive integers, with $m \geqslant n$ and $z \geqslant 2$. We say that $\mathcal{C}$ is an $\mathrm{X}(m, n, z)$ code if and only if it is a length m code over $\mathbb{Z}_z$, with the property that it is systematic on any n consecutive coordinate positions (considered cyclicly, with the left hand end of the codewords adjacent to the right). □

REMARK 4.26. Let $\mathcal{C} = \{c_0, c_1, \ldots, c_{\ell-1}\}$ be an $\mathrm{X}(m, n, z)$ code. Then the set $\mathcal{S} = \{[s^{(0)}], [s^{(1)}], \ldots, [s^{(\ell-1)}]\}$ given by $[s^{(i)}] = [c_i]$ for all i in the range $0 \leqslant i \leqslant \ell - 1$ is an $(n, 1, m, z)$-PMF. This process may be reversed: the length m

generating cycles of the sequences in an $(n, 1, m, c)$-PMF form an $\mathrm{X}(m, n, z)$ code. Thus these two combinatorial objects are equivalent. □

Let m, m', n and z be positive integers, with $m, m' \geqslant n \geqslant 1$ and $z \geqslant 2$. Let $\mathcal{C}$ and $\mathcal{C}'$ be $\mathrm{X}(m, n, z)$ and $\mathrm{X}(m', n, z)$ codes, respectively. Let $c_0, c_1, \ldots, c_{z^n-1}$ be an ordering of the codewords of $\mathcal{C}$, such that if c_x is written in coordinates as $(c_{x,m-1}, c_{x,m-2}, \ldots, c_{x,0})$, then $(c_{x,n-1}, c_{x,n-2}, \ldots, c_{x,0}) = \psi_{z,n}(x)$. That is, the rightmost n digits of c_x form the n digit base z representation of x. Let $c'_0, c'_1, \ldots, c'_{z^n-1}$ be codewords of $\mathcal{C}'$, ordered similarly.

LEMMA 4.27. *With the notation above, for any i in the range $0 \leqslant i \leqslant m+m'-n$, and any j in the range $0 \leqslant j \leqslant n$, define the functions G_i and $H_{i,j}$ by*

$$\begin{aligned} G_i(x) &= [c_x \,|\, c'_x]^n_i \quad \text{and} \\ H_{i,j}(x) &= [c_x \,|\, c'_{f_{z,j}(x)}]^n_i, \end{aligned}$$

for any $x \in \mathbb{Z}_{z^n}$. Then G_i and $H_{i,j}$ are each bijections from $\mathbb{Z}_{z^n}$ to $\mathbb{Z}_z^n$.

PROOF. Suppose $0 \leqslant i \leqslant m - n$. Then $G_i(x) = H_{i,j}(x) = [c_x]^n_i$. Similarly, if $m \leqslant i \leqslant m + m' - n$, then $G_i(x) = [c'_x]^n_{i-m}$ and $H_{i,j}(x) = [c'_{f_{z,j}(x)}]^n_{i-m}$. In each case, since $\mathcal{C}$ and $\mathcal{C}'$ are systematic on any n consecutive coordinates, and since $f_{z,j}$ is a bijection on $\mathbb{Z}_{z^n}$, we deduce that G_i and $H_{i,j}$ are each bijections.

Now suppose $m - n + 1 \leqslant i \leqslant m - 1$, and write $v = (v_{n-1}, v_{n-2}, \ldots, v_0) = G_i(x)$. We have $(v_{n-1}, v_{n-2}, \ldots, v_{n-m+i}) = (c_{x,m-1-i}, c_{x,m-2-i}, \ldots, c_{x,0})$ and $(v_{n-m-1+i}, v_{n-m-2+i}, \ldots, v_0) = (c'_{x,m'-1}, c'_{x,m'-2}, \ldots, c'_{x,m'+m-n-i})$. By their definitions, the codewords c_x and c'_x agree on their rightmost n coordinates.

Thus

$$v = (c'_{x,m-1-i}, c'_{x,m-2-i}, \ldots, c'_{x,0}, c'_{x,m'-1}, c'_{x,m'-2}, \ldots, c'_{x,m'+m-n-i})$$

That is, v consists of n consecutive coordinates from $\mathcal{C}'$, and since $\mathcal{C}'$ is systematic on these coordinates G_i is a bijection.

Finally, suppose again that $m-n+1 \leqslant i \leqslant m-1$, and write $v = (v_{n-1}, v_{n-2}, \ldots, v_0) = H_{i,j}(x)$. As above, $(v_{n-1}, v_{n-2}, \ldots, v_{n-m+i}) = (c_{x,m-1-i}, c_{x,m-2-i}, \ldots, c_{x,0})$, the rightmost $m-i$ digits of $\psi_{z,n}(x)$. It is a simple matter to convert these digits into the rightmost $m-i$ digits of $\psi_{z,n}(f_{z,j}(x))$—call these $(v'_{n-1}, v'_{n-2}, \ldots, v'_{n-m+i})$, say. Then

$$(v'_{n-1}, v'_{n-2}, \ldots, v'_{n-m+i}, v_{n-m+i-1}, v_{n-m+i-2}, \ldots, v_0) =$$
$$(c'_{x,m-1-i}, c'_{x,m-2-i}, \ldots, c'_{x,0}, c'_{x,m'-1}, c'_{x,m'-2}, \ldots, c'_{x,m'+m-n-i}).$$

Once again, since $\mathcal{C}'$ is systematic on these n consecutive coordinates, and since $f_{i,j}$ is a bijection on $\mathbb{Z}_{z^n}$, we have that $H_{i,j}$ is a bijection, as required. □

We present a construction for perfect multi-factors.

CONSTRUCTION 4.28. Let n and e be positive integers, and let z be any integer with $z \geqslant 2$. For each i in the range $0 \leqslant i \leqslant e-1$, let $m^{(i)}$ be an integer with $m^{(i)} \geqslant n$, and let $M = \sum_{i=0}^{e-1} m^{(i)}$. Let $\mathcal{C}^{(i)}$ be any $\mathrm{x}(m^{(i)}, n, z)$ code, with codewords $c_0^{(i)}, c_1^{(i)}, \ldots, c_{z^n-1}^{(i)}$ ordered according to their rightmost n coordinates, as above. For each x in the range $0 \leqslant x \leqslant z^n - 1$ define the M-tuple d_x by

$$d_x = c_x^{(0)} \,|\, c_x^{(1)} \,|\, \cdots \,|\, c_x^{(e-1)}.$$

Let b be any integer in the range $0 \leqslant b \leqslant n$, let $t = z^b$ and let $\ell = z^n/t = z^{n-b}$.

We define the ℓ sequences $[s^{(0)}], [s^{(1)}], \ldots, [s^{(\ell-1)}]$ by

$$[s^{(i)}] = [d_{it} \,|\, d_{it+1} \,|\, \cdots \,|\, d_{(i+1)t-1}],$$

for each i in the range $0 \leqslant i \leqslant \ell - 1$. □

THEOREM 4.29. *The set of sequences* $\mathcal{S} = \{[s^{(0)}], [s^{(1)}], \ldots, [s^{(\ell-1)}]\}$ *given by Construction 4.28 forms an* (n, t, M, z)-PMF.

PROOF. We are required to show that for any integer k in the range $0 \leqslant k \leqslant M - 1$, all n-tuples occur as windows of exactly one sequence of $\mathcal{S}$ at a position congruent to k modulo M.

For each u in the range $0 \leqslant u \leqslant e$, define the integers M_u iteratively, by $M_0 = 0$ and $M_{u+1} = M_u + m^{(u)}$. Thus M_u represents the position of the first digit of $c_x^{(u)}$ in d_x, except when $u = e$ in which case $M_u = M$.

Suppose $M_u \leqslant k \leqslant M_{u+1} - n$, for some u in the range $0 \leqslant u \leqslant e - 1$. The n-windows of sequences in $\mathcal{S}$ at positions congruent to $k \pmod M$ are $[c_x^{(u)}]_{k-M_u}^n$, for some x. For any x in the range $0 \leqslant x \leqslant z^n - 1$, define $i = \lfloor x/t \rfloor$ and $j = x \bmod t$. We have that $[s^{(i)}]_{jM+k}^n = [c_x^{(u)}]_{k-M_u}^n$, and since $\mathcal{C}^{(u)}$ is systematic on all windows of n consecutive coordinate positions, we have that every n-tuple must appear in exactly one sequence of $\mathcal{S}$ at a position congruent to k modulo M.

Now if $e \geqslant 2$, suppose $M_u - n + 1 \leqslant k \leqslant M_{u+1} - 1$, for some u in the range $1 \leqslant u \leqslant e - 2$ Then the n-windows of the sequences of $\mathcal{S}$ at positions congruent to k modulo M lie partly in a word from $\mathcal{C}^{(u)}$ and partly in a word from $\mathcal{C}^{(u+1)}$. More precisely, for any x in the range $0 \leqslant x \leqslant z^n - 1$, letting i and j be defined as above we have that $[s^{(i)}]_{Mj+k}^n = [c_x^{(u)} \,|\, c_x^{(u+1)}]_{k-M_u}^n$. The function on the right hand side has the same form as the function G_{k-M_u} of Lemma 4.27, and so when

viewed as a function of x is a bijection from $\mathbb{Z}_{z^n}$ to $\mathbb{Z}_z^n$. The result then follows for these values of k.

Finally, we suppose $M - n + 1 \leqslant k \leqslant M - 1$. This time, the n-windows of the sequences of $\mathcal{S}$ at positions congruent to k modulo M lie partly in a word from $\mathcal{C}^{(e-1)}$ and partly in a word from $\mathcal{C}^{(0)}$. Moreover, if the former codeword is $c_x^{(e-1)}$, say, then by the definitions used it follows that the latter codeword shall be $c^{(0)}_{f_{z,b}(x)}$. That is, for any x in the range $0 \leqslant x \leqslant z^n - 1$, letting i and j be defined as above we have that $[s^{(i)}]^n_{Mj+k} = [c_x^{(e-1)} \,|\, c^{(0)}_{f_{z,b}(x)}]^n_{k-M_{e-1}}$. The function on the right has the same form as the function $H_{k-M_{e-1},b}$ of Lemma 4.27, and the result follows. □

When $b = n$ then Construction 4.28 gives rise to an (n, M)-SDB sequence. Notice that if the codes $\mathcal{C}^{(i)}$ are chosen from the trivial code of every n-tuple over $\mathbb{Z}_z$ and the generalised parity check code consisting of every $(n+1)$-tuple whose coordinates sum to zero in $\mathbb{Z}_z$, with the former used for lower values of the index i than the latter, then this construction is identical to Construction 4.10. Thus we have presented a generalisation of Construction 4.10 allowing the use of a wider variety of codes and allowing the construction of a class of perfect multi-factors as well as spaced de Bruijn sequences. This presentation of Construction 4.10 gives us a more general viewpoint form which to understand that method.

Suppose we vary the value of b used in Construction 4.28, keeping all other parameters the same. Starting with $b = n$, we have seen that a spaced de Bruijn sequence arises—call this perfect multi-factor $\mathcal{S}_n$, say. When $b = n-1$ we obtain a set of z distinct sequences, $\mathcal{S}_{n-1} = \{[s^{(0)}], [s^{(1)}], \ldots, [s^{(z-1)}]\}$. Notice that by the construction method, if $w^{(i)}$ is the minimal generating cycle for $[s^{(i)}]$, then

the z^n-tuple $w^{(0)} \mid w^{(1)} \mid \cdots \mid w^{(z-1)}$ is the minimal generating cycle for the spaced de Bruijn sequence $\mathcal{S}_n$. This pattern continues: with $b = n - 2$ the perfect multi-factor $\mathcal{S}_{n-2}$ arising has the property that the minimal generating cycles for its sequences may be joined to give the minimal generating cycles for the sequences of $\mathcal{S}_{n-1}$. In other words, and in general, for all i in the range $n - 1 \leqslant i \leqslant 0$ the minimal generating cycles for the sequences in the perfect multi-factor $\mathcal{S}_i$ can be formed by dividing the minimal generating cycles for the sequences of $\mathcal{S}_{i+1}$ into z equal portions. All this is clear from the construction method. Perfect factors derived from these perfect multi-factors in the manner of Theorem 4.24 also inherit a similar property, whereby the cycles of one perfect factor may be divided or joined as above to form another. This is an unusual feature, and certainly not one that would be expected of perfect factors or perfect multi-factors in general.

When $b = 0$, $e = 1$ and $\mathcal{C}^{(0)}$ is the trivial code consisting of all n-tuples, then the perfect multi-factors arising from Construction 4.28 consist simply of all sequences arising from taking all n-tuples as generating cycles. This is identical to the perfect multi-factors of [39, Construction 3.5].

REMARK 4.30. We make the following observations regarding the existence of $\mathrm{X}(m, n, z)$ codes:

(1) for any z and any n there exist $\mathrm{X}(n, n, z)$ and $\mathrm{X}(n + 1, n, z)$ codes obtained by simply considering all n-tuples over $\mathbb{Z}_z$, and all $n+1$-tuples whose digits sum to 0 in $\mathbb{Z}_z$;

(2) if $z = p^r$, where p is prime and r a positive integer, and if $1 \leqslant n \leqslant z + 1$ then there exists an $\mathrm{X}(z + 1, n, z)$ code—this follows from the existence of

doubly-extended Reed-Solomon codes ([38, Chapter 11, §5]);

(3) puncturing any $i \leqslant m - n$ coordinate positions from all the words of an $\mathrm{X}(m, n, z)$ code formed from a Reed-Solomon code as described above gives $\mathrm{X}(m - i, n, z)$ code, since Reed-Solomon codes are systematic on *any* n coordinate positions;

(4) the M-tuples d_x of Construction 4.28 formed by concatenating codewords from $\mathrm{X}(m^{(i)}, n, z)$ codes themselves form a $\mathrm{X}(M, n, z)$ code, and

(5) in [39, Theorem 3.13] it is shown that $(n, 1, m, z)$-PMFs exist for all $m \geqslant n \geqslant 1$ and $z \geqslant 2$. By Remark 4.26, constructing these is equivalent to constructing $\mathrm{X}(m, n, z)$ codes.

□

Thus, as well as the general existence question for $\mathrm{X}(m, n, z)$ codes being solved, there are a number of construction methods available.

Earlier, we described how perfect factors and perfect multi-factors may be combined to form new perfect factors (Theorem 4.24). By analogy with our derivation of equation (4.1), we define the decoding problem for perfect factors and for perfect multi-factors as follows.

PROBLEM 4.31 (The decoding problem for perfect factors). Let $R = \{[r^{(0)}], [r^{(1)}], \ldots, [r^{(\ell-1)}]\}$ be an (n, m, y)-PF over $\mathcal{A}$. The decoding problem for R is to determine, given any $u \in \mathcal{A}^n$ the unique pair of integers k and f with $0 \leqslant k \leqslant (y^n/m) - 1$ and $0 \leqslant f \leqslant m - 1$ such that $[s^{(k)}]_f^n = u$. □

PROBLEM 4.32 (The decoding problem for perfect multi-factors). Let $S = \{[s^{(0)}], [s^{(1)}], \ldots, [s^{(\ell'-1)}]\}$ be an (n, t, m, z)-PMF over $\mathcal{B}$. The decoding problem

for S is to determine, given any $v \in \mathcal{B}^n$ and any f in the range $0 \leqslant f \leqslant m$ the unique pair of integers g and h with $0 \leqslant h \leqslant (z^n/t) - 1$ and $0 \leqslant g \leqslant t - 1$ such that $[s^{(h)}]^n_{gm+f} = v$. □

With the notation of Theorem 4.24 and of Problems 4.31 and 4.32 above, decoding the perfect factor $T = \{[r^{(i)}] \times [s^{(j)}] \,|\, 0 \leqslant i < \ell \text{ and } 0 \leqslant j < \ell'\}$ to find the position of an n-tuple $w = u \times v \in \mathcal{A} \times \mathcal{B}$ requires one solution of Problem 4.31 followed by one solution of Problem 4.32. Then w appears in $[r^{(k)}] \times [s^{(h)}]$ at position $gm + f$. Notice that Problems 4.31 and 4.32 conform to Problems 1.3 and 4.4 in the case $t = c^n$.

Consider the codes $\mathcal{C}^{(i)}$ used in Construction 4.28. Suppose it is easy to determine the rightmost n digits of any codeword (that is, the digits which were used to determine the ordering placed on the codewords) from any other n-tuple of consecutive digits. Call this calculation 'problem X', say. It is straightforward to construct a decoding algorithm for the perfect multi-factors arising from Construction 4.28, in a similar manner to Algorithm 4.16, provided that a means to solve problem X for the codes used is to hand. The modified algorithm would involve the computation of the inverses G^{-1} and H^{-1}, where previously the inverses $A^{-1}, B^{-1}, \ldots, F^{-1}$ were required. During this stage, n consecutive digits from a codeword c_x would be identified, as in the proof of Theorem 4.29. Solving problem X would give the last n digits of that codeword, to which the function $\psi^{-1}_{z,n}$ would be applied to yield x.

The ease with which we can solve problem X depends, of course, on the method used to construct the codes $\mathcal{C}^{(i)}$. The codes resulting in Case (5) of Remark 4.30 do not admit an obvious means of approaching problem X. For

those codes resulting from $[z+1, n]$ Reed-Solomon codes described in Case (2), it is clear that solving problem X is no more difficult than the problem of correcting $z+1-n$ erasures. Of course, problem X is trivial to solve for those codes arising in Case (1). These are the only codes used in Construction 4.10, and this feature is central to the efficiency of Algorithm 4.16.

This Section has concluded this Chapter by demonstrating how our method for constructing easily decoded de Bruijn sequences over composite alphabets generalises to a construction for easily-decoded Perfect Factors. We have shown how codes that are more general than those required previously may be employed. This gives us a deeper insight into the workings of Construction 4.10.

5 Position Location with Error Control

5.1 Introduction

In Section 1.1 we discussed how sensor read errors could hamper the performance of position location schemes based on certain window sequences, such as de Bruijn sequences. To counter this, we suggested using a sequence whose windows differed from each other in at least a given number of places δ. From the Coding Theory results presented in Section 1.4, such a system would be able to detect up to $\delta - 1$ (or correct up to $\lfloor \frac{\delta-1}{2} \rfloor$) errors in the input. However, the maximum possible period of such a sequence is clearly less than that of a span n de Bruijn sequence. Thus the price for this error control capability is the increase in the span n required to obtain a sequence of a given period. This corresponds to the cost of the extra sensors required to maintain the resolution of the position location system. Alternatively, we may view the cost as a reduction in system resolution given a fixed number of sensors. To maximise efficiency, we are interested in obtaining, for a given n and δ, sequences as described above that have as large a least period as possible.

Kumar and Wei ([30]) were the first authors to propose the use of error correction in these position location applications, although they only consider using m-sequences. They gave counting arguments to provide a bound on the window size n required to ensure the existence of a primitive polynomial giving rise to an m-sequence having at least a given minimum distance between its n-windows (or, equivalently, a minimum weight n-window of at least a given weight). However, this result does not provide any information to assist in determining *which*

primitive polynomials give rise to the m-sequences with the properties they describe. Bartee and Wood ([4]) searched for those binary m-sequences whose minimum weight n-window has greatest weight, but were "unable to find any systematic relationship between the set of optimum m-sequences and their generating functions" ([34], see also [60]).

Rather than focusing on the distance properties of windows of m-sequences we take an alternative approach, asking instead when some or all of the words of a code with a known minimum distance may be 'overlapped' to form a sequence. We exhibit a one-to-one correspondence between sequences formed by overlapping words from linear codes and certain families of linear recurring sequences. Through this we consider the relationship of the minimal polynomial of a linear recurring sequence to the minimum distance between the windows of that sequence. By applying our results to m-sequences, we obtain a solution to the problem of Bartee and Wood, characterising the precise minimum distance between windows of an m-sequence in terms of the minimal polynomial (Corollary 5.26).

We have also performed computer searches for optimal sequences with a given minimum distance δ between their n-windows. In the case $\delta = 2$, analysis of the results leads to the establishment of simple, specific properties of the coefficients of primitive polynomials that guarantee the existence of m-sequences and related sequences with a minimum distance of 2 between their n-windows. These sequences have large period relative to n. We have produced strong computational evidence regarding the existence of the requisite polynomials.

We go on to give a brief discussion of how error detection or correction may

be implemented for these sequences, and give a combinatorial construction for an infinite class of such sequences. Finally, we use some of this work to present a construction for arrays whose windows have an analogous distance property.

5.2 Linear Recurrences

We formally define our main objects of study as follows.

Definition 5.1. Let n and e be positive integers, with $e > n$, and let $[s]$ be a sequence with least period e. We define the *(minimum) distance* δ of $[s]$ with respect to the span n by

$$\delta = \min_{0 \leqslant i < j < e} \mathrm{d}([s]_i^n, [s]_j^n).$$

If $[s]$ has distance δ with respect to n we say that $[s]$ is an (n, δ) *code window sequence*, hereafter denoted an (n, δ)-CW sequence. □

Clearly, $n \geqslant \delta$. These sequences may be considered in graph theoretic terms. For some fixed span $n \geqslant 1$ and alphabet size $b \geqslant 2$, let G be the graph $\mathcal{H}_{b,n}$ described in Section 1.4, with the edges of this graph coloured in, say, red. Notice that the de Bruijn graph $\mathcal{G}_{b,n}$ described in Section 1.2 has the same vertex set, so we may superimpose on G the edges of $\mathcal{G}_{b,n}$ with these new edges in, say, green. (There may be some edges that are both red and green, we allow this.) Then equivalent to a b-ary (n, δ)-CW sequence is a sequence of distinct vertices $v_0, v_1, \ldots, v_{e-1}$ in G, which form a cycle with respect to the green edges, and no two of which are joined by a path of red edges of length less than δ.

Let $L_b(n, \delta)$ denote the maximum length of a b-ary (n, δ)-CW sequence. A b-ary (n, δ)-CW sequence with least period equal to $L_b(n, \delta)$ is said to be *optimal*.

When $\delta = 1$ this coincides with the definition of a de Bruijn sequence. Thus (n, δ)-CW sequences are also of interest purely as combinatorial objects, being natural generalisations of this important and well studied class of sequences.

As usual, we write $A_b(n, \delta)$ to denote the maximum number of words in a b-ary code of length n with minimum distance δ. The following bounds are immediate from the above discussion.

LEMMA 5.2. *For all integers $b \geqslant 2$ and $n, \delta \geqslant 1$, we have*

$$L_b(n, \delta) \leqslant A_b(n, \delta) \quad \textit{and} \tag{5.1}$$

$$L_b(n, 1) = b^n. \tag{5.2}$$

We now formalise the concept of 'overlapping' codewords to form a sequence.

DEFINITION 5.3. Let n and e be positive integers, with $e > n$. A sequence of n-tuples $c_0, c_1, \ldots, c_{e-1}$, written in coordinates as $c_i = (c_i^0, c_i^1, \ldots, c_i^{n-1})$ for each $0 \leqslant i \leqslant e-1$, are said to be *overlapping* if and only if

$$c_{i+1 \bmod e}^j = c_i^{j+1} \quad \text{for all } i = 0, 1, \ldots, e-1 \text{ and } j = 0, 1, \ldots, n-2.$$

A code $\mathcal{C}$ of length n which admits a sequence of distinct overlapping words $c_0, c_1, \ldots, c_{e-1}$ for some $e > n$ is said to have the *sequencing property* (hereafter the S-property). We also say that $\mathcal{C}$ has the S-*property for* $c_0, c_1, \ldots, c_{e-1}$ *and* e in the event that the c_i and e are to be specified. □

Clearly, q-ary codewords $c_0, c_1, \ldots, c_{e-1}$ are overlapping if and only if, considered as vertices in the de Bruijn graph $\mathcal{G}_{q,n}$, they satisfy $c_i \rightarrowtail c_{i+1 \bmod e}$ for all i in the range $0 \leqslant i \leqslant e-1$. Thus sequences of overlapping codewords correspond to cycles in the de Bruijn graph.

The sequence $[s]$ given by $s_i = c^0_{i \bmod e}$ has $[s]^n_i = c_{i \bmod e}$ for all $i \in \mathbb{Z}$. Thus if $\mathcal{C}$ has minimum distance d then $[s]$ is a (n, δ)-CW sequence for some $\delta \geqslant d$. It is possible that the codewords used form a sub-code of $\mathcal{C}$ with distance $\delta > d$. Since any (n, δ)-CW sequence can be thought of as being formed in this way, the study of (n, δ)-CW sequences is equivalent to the study of codes $\mathcal{C}$ with the S-property.

LEMMA 5.4. *Let n and e be positive integers, with $e > n$, and let $[s]$ be a sequence over $\mathbb{F}_q$ of least period e. Let $V \subseteq \mathbb{F}_q^n$ denote the linear span of the set of n-windows of $[s]$. Recall that $\mathrm{lc}[s]$ denotes the linear complexity of $[s]$. Then the following two statements are equivalent:*

(1) $\dim V = k < n$;

(2) $\mathrm{lc}[s] = k < n$.

PROOF. That (1) implies (2) and that (2) implies (1) both follow by applying Theorem 1.4. □

Clearly, under the conditions of Lemma 5.4, V is the smallest linear code with the S-property for $c_0, c_1, \ldots, c_{e-1}$ and e. Thus this result shows that for $k < n$, sequences of least period $e > n$ obtained by overlapping a spanning set of words taken from an $[n, k]$ code are equivalent to sequences of linear complexity k and order e.

We now consider the restriction '$k < n$' above—the following Examples relax this condition.

EXAMPLE 5.5. Let n be a positive integer and let $[s]$ be a span n de Bruijn sequence over $\mathbb{F}_q$. Then $[s]$ is a $(n, 1)$-CW sequence, and the n-windows of $[s]$

can be considered as coming from a $[n, n]$ code. As all n-windows appear, the dimension of the space spanned by the n-windows is n. However, it has been shown ([5]) that the linear complexity $\mathrm{lc}[s]$ of $[s]$ satisfies

$$q^{n-1} + n \leqslant \mathrm{lc}[s] \leqslant q^n - 1$$

(unless $q = 2$ and $n = 1$ or 2, in which case $\mathrm{lc}[s] = 2$ or 3 respectively). □

Thus maximal sequences from an $[n, n, 1]$ code never have linear complexity equal to the dimension of the space spanned by their n-windows. This demonstrates that the condition '$k < n$' is not redundant. Example 5.6 below considers the same condition, and shows that whilst being sufficient to determine the linear complexity of the sequence, it is not always necessary.

EXAMPLE 5.6. Let $[s]$ be an m-sequence with minimal polynomial of degree n. Then $[s]$ is a $(n, 1)$-CW sequence and does have linear complexity equal to the dimension of the space spanned by its n-windows, namely n. □

REMARK 5.7. In the event that $\mathcal{C}$ is linear, conditioning $\mathcal{C}$ to have distance at least 2 is sufficient to ensure that $k < n$. □

Good sequences may be formed by overlapping words from non-linear codes, as demonstrated by the following example.

EXAMPLE 5.8. We consider the existence of $(5, 2)$-CW sequences over $\mathbb{F}_2$. Suppose $[s]$ is such a sequence, formed by overlapping codewords from a linear code with minimum distance $\geqslant 2$. Then the n-windows of $[s]$ must span a space of dimension $k \leqslant 4$. Three cases apply.

CASE 1: $k = 4$. The only $[5, 4, 2]$ code over $\mathbb{F}_2$ is the parity check code, consisting

of all codewords of even weight. Thus all 5-windows of $[s]$ are of even weight, and so we deduce that $[s]$ must have period at most $n = 5$.

CASE 2: $k = 3$. Consider $f(x) = x^3 + x + 1$, a primitive polynomial of degree 3 over $\mathbb{F}_2$. Let $[s]$ have minimal polynomial $f(x)$, so $[s]$ is defined (up to cyclic shifting) by the generating cycle

$$[s] = [0010111].$$

Then by Theorem 1.4, the 5-windows of $[s]$ span a space of dimension 3. It is simple to confirm that these windows (and the code they span) have minimum distance equal to 2. Thus $[s]$ is a $(5, 2)$-CW sequence of least period 7. By Remark 5.12, to follow, this period is optimal for the case $k = 3$.

CASE 3: $k < 3$. Any space of dimension at most 2 has at most $2^2 = 4$ codewords. Thus in these cases the sequence $[s]$ has period at most 4.

Thus we have shown that when using a linear code to form a sequence $[s]$ with design distance 2 with respect to 5, the greatest possible period for $[s]$ is 7.

Now consider the sequence $[t]$ defined by the generating cycle

$$[t] = [0001011101]$$

of period 10. It is easily confirmed that this sequence is a $(5, 2)$-CW sequence with minimal polynomial $f(x) = x^6 + x^5 + x + 1$. □

The sequence $[t]$ in Example 5.8 is in fact an optimal $(5, 2)$-CW sequence over $\mathbb{F}_2$, as found by computer search. The results of the computer searches are given in Appendix A, and will be discussed further in Section 5.5. (This particular sequence may be found as the sequence with Ref. 6 in Table A.2.)

We have shown that any (n, δ)-CW sequence formed by overlapping words taken from a code $\mathcal{C}$ with the S-property falls into one of two categories: those with linear complexity $k < n$, for which $\mathcal{C}$ is a $[n, k]$ code, and those of complexity at least n. Given a particular sequence $[s]$, it may be useful to determine which of the above cases applies. For example, this knowledge may be helpful in designing error detecting or correcting procedures for n-windows of the sequence (see Section 5.7). By Lemma 5.4, these cases may be distinguished by determining the linear complexity of $[s]$. This is easily calculated, for example by the use of the Berlekamp-Massey Algorithm ([35]).

We shall concentrate on the more tractable case of linear codes. We will thus find the following definitions useful.

DEFINITION 5.9. We say that a code $\mathcal{C}$ has the T-*property* if and only if the following hold:

(1) $\mathcal{C}$ is an $[n, k]$ code for some integers n and k with $n > k$, and

(2) $\mathcal{C}$ has the S-property for some $c_0, c_1, \ldots, c_{e-1} \in \mathcal{C}$ and some $e > n$, with $\langle c_0, c_1, \ldots, c_{e-1} \rangle = \mathcal{C}$.

We say $\mathcal{C}$ has the T-*property for* $c_0, c_1, \ldots, c_{e-1}$ *and* e in the event that these are to be specified. □

Let n be any positive integer. If $[s]$ is a sequence with $\mathrm{lc}[s] = k < n$ then we have seen that the n-windows of $[s]$ may be thought of as spanning an $[n, k]$ code $\mathcal{C}$ with the T-property, and that $[s]$ is a (n, δ)-CW sequence for some δ greater than or equal to the minimum distance d of $\mathcal{C}$. We are motivated to borrow the following terminology: we refer to the minimum distance d of $\mathcal{C}$ as the *design*

distance of $[s]$ *with respect to* n.

LEMMA 5.10. *Let* $\mathcal{C}$ *be an* $[n,k]$ *code with the* T*-property, and let* $[s]$ *be any sequence formed by overlapping a spanning set of at least* $n+1$ *distinct codewords from* $\mathcal{C}$. *Then the minimal polynomial of* $[s]$ *is uniquely determined by* $\mathcal{C}$, *independently of the particular codewords chosen.*

PROOF. Let $f(x) = x^k + f_{k-1}x^{k-1} + \cdots + f_0$ be the minimal polynomial of such a sequence $[s]$, and define $\mathbf{f} = (1, f_{k-1}, f_{k-2}, \ldots, f_0) \in \mathbb{F}_q^{k+1}$. Define $\mathcal{F}$ to be the one-dimensional subspace of $\mathbb{F}_q^{k+1}$ given by all constant multiples of $\mathbf{f}$, and let $\mathcal{A}$ be the space spanned by the $(k+1)$-windows of $[s]$. Since a spanning subset of codewords was used to form $[s]$, Lemma 5.4 shows that $\mathrm{lc}[s] = k$. Then, by Theorem 1.4, $\mathcal{A}$ has dimension k. Also, since $f(x)$ is the minimal polynomial for $[s]$, we deduce that $\mathcal{A} = \mathcal{F}^{\perp}$.

Now let $[s']$ be any other such sequence, and define $f'(x), \mathbf{f}', \mathcal{F}'$ and $\mathcal{A}'$ analogously to the above, noting $\mathcal{A}' = \mathcal{F}'^{\perp}$. We observe that $\mathcal{A} = \mathcal{A}'$, since each is equal to the k-dimensional space obtained by projecting $\mathcal{C}$ onto its first $k+1$ coordinates. Thus $\mathcal{F} = \mathcal{F}'$ and so $f(x) = f'(x)$ as required. □

This result motivates the following definition:

DEFINITION 5.11. Let n and k be positive integers, $n > k$, and let $\mathcal{C}$ be an $[n,k]$ code with the T-property. We denote by $f_{\mathcal{C}}(x)$ the unique minimal polynomial of the sequences obtained by overlapping a spanning subset of codewords of $\mathcal{C}$.

□

We observe that since $f_{\mathcal{C}}(x)$ is the minimal polynomial of a periodic sequence, $x \nmid f_{\mathcal{C}}(x)$.

REMARK 5.12. Suppose $[s]$ is an (n, δ)-CW sequence with $\delta \geqslant 2$ formed by overlapping words taken from a linear code $\mathcal{C}$. Then it is never possible to use all the words of $\mathcal{C}$ in $[s]$. For, in particular, the all zero codeword cannot appear in $[s]$ at it would necessarily be followed in $[s]$ by an n-window of Hamming weight $1 < \delta$. □

REMARK 5.13. Suppose again that $[s]$ is an (n, δ)-CW sequence formed by overlapping words taken from an $[n, k, d]$ code $\mathcal{C}$, but that unlike the sequence considered in Lemma 5.10, the codewords used do not span $\mathcal{C}$. Let $\mathcal{C}'$ be the space spanned by the codewords used. Then $\mathcal{C}'$ is a $[n, k', d']$ code with $k' < k$ and $d' \geqslant d$. We may consider $[s]$ as being formed by overlapping words from the code $\mathcal{C}'$, and may then apply Lemmas 5.4 and 5.10. □

5.3 Which Linear Codes Form Sequences?

In this section we study the structure of the generator and parity check matrices of those codes having the T-property.

LEMMA 5.14. *Let n and k be integers, $n > k$, and let $\mathcal{C}$ be an $[n, k]$ code with the T-property. Let $\mathbf{G} = \left(\begin{array}{c|c|c|c} \mathbf{g}_0^\top & \mathbf{g}_1^\top & \cdots & \mathbf{g}_{n-1}^\top \end{array} \right)$ be any generator matrix for $\mathcal{C}$. Then the $k \times k$ matrix $\mathbf{A} = \left(\begin{array}{c|c|c|c} \mathbf{g}_0^\top & \mathbf{g}_1^\top & \cdots & \mathbf{g}_{k-1}^\top \end{array} \right)$ is non-singular.*

PROOF. By the T-property, there exist codewords $\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{e-1}$ satisfying the overlapping condition. For each i in the range $0 \leqslant i \leqslant e-1$, let $\mathbf{m}_i \in \mathbb{F}_q^k$ be the unique message corresponding to the codeword $\mathbf{c}_i$, so $\mathbf{c}_i = \mathbf{m}_i \mathbf{G}$, and define $\mathbf{d}_i \in \mathbb{F}_q^k$ to be the projection of $\mathbf{c}_i$ onto its first k coordinates, so

$$\mathbf{d}_i = \mathbf{m}_i \mathbf{A}. \tag{5.3}$$

Notice that the sequence $[s]$ arising from the codewords $\mathbf{c}_i$ has $[s]_i^k = \mathbf{d}_i$ and that by Lemma 5.4 we have $\mathrm{lc}[s] = k$. Then by Theorem 1.4 the $\mathbf{d}_i$ span a space of dimension k. Thus the matrix A must have full rank. □

LEMMA 5.15. *Let* $\mathcal{C}, [s], \mathbf{G}, \mathbf{A}, \mathbf{d}_i, \mathbf{c}_i$ *and* $\mathbf{m}_i$ *be as in Lemma 5.14, and let* $f_{\mathcal{C}}(x)$ *be as in Definition 5.11. Then*

(1) $\mathbf{m}_{i \bmod e} = \mathbf{m}_0 \mathbf{R}^i$ *for all* $i \in \mathbb{Z}$, *where* $\mathbf{R}$ *is a* $k \times k$ *non-singular cyclic matrix with minimal polynomial* $f_{\mathcal{C}}(x)$;

(2) $\mathbf{m}_0$ *is a cyclic vector for* $\mathbf{R}$, *and*

(3) $s_i = \mathbf{m}_0 \mathbf{R}^i \mathbf{g}_0^\top$ *for all* $i \in \mathbb{Z}$.

PROOF. For part (1), let $\mathbf{F}_{\mathcal{C}}$ be the companion matrix for $f_{\mathcal{C}}(x)$. Following equation (1.2) we have that

$$\mathbf{d}_{i+1 \bmod e} = \mathbf{d}_i \mathbf{F}_{\mathcal{C}}$$

for $i = 0, 1, \ldots, e-1$. Using equation (5.3) and Lemma 5.14 we have that for all $i = 0, 1, \ldots, e-1$

$$\begin{aligned} \mathbf{m}_{i+1 \bmod e} &= \mathbf{d}_{i+1 \bmod e} \mathbf{A}^{-1} \\ &= \mathbf{d}_i \mathbf{F}_{\mathcal{C}} \mathbf{A}^{-1} \\ &= \mathbf{m}_i \mathbf{A} \mathbf{F}_{\mathcal{C}} \mathbf{A}^{-1} \\ &= \mathbf{m}_i \mathbf{R} \end{aligned}$$

where $\mathbf{R} = \mathbf{A} \mathbf{F}_{\mathcal{C}} \mathbf{A}^{-1}$ is conjugate to $\mathbf{F}_{\mathcal{C}}$ and so has minimal polynomial $f_{\mathcal{C}}(x)$. Since $f_{\mathcal{C}}(x)$ has degree k we have that $\mathbf{R}$ is cyclic. Note that $\mathbf{R}$ is non-singular if (and only if) $\mathbf{F}_{\mathcal{C}}$ is non-singular, which is true since $x \nmid f_{\mathcal{C}}(x)$. Also, the order of $\mathbf{R}$ is equal to that of $f_{\mathcal{C}}(x)$, namely e. By induction on i, we have

$\mathbf{m}_{i \bmod e} = \mathbf{m}_0 \mathbf{R}^i$ for all $i \in \mathbb{Z}$, as required. Since the $\mathbf{c}_i$ span $\mathcal{C}$, the $\mathbf{m}_i$ must span $\mathbb{F}_q^k$, and so $\mathbf{m}_0$ is a cyclic vector for $\mathbf{R}$. This establishes part (2).

Part (3) follows since $s_i = c_i^0 = \mathbf{m}_i \mathbf{g}_0^\top = \mathbf{m}_0 \mathbf{R}^i \mathbf{g}_0^\top$. □

Let $\mathcal{F}_{n,k}$ denote the set of all monic polynomials $p(x)$ of degree k with $x \nmid p(x)$ and $\mathrm{ord}(p) > n$. Given $p(x) \in \mathcal{F}_{n,k}$, we denote by $\mathcal{R}_p$ the set of all pairs $(\mathbf{R}, \mathbf{x})$ where $\mathbf{R}$ is a $k \times k$ matrix with minimal polynomial $p(x)$ (equivalently, that $\mathbf{R}$ is conjugate to the companion matrix $\mathbf{P}$ of $p(x)$), and where $\mathbf{x}$ is a cyclic vector for $\mathbf{R}^\top$.

LEMMA 5.16. *Let n and k be integers, $n > k$, and let $\mathcal{C}$ be an $[n, k]$ code. Let $p(x)$ be any element of $\mathcal{F}_{n,k}$. The following two statements are equivalent.*

(1) $\mathcal{C}$ has the T*-property with $f_{\mathcal{C}}(x) = p(x)$.*

(2) Any generator matrix $\mathbf{G} = \left(\mathbf{g}_0^\top \,\middle|\, \mathbf{g}_1^\top \,\middle|\, \cdots \,\middle|\, \mathbf{g}_{n-1}^\top \right)$ for $\mathcal{C}$ satisfies $\mathbf{g}_i^\top = \mathbf{R}^i \mathbf{x}^\top$ for some $(\mathbf{R}, \mathbf{x}) \in \mathcal{R}_p$.

PROOF. Suppose (1) holds, and let $\mathbf{G}$ be any generator matrix for $\mathcal{C}$. Let $\mathbf{R}$, $[s]$ and $\mathbf{m}_i$ be defined as in Lemma 5.15. For all $i \in \mathbb{Z}$ and all j in the range $0 \leqslant j \leqslant n-1$ we have $\mathbf{m}_i \mathbf{g}_j = s_{i+j} = \mathbf{m}_i \mathbf{R}^j \mathbf{g}_0^\top$. The $\mathbf{m}_i$ span $\mathbb{F}_q^k$, or else their corresponding codewords could not span a space of dimension k, contradicting the T-property. Fixing j and allowing i to vary, we must have $\mathbf{g}_j^\top = \mathbf{R}^j \mathbf{g}_0^\top$. This holds for all values of j.

Since G is a generator matrix for $\mathcal{C}$ it must have rank k. Thus the $\mathbf{g}_i$ span $\mathbb{F}_q^k$, and so $\mathbf{g}_0$ is a cyclic vector for $\mathbf{R}^\top$. The definition of $\mathbf{R}$ ensures that it has minimal polynomial $p(x)$. Thus $(\mathbf{R}, \mathbf{g}_0) \in \mathcal{R}_p$, establishing (2).

Now suppose (2) holds. That is, let $\mathbf{G}$ be a generator for $\mathcal{C}$ and suppose that the columns of $\mathbf{G}$ satisfy $\mathbf{g}_i^\top = \mathbf{R}^i\mathbf{x}^\top$ for some $(\mathbf{R}, \mathbf{x}) \in \mathcal{R}_p$. Since $\mathbf{R}$ is cyclic, let $\mathbf{m}$ be a cyclic vector for $\mathbf{R}$, and define the sequence $[s] = (s_i)_{i\in\mathbb{Z}}$ by $s_i = \mathbf{m}\mathbf{R}^i\mathbf{x}^\top$ for all $i \in \mathbb{Z}$. For any i and ℓ in $\mathbb{Z}$ with $0 \leqslant \ell \leqslant n-1$ we have $s_{i+\ell} = \mathbf{m}\mathbf{R}^{i+\ell}\mathbf{x}^\top = \mathbf{m}\mathbf{R}^i\mathbf{g}_\ell$, and so $[s]_i^n = \mathbf{m}\mathbf{R}^i\mathbf{G}$. Thus the n-windows of $[s]$ each lie in $\mathcal{C}$. Moreover, since $\mathbf{m}$ is a cyclic vector for $\mathbf{R}$, the vectors given by $\mathbf{m}\mathbf{R}^i$ span $\mathbb{F}_q^k$, and so the n-windows of $[s]$ span $\mathcal{C}$. The consecutive n-windows of $[s]$ are a sequence of codewords spanning $\mathcal{C}$ and satisfying the overlapping property, and so $\mathcal{C}$ has the T-property. Finally, by Theorem 1.6 $[s]$ has minimal polynomial $f_{\mathcal{C}}(x) = p(x)$ so (1) is established. □

We have shown that a code $\mathcal{C}$ has the T-property if and only if every generator matrix for $\mathcal{C}$ is of a particular form. The next result characterises this relationship precisely. For any linear code $\mathcal{C}$, let $\mathcal{G}_{\mathcal{C}}$ denote the set of all possible generator matrices for $\mathcal{C}$.

THEOREM 5.17. *Let n and k be integers, $n > k$, and let $\mathcal{C}$ be an $[n,k]$ code with the T-property. Let $p(x) = f_{\mathcal{C}}(x)$, and let $\mathbf{P}$ be the companion matrix for $p(x)$. Given $\mathbf{G} = \left(\mathbf{g}_0^\top \,\middle|\, \mathbf{g}_1^\top \,\middle|\, \cdots \,\middle|\, \mathbf{g}_{n-1}^\top \right) \in \mathcal{G}_{\mathcal{C}}$, let $\mathbf{R} = \mathbf{A}\mathbf{P}\mathbf{A}^{-1}$, with $\mathbf{A} = \left(\mathbf{g}_0^\top \,\middle|\, \mathbf{g}_1^\top \,\middle|\, \cdots \,\middle|\, \mathbf{g}_{k-1}^\top \right)$, as in Lemma 5.14. Then the map $\phi : \mathbf{G} \mapsto (\mathbf{R}, \mathbf{g}_0)$ is a bijection from $\mathcal{G}_{\mathcal{C}}$ to $\mathcal{R}_p$.*

PROOF. Fix $\mathbf{G} \in \mathcal{G}$, and let $\phi(\mathbf{G}) = (\mathbf{R}, \mathbf{x})$. By Lemmas 5.10 and 5.14, the matrix $\mathbf{R}$ is uniquely defined and is conjugate to $\mathbf{P}$. By the proof of Theorem 5.16, $\mathbf{g}_0$ is cyclic for $\mathbf{R}^\top$ and so ϕ is a well defined function from $\mathcal{G}$ to $\mathcal{R}_p$.

We exhibit an inverse to ϕ having domain $\mathcal{R}_p$. For any $(\mathbf{T}, \mathbf{y}) \in \mathcal{R}_p$ define

the function $\psi : (\mathbf{T}, \mathbf{y}) \mapsto \mathbf{J} = \left(\mathbf{j}_0^\top \,\middle|\, \mathbf{j}_1^\top \,\middle|\, \cdots \,\middle|\, \mathbf{j}_{n-1}^\top \right)$, where the $\mathbf{j}_i$ are as given by $\mathbf{j}_i = \mathbf{y}(\mathbf{T}^\top)^i$. Clearly, ψ is well defined on $\mathcal{R}_p$.

Let $\mathbf{m}$ be a cyclic vector for $\mathbf{T}$, and define the sequence $[s]$ by $s_i = \mathbf{m}\mathbf{T}^i\mathbf{y}^\top$. Then defining $\mathbf{m}_i = \mathbf{m}\mathbf{T}^i$ for all $i \in \mathbb{Z}$, we have $[s]_i^n = \mathbf{m}_i\mathbf{J}$. By Theorem 1.6, $[s]$ has minimal polynomial $p(x)$ and so by Corollary 1.5 lies in the vector space V of sequences with minimal polynomial $p(x)$. Now, any sequence $[t]$ formed by overlapping a spanning subset of words from $\mathcal{C}$ also has minimal polynomial $p(x)$, and so also lies in V. Also, all shifts of either of the sequences $[s]$ and $[t]$ lie in V, and in fact span V. Thus the n-windows of $[s]$ must span the space generated by the n-windows of $[t]$. Therefore $\mathbf{J}$ is a generator matrix for $\mathcal{C}$, and ψ is a map from $\mathcal{R}_p$ to $\mathcal{G}$.

It remains to show that ψ is the inverse of ϕ. Fix $\mathbf{G} \in \mathcal{G}$, let $\phi(\mathbf{G}) = (\mathbf{R}, \mathbf{x})$ and let $\mathbf{J} = \psi(\mathbf{R}, \mathbf{x})$. By the definitions, $\mathbf{g}_0 = \mathbf{j}_0$. Define $[s]$ by $s_i = \mathbf{m}_i\mathbf{g}_0^\top$, where $\mathbf{m}$ is a cyclic vector for $\mathbf{R}$ and $\mathbf{m}_i = \mathbf{m}\mathbf{R}^i$ for all $i \in \mathbb{Z}$. Then for all $i \in \mathbb{Z}$ and all $0 \leqslant j < n$ we have

$$\begin{aligned} \mathbf{m}_i\mathbf{g}_j^\top &= \mathbf{m}_i\mathbf{R}^j\mathbf{g}_0^\top \\ &= \mathbf{m}_i\mathbf{R}^j\mathbf{j}_0^\top \\ &= \mathbf{m}_i\mathbf{j}_j^\top. \end{aligned}$$

Thus $\mathbf{m}_i\mathbf{G} = \mathbf{m}_i\mathbf{J}$, and since the $\mathbf{m}_i$ span $\mathbb{F}_q^k$ we deduce $\mathbf{G} = \mathbf{J}$ as required. □

The following result gives an alternative expression for $f_{\mathcal{C}}(x)$ and the matrix $\mathbf{R}$ of Lemma 5.15.

COROLLARY 5.18. *Let n and k be positive integers with $n > k$, and let $\mathcal{C}$ be an $[n, k]$ code with the* T*-property. Let $\mathbf{G}$, $\mathbf{A}$ and $\mathbf{R}$ be as defined in Lemma 5.15 and*

let $\mathbf{B} = \left(\begin{array}{c|c|c|c} \mathbf{g}_1^\top & \mathbf{g}_2^\top & \cdots & \mathbf{g}_k^\top \end{array} \right)$. *Recall that* $\mathbf{F}_{\mathcal{C}}$ *denotes the companion matrix of* $f_{\mathcal{C}}(x)$. *Then we have that:*

(1) $\mathbf{R} = \mathbf{B}\mathbf{A}^{-1}$, *and*

(2) $\mathbf{F}_{\mathcal{C}} = \mathbf{A}^{-1}\mathbf{B}$.

PROOF. By Lemma 5.16, $\mathbf{g}_{i+1}^\top = \mathbf{R}\mathbf{g}_i^\top$, so $\mathbf{B} = \mathbf{R}\mathbf{A}$. Thus $\mathbf{R} = \mathbf{B}\mathbf{A}^{-1}$ and $\mathbf{F}_{\mathcal{C}} = \mathbf{A}^{-1}\mathbf{R}\mathbf{A} = \mathbf{A}^{-1}\mathbf{B}$. □

If $\mathcal{C}$ has the T-property then the second part of Corollary 5.18 gives an easy method of computing $f_{\mathcal{C}}(x)$ from $\mathbf{G}$: the coefficients of $f_{\mathcal{C}}(x) = x^k + f_{k-1}x^{k-1} + \cdots + f_0$ will appear as the coefficients of the vector $\mathbf{A}^{-1}\mathbf{g}_k^\top = (f_0, f_1, \ldots, f_{k-1})^\top$.

We now shift our focus to concentrate on the parity check matrix $\mathbf{H}$ of $\mathcal{C}$. We will find the following notation useful. We have already defined $f_{\mathcal{C}}(x) = x^k + f_{k-1}x^{k-1} + \cdots + f_0$. We extend this to give a sequence $[f_{\mathcal{C}}] = (f_i)_{i\in\mathbb{Z}}$ by defining $f_k = 1$ and $f_i = 0$ for all $i < 0$ and $i > k$.

LEMMA 5.19. *The* $(n-k) \times n$ *matrix* $\mathbf{H} = (h_{ij})$ *given by* $h_{ij} = f_{j-i}$ *for all* $0 \leqslant i < n-k$ *and* $0 \leqslant j < n$ *is a parity check matrix for* $\mathcal{C}$.

PROOF. The matrix $\mathbf{H}$ may be written explicitly as

$$\begin{pmatrix} f_0 & f_1 & \cdots & f_{n-k-2} & f_{n-k-1} & \cdots & f_k & 0 & \cdots & 0 & 0 \\ 0 & f_0 & \cdots & f_{n-k-3} & f_{n-k-2} & \cdots & f_{k-1} & f_k & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & f_0 & f_1 & \cdots & f_{2k-n+2} & f_{2k-n+3} & \cdots & f_k & 0 \\ 0 & 0 & \cdots & 0 & f_0 & \cdots & f_{2k-n+1} & f_{2k-n+2} & \cdots & f_{k-1} & f_k \end{pmatrix}. \quad (5.4)$$

The ith row of $\mathbf{H}$ consists of the n-tuple $[f]_{-i}^n$, which includes all the non-zero terms of $[f_{\mathcal{C}}]$. Since $f_{\mathcal{C}}(x)$ is the minimal polynomial of $[s]$, and since the n-windows of $[s]$ span $\mathcal{C}$, each row of $\mathbf{H}$ is therefore orthogonal to $\mathcal{C}$.

By the structure of $[f_{\mathcal{C}}]$ and $\mathbf{H}$, the ith row of $\mathbf{H}$ has its first non-zero term in the ith column. Thus $\mathbf{H}$ has rank $n-k$ and must span $\mathcal{C}^{\perp}$. □

The matrix $\mathbf{H}$ is similar to the usual form of the parity check matrix for any cyclic code with generator polynomial $f(x) = f_k x^k + f_{k-1}x^{k-1} + \cdots + f_0$. The difference is that cyclic codes require that $f(x) \,|\, x^n - 1$ whereas we require that $e = \mathrm{ord}(f(x)) > n$. In this sense $\mathcal{C}$ can be viewed as a cyclic code truncated (by puncturing) from length e to length n.

We are now in a position to strengthen Lemma 5.10. Let $\mathcal{K}_{n,k}$ be the set of all $[n,k]$ codes with the T-property. The following result characterises the relationship between linear codes with the T-property and linear recurring sequences.

THEOREM 5.20. *Let n and k be integers, $n > k$. The function $\theta : \mathcal{C} \mapsto f_{\mathcal{C}}(x)$ for all $\mathcal{C} \in \mathcal{K}_{n,k}$ is a bijection from $\mathcal{K}_{n,k}$ to $\mathcal{F}_{n,k}$.*

PROOF. By Lemma 5.10, θ is well defined on $\mathcal{K}_{n,k}$. Given $\mathcal{C} \in \mathcal{K}_{n,k}$, if $p(x) = \theta(\mathcal{C}) = f_{\mathcal{C}}(x)$, then by Lemma 5.19 $p(x)$ uniquely determines $\mathcal{C}$. Thus θ is injective.

We demonstrate that any $p(x) \in \mathcal{F}_{n,k}$ determines a code $\mathcal{C} \in \mathcal{K}_{n,k}$ with $\theta(\mathcal{C}) = p(x)$. This implies that θ is surjective. The method of Lemma 5.19 may be used to construct a $(n-k) \times k$ matrix $\mathbf{H}$ with rank $n-k$, given any $p(x) \in \mathcal{F}_{n,k}$. We claim that the $[n,k]$ code $\mathcal{C}$ with parity check matrix $\mathbf{H}$ has the T-property—this would establish the required result.

To see that the claim is true, let $\mathbf{P}$ be the companion matrix for $p(x)$. Notice that $\mathbf{x} = (1,0,0,\ldots,0)$ is a cyclic vector for $\mathbf{P}^{\top}$, and that $\mathbf{m} = (0,0,\ldots,0,1)$ is

a cyclic vector for $\mathbf{P}$. By Theorem 1.6 the sequence $[s] = (s_i)_{i\in\mathbb{Z}}$ given by $s_i = \mathbf{m}\mathbf{P}^i\mathbf{x}^\top$ has minimal polynomial $p(x)$ and period $\mathrm{ord}(p) > n$. The consecutive n-windows from a period of $[s]$ form a sequence of words with the overlapping property. These words are orthogonal to the rows of $\mathbf{H}$, and so lie in $\mathcal{C}$. By Theorem 1.4 the n-windows of $[s]$ span a k dimensional space, namely $\mathcal{C}$. Thus $\mathcal{C}$ has the T-property and $f_{\mathcal{C}}(x) = p(x)$, as required. □

REMARK 5.21. We are now able to describe a simple test for determining whether any $k \times n$ matrix $\mathbf{G}$ is a generator matrix for an $[n, k]$ code $\mathcal{C}$ having the T-property. Firstly, by Lemma 5.14, if the first k columns of $\mathbf{G}$ are linearly dependent then $\mathbf{G}$ cannot be a generator matrix for a code with the T-property.

Next, since $\mathbf{G}$ has rank k, we may let $\mathcal{C}$ be the $[n, k]$ code generated by $\mathbf{G}$. Compute $\mathbf{R} = \mathbf{B}\mathbf{A}^{-1}$. By Lemma 5.16, $\mathcal{C}$ has the T-property only if the columns of $\mathbf{G}$ satisfy

$$\mathbf{g}_i^\top = \mathbf{R}^i\mathbf{g}_0 \quad \text{for all } i = 0, 1, \ldots, n-1. \tag{5.5}$$

If so, then since $\mathbf{A}$ was non-singular, the $\mathbf{g}_i$ span $\mathbb{F}_q^k$ and so $\mathbf{R}$ is cyclic and the minimal polynomial $p(x)$ of $\mathbf{R}$ has degree k. Thus $p(x)$ is the characteristic polynomial of $\mathbf{R}$, and is easily computed.

By Theorem 5.20 there is a bijection θ between $[n, k]$ codes with the T-property and polynomials in $\mathcal{F}_{n,k}$. If $\mathcal{C}$ has the T-property, then by Theorem 5.17, $\theta(\mathcal{C})$ must be the minimal polynomial of $\mathbf{R}$. Thus we next test whether the minimal polynomial $p(x)$ of $\mathbf{R}$ is in $\mathcal{F}_{n,k}$, that is whether $\mathrm{ord}(p(x)) > n$ and $x \nmid p(x)$. The latter is obvious, as it relates simply to whether or not the

constant term of $p(x)$ is non-zero. The former is also easy to check: from equation (5.5), $\mathrm{ord}(p(x)) < n$ if and only if two columns of $\mathbf{G}$ are identical, otherwise $\mathrm{ord}(p(x)) = n$ if and only if $p(x) \mid x^n - 1$. If $p(x) \notin \mathcal{F}_{n,k}$, then $\mathcal{C}$ does not have the T-property. Otherwise, by Lemma 5.16 or by Theorems 5.20 and 5.17, $\mathbf{G}$ is a generator matrix for an $[n, k]$ code $\mathcal{C}$ with the T-property. $\square$

5.4 Minimum Distance

We have seen that the set of sequences that may be constructed by overlapping a spanning set of words from an $[n, k]$ code is precisely the set of sequences obtainable from linear recurrence relations of degree k and order $e > n$. In this section we consider whether it is possible for the minimum distance between the codewords used in the sequence to exceed the design distance of the sequence, that is the minimum distance of the linear code that the n-windows of the sequence generate. Indeed, in the case $k = n$ we have already seen that this is so: the sequence $[t]$ of Example 5.8 had minimum distance 2 between its 5-windows but generated a linear code with minimum distance 1. We show that the design distance and actual minimum distance of such a sequence are equal whenever the sequence has a certain minimum period.

Fix integers n and k with $n > k$, and let $f(x)$ be a monic degree k polynomial with order $e > n$ and with $x \nmid f(x)$. We have already seen that the n-windows of the sequences with minimal polynomial $f(x)$ generate an $[n, k]$ code $\mathcal{C}$ say, and have described a parity check matrix $\mathbf{H} = (h_{ij})$ for $\mathcal{C}$ in terms of $f(x)$. The following result is simply an application of Theorem 1.7 to the parity check matrix $\mathbf{H}$.

THEOREM 5.22. *With the above notation, the minimum distance of* $\mathcal{C}$*, and hence the design distance of* $[s]$ *with respect to* n*, is equal to the least positive number of elements in the multi-set* $\{[f]_{k-n+1}^{n-k}, [f]_{k-n+2}^{n-k}, \ldots, [f]_{k}^{n-k}\}$ *which, considered as vectors in* $\mathbb{F}_q^{n-k}$*, are linearly dependent.*

PROOF. From equation 5.4 on page 140, the columns of the parity check matrix $\mathbf{H}$ of $\mathcal{C}$ are precisely the elements of the multi-set given. □

EXAMPLE 5.23. Let $\mathbb{F}_q = \mathbb{F}_3$, let $k = 3$ and let $f(x) = x^3 + x^2 + 2x + 1$, so $[f] = \ldots 0001211000 \ldots$. Starting with the 3-tuple "001", the recurrence relation defined by $f(x)$ gives the sequence

$$[s] = [00122212010110021112102022]$$

of period $e = 26$. Notice that $e = q^n - 1$ and thus $f(x)$ is in fact primitive.

We may choose any value of n in the range $k < n < e$. Starting with $n = 4$, we have $n - k = 1$. The appropriate 1-tuples from $[f]$ are (1), (2), (1) and (1). None are zero in themselves, but $(1)+(2) = (0)$, so the design distance of $[s]$ is 2. Indeed this distance is attained, as shown by the 4-windows (0012) and (1112) for example. That the design distance is attained should not be surprising since the $[4, 3]$ code $\mathcal{C}$ generated by the 4-windows of $[s]$ has $q^k = 27$ words, all but one of which (the all-zero word) appear in $[s]$ (see the description of m-sequences in Section 1.3).

With $n = 5$, the relevant 2-tuples from $[f]$ are (01), (12), (21), (11) and (10). Since $(12) + (21) = (00)$ the design distance of $[s]$ with respect to the window size 5 is again 2, and is achieved by the words (01222) and (02022), for example.

The case $n = 6$, gives the appropriate 3-tuples from $[f]$ as (001), (012),

(121), (211), (110) and (100). No pair of these is linearly dependent, although $(012)+2\cdot(100)+(121)=(000)$, so the design distance of $[s]$ with respect to the window size 6 is 3. Again, this is attained.

We may continue for larger n in a similar fashion. □

In the above example, the minimum distance achieved between n-windows is equal to the design distance. This is not always the case, as the following Example shows.

EXAMPLE 5.24. Let $\mathbb{F}_q=\mathbb{F}_2$ and let $f(x)=x^8+x^7+x^6+x^4+x^2+x+1$. This is the minimal polynomial of 15 sequences of period 17, between them possessing all 255 non-zero binary 6-tuples as windows. Some of these sequences are listed below:

$$\begin{aligned}[s] &= [00000001101001011]\\ [t] &= [00000010111011101]\\ [u] &= [00000100011110001]\\ [v] &= [00000111001100111].\end{aligned}$$

That $f(x)$ is the minimal polynomial of each follows since it is easy to check that $f(x)$ is a characteristic polynomial for these sequences, and minimality is assured by the fact that $f(x)$ is irreducible (see the Peterson Tables [52, Appendix C]).

Let $n=10$, then $n-k=2$ and the appropriate 2-tuples from $[f]$ include two instances of (11), giving a design distance of 2 between 10-windows of the sequences—this distance is the true minimum distance.

Now let $n=11$. The 3-tuples from $[f]$ include two instances of (111) so again

the design distance for 11-tuples is 2. This is the true distance for the sequences $[u]$ and $[v]$, but for $[s]$ and $[t]$ the least distance between 11-windows is 3. □

This demonstrates that the design distance of the n-tuples of a linear recurring sequence can be exceeded by the actual distance achieved. The sequences $[s]$ and $[t]$ each contain a subset of the available codewords having a higher minimum distance than that of the underlying linear code. Example 5.24 also demonstrates that the minimal polynomial alone does not determine the minimum distance between the n-windows of the sequence—different sequences with the same minimal polynomial may have different minimum distances between their n-windows.

It is difficult to characterise precisely the behaviour of the true minimum distance between n-windows of a sequence, given a certain design distance. However, the following result demonstrates that if the sequence concerned has at least a certain minimum least period, then the design distance and true minimum distance coincide. Alternatively, the design distance cannot be exceeded by the achieved minimum distance between n-windows of $[s]$ when the least period of $[s]$ is sufficiently large.

THEOREM 5.25. *Let $[s]$ be a sequence over $\mathbb{F}_q$, and suppose that $[s]$ has been constructed by overlapping a spanning set of words taken from an $[n,k]$ code $\mathcal{C}$ with minimum distance d. If the least period of $[s]$ is at least q^{k-1} then $[s]$ has minimum distance precisely d with respect to n.*

PROOF. Let $\mathbf{c} \in \mathcal{C}$ be a codeword of minimum weight d, and let C be the set of all multiples of $\mathbf{c}$. Notice $|C| = q$, and that the cosets of C partition $\mathcal{C}$ into q^{k-1}

equal parts. By the Pigeon Hole Principle any set of $q^{k-1}+1$ or more codewords includes at least one pair of codewords from the same coset, and these two words must be at distance d.

Finally, note that it is impossible for the period of $[s]$ to equal q^{k-1}. For, if so then then the minimal polynomial $f(x)$ of $[s]$ would have order q^{k-1}, and yet this polynomial has degree k and so its order must divide q^k-1, a contradiction.

□

The following Corollary provides an answer to the problem of Bartee and Wood described in Section 5.1.

COROLLARY 5.26. *Let $[s]$ be an m-sequence with degree k minimal polynomial $f(x) = x^k + f_{k-1}x^{k-1} + \cdots + f_0$. Define $f_k = 1$ and $f_i = 0$ for $i > k$ and $i < 0$, and let $[f] = (f_i)_{i\in\mathbb{Z}}$. Then for any $n > k$ the minimum distance between n-windows of $[s]$ is equal to the least positive number of elements in the multi-set $\mathcal{F} = \{[f]_{k-n+1}^{n-k}, [f]_{k-n+2}^{n-k}, \ldots, [f]_{k}^{n-k}\}$ which, considered as vectors in $\mathbb{F}_q^{n-k}$, are linearly dependent.*

PROOF. Apply Theorems 5.22 and 5.25. □

Bartee and Wood ([4]) comment that "no primitive polynomial characteristics have yet been found that allow a nonenumerative calculation of [the minimum distance between windows of an m-sequence]". By 'enumerative' they refer to generating the m-sequence explicitly, and thus finding its least weight n-window directly. This will give the minimum distance between n-windows, by Theorem 5.25 and the shift-and-add property of m-sequences. However, Corollary 5.26 allows us to adopt an alternative approach: we may simply take all

1-tuples, 2-tuples, 3-tuples, and so on from $\mathcal{F}$ and check whether they are linearly dependent. The size of the smallest linearly dependent set is thus obtained, and gives the n-window minimum distance. For small minimum distances in particular, this process is significantly easier than the enumerative technique described above.

Let $[s]$ be an m-sequence of least period n, arising from a primitive polynomial of degree k. Define $\mathcal{C}$ to be the m-sequence code, namely the an $[n, k]$ code consisting of all shifts of the minimal generating cycle of $[s]$, together with the all-zero n-tuple. It is clear that for any m in the range $k \leqslant m \leqslant n$, the set of m-windows of $[s]$, together with $0_{|m|}$, form a code $\mathcal{C}'$ that is obtained by truncating $\mathcal{C}$ to length m. Also, it may be shown that $\mathcal{C}^{\perp}$ is an $[n, n-k]$ Hamming code, that is a code with every non-zero k-tuple appearing as a column of its parity check matrix. We deduce that $\mathcal{C}'$ is dual to the code obtained by shortening this Hamming code to length m. This result was presented by Fredricsson ([23]), who used it to analyse the low-order moments of the weight distribution of $\mathcal{C}'$.

We present an example to show that in the case $q = 2, k = 4$ the condition on the period of $[s]$ given in Theorem 5.25 is as tight as possible.

EXAMPLE 5.27. Let $[s] = [0001011]$, a sequence over $\mathbb{F}_2$ with minimal polynomial $f(x) = x^4 + x^2 + x + 1$, so $k = 4$. We consider windows of size $n = 6$. The sequence $[f]$ has its relevant $(n-k)$-windows equal to (01), (11), (11), (10), (01) and (10). Since two of these are equal, the design distance of $[s]$ with respect to 6 is 2, although the actual distance achieved is 3. Notice that the period of $[s]$ is $q^{k-1} - 1$. □

We present another application of Theorem 5.25.

LEMMA 5.28. *Fix $n \geqslant 5$. The least period of an $(n,2)$-CW sequence $[s]$ over $\mathbb{F}_2$ with $\mathrm{lc}[s] < n$ is at most $2^{n-2}-1$.*

PROOF. Suppose that $[s]$ is an $(n,2)$-CW sequence with period greater than $2^{n-2}-1$ and $\mathrm{lc}[s] < n$. The period of $[s]$ must divide $2^{\mathrm{lc}[s]}-1$, thus we cannot have $\mathrm{lc}[s] < n-2$. If $\mathrm{lc}[s] = n-1$ then by Theorem 1.4, the n-windows of $[s]$ must span an $[n, n-1]$ code $\mathcal{C}$. Since the period of $[s]$ is at least 2^{n-2}, by Theorem 5.25 the minimum distance between n-windows of $[s]$ is equal to that of $\mathcal{C}$. For $\mathcal{C}$ to have minimum distance 2 it must be a parity check code, and thus the period of $[s]$ is n, giving $n \geqslant 2^{n-2}$ which for $n \geqslant 5$ is a contradiction. The result follows. □

Thus we have an upper bound on the least period of $(n,2)$-CW sequences over $\mathbb{F}_2$ that may be formed from a linear code with the T-property.

5.5 Computer Searches

In this section we describe the results of some computer searches for optimal (n,δ)-CW sequences for various values of the parameters. Patterns in the results lead us to some general conjectures. This Section considers only the binary case.

From the results above, we see that (n,δ)-CW sequences can be partitioned into two categories: those with linear complexity less than n, and those with complexity at least n. The former case corresponds to sequences formed by overlapping words of a linear code with the T-property; this case is of particular interest due to its tractability. In the latter case the sequence cannot be formed

TABLE 5.1. Least period of optimal (n, δ)-CW sequences over $\mathbb{F}_2$.

n	Complexity $< n$				Complexity $\geqslant n$			
	Minimum distance δ				Minimum distance δ			
	2	3	4	5	2	3	4	5
5	7	—	—	—	**10**	—	—	—
6	14	7	—	—	**15**	—	—	—
7	**31**	**14**	—	—	30	8	—	—
8	**63**	15	—	—	**63**	**16**	—	—
9		**31**	—	—		29	**11**	—
10		**62**	15	—		45	**22**	**11**
11			**31**	12			28	**22**
12			**62**	15			40	**26**
13				**31**				28
14				**62**				40

Entries in bold face denote optimal periods regardless of linear complexity. An entry of '—' indicates that no such sequence exists. An empty entry shows that the search was not completed for the particular values of the parameters concerned.

in this way. For given values of n and δ, we are primarily interested in forming (n, δ)-CW sequences with as great a least period as possible, regardless of which of the above cases the sequences reside in. Thus it is of interest to compare the maximum least periods of (n, δ)-CW sequences attainable in each of the two cases described.

A computer search for all (n, δ)-CW sequences was performed for small values of the parameters, and the optimal sequences in each of the cases described above recorded. The complete sequences and their minimal polynomials are given in Appendix A. Table 5.1 shows the least period of the longest possible (n, δ)-CW sequences in each of the cases considered.

Whilst we are unable to draw any conclusions regarding the general behaviour of these sequences, we make the following observations on the results presented

here and in Appendix A.

(1) The minimal polynomials of sequences with complexity less than n were very often either $p(x)$ or $(x+1)^2 p(x)$, where $p(x)$ is a primitive polynomial. The former are marked 'Type M' (for these are m-sequences) and the latter 'Type D' in Appendix A.

(2) The minimal polynomials of sequences with complexity at least n were very often equal to $x^\ell + 1$ or $(x^\ell + 1)/(x + 1)$, where ℓ is the least period of the sequence. These sequences are marked 'Type H' in Appendix A.

(3) A number of the sequences were 'self-dual' ([32]). These include those sequences of Type D. The remaining such sequences are marked 'Type S'. (A sequence is 'self-dual' if and only if it is equal to a cyclic shift of its bit-wise complement.)

(4) For each δ, the periods of (n, δ)-CW sequences with complexity at least n tended to exceed those with complexity less than n for the first few values of n for which such sequences existed. For the higher values of n for which a search was feasible, the opposite trend is observed. (This can be seen in the pattern of bold-face entries in Table 5.1.)

Sequences of form other than the Types described above are marked 'Type O'.

We focus on remark (4). This suggests that for larger values of n, optimal (n, δ)-CW sequences may be found from codes with the T-property. We are led to the following conjecture.

CONJECTURE 5.29. *For each integer δ there exists a constant C_δ such that for all $n \geqslant C_\delta$, amongst the (n, δ)-CW sequences of greatest possible period there*

exists a sequence with linear complexity less than n.

That is, for each δ and for n sufficiently large, codes with the T-property may produce optimal (n, δ)-CW sequences.

We concentrate for a moment on the case $\delta = 2$. This is perhaps the most important case with regard to the position location application described in Section 5.1, as it corresponds to single error detection, and represents the first step away from systems with no error control capability. Ignoring the exception at $n = 6$, we see from Table 5.1 that the optimal periods of (n, δ)-CW sequences with complexity less that n follow a familiar pattern, namely that the least period is given by $2^{n-2} - 1$. This suggests that the upper bound of Lemma 5.28 may be tight.

CONJECTURE 5.30. *For all $n \geqslant 5, n \neq 6$, the maximum least period for a $(n, 2)$-CW sequence over $\mathbb{F}_2$ with complexity less than n is $2^{n-2} - 1$.*

This would imply that $L_2(n, 2) \geqslant 2^{n-2} - 1$ for all $n \geqslant 7$. The only instances with $\delta = 2$ where the entry in Table 5.1 for sequences with linear complexity at least n exceeds the entry for sequences with linear complexity less than n are the cases $n = 5$ and 6. Strengthening Conjecture 5.30 by suggesting that Conjecture 5.29 holds in the case $\delta = 2$ with $C_2 = 7$, we obtain:

CONJECTURE 5.31. *For all $n \geqslant 6$, $L_2(n, 2) = 2^{n-2} - 1$.*

The author has no other results or conjectures regarding the existence or structure of (n, δ)-CW sequences with complexity n or greater. In the case of complexity less than n, the observation (1) that sequences of Types M and D can be optimal leads to the constructions given in the next Section. These depend

on special types of primitive polynomial, whose existence is also the subject of conjecture. The computational evidence in favour, however, strengthens the case for Conjecture 5.30.

5.6 Constructions from m-sequences

In this section we concentrate on the design of $(n, 2)$-CW sequences. In Section 5.5 we conjectured that for sufficiently large n, optimal sequences may be formed from linear codes in this case. We also observed that many of the optimal sequences found with complexity less than n had minimal polynomials of Type M or Type D. We present general constructions based on these results, which use primitive polynomials with particular properties. We also make conjectures regarding the general existence of these polynomials, and present computational evidence to support them.

We begin with a simple result that gives a construction for $(n, 2)$-CW sequences over any field $\mathbb{F}_q$.

THEOREM 5.32. *Let $f(x) = f_k x^k + f_{k-1}x^{k-1} + \cdots + f_0$ be a primitive polynomial of degree k over $\mathbb{F}_q$ and let $[s]$ be its corresponding m-sequence. Let r be the least positive integer such that $[s]$ is a $(k+r, 2)$-CW sequence. Then r is the least integer such that the sequence $f_{k-1}, f_{k-2}, \ldots, f_0$ of coefficients of $f(x)$ does not contain r consecutive zero terms.*

PROOF. The $(k+r)$-windows of $[s]$ form a $[k+r, k]$ code $\mathcal{C}$. By Theorem 5.22, $\mathcal{C}$ has minimum distance 2 if and only if the sequence $f_{k-1}, f_{k-2}, \ldots, f_0$ does not contain r consecutive zero terms. The least period of $[s]$ is $q^r - 1 > q^{r-1}$, and

so by Theorem 5.25 the set of $(k+r)$-window of $[s]$ forms a code with minimum distance equal to that of $\mathcal{C}$. $\square$

We give an alternative proof of this result. This demonstrates that the same result may be obtained by considering the properties of m-sequences, without using our previous theory on sequences obtained by overlapping words of a linear code. In this way it also serves as preparation for the proof of a subsequent result.

PROOF. Fix $r > 0$, let v be a $(k+r)$-window in $[s]$ and pick $w \in \mathbb{F}_q^{k+r} \setminus \{v\}$. By the shift-and-add property of m-sequences, w can occur as a $(k+r)$-window in $[s]$ if and only if $[s]$ also contains as a $(k+r)$-window the term-wise difference of v and w. Thus $[s]$ is a $(k+r, 2)$-CW sequence if and only if it contains no $(k+r)$-windows of (Hamming) weight 1.

Let $u = (u_0, u_1, \ldots, u_{k+r})$ be a $(k+r)$-window of $[s]$ in which only u_j is non-zero. Notice we must have $r < j < k$ or else u, and thus the m-sequence $[s]$, contains a sequence of k consecutive zero terms. Since $[s]$ satisfies

$$\sum_{i=0}^{k} f_i s_{i+t} = 0 \quad \text{for all } t \in \mathbb{Z},$$

we have that

$$\sum_{i=0}^{k} f_i u_i = \sum_{i=0}^{k} f_i u_{i+1} = \cdots = \sum_{i=0}^{k} f_i u_{i+r-1} = 0 \tag{5.6}$$

thus $f_j = f_{j-1} = \cdots = f_{j-r+1} = 0$, a contradiction.

Conversely, suppose j is such that $f_j = f_{j-1} = \cdots = f_{j-r+1} = 0$, and let $u = (u_0, u_1, \ldots, u_{r-1})$, where only u_j is non-zero. As $[s]$ is an m-sequence, u must occur exactly once in each period of $[s]$, and by (5.6) must then be succeeded in $[s]$ by r consecutive zero terms. This gives a $(k+r)$-window in $[s]$ of weight 1, a contradiction. $\square$

A question that now arises naturally concerns the existence of primitive polynomials whose longest run of zero coefficients is as short as possible. There has been previous work on the existence of primitive polynomials with certain conditions on their coefficients. This work has tended to centre on two problems. The first is the problem of finding primitive polynomials of least weight. These polynomials are of interest as they may be used to increase the efficiency of certain algorithms for computation in finite fields. The second problem concerns the existence of primitive polynomials of degree k with certain coefficients predetermined, notably the coefficient of x^{k-1}, whose value is the trace of the corresponding primitive element ([11], [27], [45]). It should be noted that the conditions which we seek on the coefficients of primitive polynomials involve all the coefficients, and so, in an intuitive sense, may be stronger than those mentioned above. Given the difficulties experienced by other authors in this area, we should not expect strong results to be easily forthcoming, and indeed the author has no general results.

In the case $\mathbb{F}_q = \mathbb{F}_2$, primitive polynomials having no zero coefficients do not exist for degrees 3 or over. We consider instead whether there exist primitive polynomials over $\mathbb{F}_2$ of arbitrary degree not having two consecutive zero coefficients. By Theorem 5.32, the existence of such a polynomial with degree k would imply the existence of a $(k+2,2)$-CW sequence over $\mathbb{F}_2$ with period $2^k - 1$, whose $(k+2)$-windows span a linear code of dimension k. The author was unable to produce any general results on the existence or otherwise of these polynomials. However, computer searches showed that such polynomials of degree k exist for all k in the range $3 \leqslant k \leqslant 45$, except in the case $k = 4$. The

polynomials are listed explicitly in Appendix B, Table B.1. We note that our ability to find such polynomials for such a large range of values k suggests that perhaps such polynomials do exist for all $k \geqslant 5$. In particular, note that towards the upper end of the range computed, an extremely small portion of the total search space of all primitive polynomials was examined—for example, there are $\phi(2^{45}-1)/45 = 634,404,960,000$ primitive polynomials of degree 45, but only 6,314 were tested before a positive result was found. The upper bound of 45 was a consequence of the limited computing capacity available, and we leave the general existence problem open:

CONJECTURE 5.33. *Fix $k \geqslant 3, k \neq 4$. There exists a primitive polynomial over $\mathbb{F}_2$ of degree k not having two consecutive zero coefficients.*

LEMMA 5.34. *Conjectures 5.30 and 5.33 are equivalent.*

PROOF. Suppose Conjecture 5.33 holds. Then by Theorem 5.32, for all $n \geqslant 5, n \neq 6$ the m-sequences formed from degree $n-2$ primitive polynomials over $\mathbb{F}_2$ not having two consecutive zero coefficients are $(n,2)$-CW sequences, with linear complexity $n-2$. These sequences have period $2^{n-2}-1$, which is maximal by Lemma 5.28.

Conversely, suppose that Conjecture 5.30 holds. Then for all $n \geqslant 5, n \neq 6$ there exists a $(n,2)$-CW sequences $[s]$ over $\mathbb{F}_2$ with $\mathrm{lc}[s] < n$ and least period $2^{n-2}-1$. By the argument in the proof of Lemma 5.28, we must have $\mathrm{lc}[s] = n-2$. By the least period of $[s]$, the minimal polynomial $m(x)$ of $[s]$ is primitive, and has degree $r = n-2$. Since $[s]$ has a minimum distance with respect to n of at least 2, by Theorem 5.32 the sequence of coefficients of $m(x)$ does not contain

$n - r = 2$ consecutive zero terms. □

The existence of primitive polynomials of the type described in Conjecture 5.33 for a large range of k, together with Lemma 5.34, give support to Conjecture 5.30.

We consider next how sequences like those of 'Type D' described in Section 5.5 and listed in Appendix A may be constructed. We write $[\overline{s}]$ to denote the term-wise complement of the binary sequence $[s]$.

LEMMA 5.35. *Let $[s]$ be an m-sequence formed from a degree k primitive polynomial $f(x)$ over $\mathbb{F}_2$. Then $[t] = D_0^{-1}[\overline{s}]$ is self-dual, has least period $2^{k+1} - 2$ and has minimal polynomial $(x+1)^2 f(x)$.*

PROOF. Note first that the least period n of $[s]$ is $2^k - 1$ and that by the balance property of m-sequences, $\mathrm{wt}([s]) = 0$. Thus $\mathrm{wt}([\overline{s}]) = 1$ and by Lemma 2.3 the least period m of $[t]$ is $2^{k+1} - 2$. That $[t]$ is self-dual follows from Remark 2.4.

Since $[\overline{s}]$ has a run of k consecutive zero terms, it must have minimal polynomial of degree $k+1$ or greater. Viewing $[\overline{s}]$ as $[s] + [1]$, that $(x+1)f(x)$ is the minimal polynomial for $[\overline{s}]$ follows from the fact that $D[\overline{s}]$ must be equal to a shift of $[s]$, by the shift-and-add property of m-sequences and from the fact that $\mathbb{F}_2$ has characteristic 2.

Suppose $\mathrm{lc}[t] = \ell$. We claim $\ell \geqslant k+2$, since the period of $[t]$ is $2^{k+1} - 2$ which is not a factor of $2^\ell - 1$ for any $\ell \leqslant k+1$. Now $D[t] = [\overline{s}]$, thus $(x+1)^2 f(x)$ is the minimal polynomial for $[t]$. □

The Tables in Appendix A give sequences of 'Type D' as optimal (n, δ)-CW sequence in a number of instances. The following result characterises (albeit

somewhat untidily) when given differences between n-windows of $[t]$ may occur.

LEMMA 5.36. *Let $[s]$ be an m-sequence over $\mathbb{F}_2$ formed from a primitive polynomial of degree k, and let $[t] = D_0^{-1}[\overline{s}]$. Let n be any integer, $n \geqslant 1$, and let w be any n-tuple over $\mathbb{F}_2$. Then there exist n-windows u and v of $[t]$ with difference $u + v = w$ if and only if Dw is an $(n-1)$-window of $[s]$.*

PROOF. Suppose n-tuples u and v as described exist. By the linearity of the operator D and by the shift-and-add property of m-sequences, we have

$$\begin{aligned} D(u+v) &= Du + Dv \\ &= x + y \\ &= z, \end{aligned}$$

for some $(n-1)$-windows x and y of $[\overline{s}]$, and some $(n-1)$-window $[z]$ of $[s]$.

Conversely, suppose that $Dw = z = [s]_a^{n-1}$ for some integer a. There exists b such that $[s] + E^b[s] = E^a[s]$, so z occurs in $[s] + E^b[s]$ at position 0. But $[s] + E^b[s] = D([t] + E^b[t])$, thus $[t] + E^b[t]$ must contain either w or $\overline{w}$ at position 0. In the former case we are done, as then $[t]_0^n + [t]_b^n = w$. In the latter case, by Lemma 5.35 we have that $[t]$ is self-dual of least period $2^{k+1} - 2$, and so w must occur in $[t] + E^b[t]$ at position $2^k - 1$. A similar argument then applies.

□

Thus, to ensure that $[t]$ has minimum distance at least δ between its n-windows, it is necessary and sufficient to form all non-zero n-tuples of weight less than δ, apply the operator D to each and ensure that none of the resulting $(n-1)$-tuples appear as windows of $[s]$. This may seem complicated, since ideally we seek a direct relationship between the primitive polynomial $f(x)$ chosen and the

minimum distance between n-windows of $[t]$. Theorem 5.37 shows that in the case $\delta = 2$ and $n = r + 3$, just such a relationship may be attained in this way.

THEOREM 5.37. *Let* $f(x) = f_k x^k + f_{k-1}x^{k-1} + \cdots + f_0$ *be a primitive polynomial of degree* k *over* $\mathbb{F}_2$, *let* $[s]$ *be its corresponding m-sequence and let* $[t] = D^{-1}[\overline{s}]$. *Then* $[t]$ *is a* $(k+3, 2)$-CW *sequence if and only if* $f(x)$ *does not contain amongst its coefficients a sequence of three or more consecutive equal terms.*

PROOF. Suppose that u and v are $(k+3)$-windows of $[t]$ at Hamming distance 1. Writing $u + v = w = (w_0, w_1, \ldots, w_{r+2})$, there exists some ℓ in the range $0 \leqslant \ell \leqslant k + 2$ such that

$$w_i = \begin{cases} 1 & \text{if } i = \ell, \text{ or} \\ 0 & \text{otherwise.} \end{cases} \tag{5.7}$$

Define the sequence $[z] = (z_i)_{i \in \mathbb{Z}}$ by

$$z_i \quad = \quad \begin{cases} 0 & \text{if } i < \ell - 1; \\ 1 & \text{if } i = \ell - 1 \text{ or } i = \ell, \text{ or} \\ 0 & \text{if } i > \ell, \end{cases} \tag{5.8}$$

and let $z = [z]_0^{k+2}$. Notice that $Dw = z$, so by Lemma 5.36, z must appear as a $(k+2)$-window in $[s]$. If $\ell \leqslant 1$ or $\ell \geqslant k + 1$ this implies that $[s]$ contains a window of at least k consecutive zero terms, which is not possible. Otherwise, we have

$$\sum_{i=0}^{k} f_i z_i \quad = \quad \sum_{i=0}^{k} f_i z_{i+1} \quad = \quad 0 \tag{5.9}$$

and so $f_\ell = f_{\ell-1}$ and $f_{\ell-1} = f_{\ell-2}$, giving three consecutive equal coefficients of $f(x)$.

Conversely, suppose f is such that for some ℓ in the range $2 \leqslant \ell \leqslant r$ we have $f_\ell = f_{\ell-1} = f_{\ell-2}$. Let $[z]$ be as defined in equation (5.8). As $[z]_0^k$ is not all-zero, it must occur as an k-window of $[s]$ and by equation (5.9) must do so succeeded by z_k and z_{k+1}. With w defined as in equation (5.7), we observe $Dw = [z]_0^{k+2}$ and apply Lemma 5.36 to conclude that there exist $(k+3)$-windows of $[t]$ at distance $\mathrm{wt}\,(w) = 1$. □

This result characterises when a minimum distance of 2 is attained between $(k+3)$-windows of the sequence $[t]$ described above, in terms of the primitive polynomial $f(x)$. Thus we are interested again in the existence of special primitive polynomials over $\mathbb{F}_2$, this time in those not having three consecutive equal coefficients. A computer search showed that such polynomials of degree k exist for all k in the range $3 \leqslant k \leqslant 45$. These are listed in Appendix B, Table B.2. Again, the computational evidence to suggest that these polynomials exist for all $k \geqslant 4$ is strong, since they appear to be relatively easy to find. However, in the absence of theoretical results, the general existence question remains open.

CONJECTURE 5.38. *Let k be any integer greater than 2. Then there exists a primitive polynomial over $\mathbb{F}_2$ of degree k not having three consecutive equal coefficients.*

This conjecture would establish a lower bound of $2^{n-2} - 2$ on the length of maximal $(n, 2)$-CW sequences over $\mathbb{F}_2$ having complexity less than n—just one less than the bound in Conjecture 5.30. Thus, in the event that Conjecture 5.33 is false, Theorem 5.37 presents a means of constructing $(n, 2)$-CW sequences over $\mathbb{F}_2$, attaining almost the same least periods. (For example, we have already noticed that there is no primitive polynomial of degree 4 over $\mathbb{F}_2$, and that this

gives rise to the exception occurring at $n = 6$ in the pattern of entries in the $\delta = 2$, complexity $< n$ column of Table 5.1. However, the value 14 that does appear at this position in the Table arises from a sequence of the form described in Theorem 5.37.)

We turn now to consider sequences over fields $\mathbb{F}_q \neq \mathbb{F}_2$. In this case, it is possible that there might exist primitive polynomials of arbitrary degree k not having any zero coefficients. Such a polynomial would give rise to an m-sequence which would be a $(k+1, 2)$-CW sequence of period $q^k - 1$. Regardless of the s-property, the maximum number of words in any code with distance 2 and length $k+1$ is q^k (for example, the parity-check code, which attains the Hamming bound). Thus such a sequence is certainly asymptotically optimal as $k \to \infty$, and either optimal or almost optimal for all k. Again, the author has no theoretical results regarding the existence of such sequences, but computer searches seem to find the requisite polynomials readily—see Appendix B, Tables B.3–B.6. We make the following conjecture:

CONJECTURE 5.39. *Let q be a prime power, $q \neq 2$, and let k be any positive integer. Then there exists a primitive polynomial over $\mathbb{F}_q$ of degree k with no zero coefficients.*

As stated above, this conjecture would lead to sequences just one short of the theoretical upper bound imposed by the number of words possible in a code with distance 2. Moreover, there is reasonable numerical evidence in its favour. Thus one suggestion of Conjecture 5.39 is that, in the case $\delta = 2$ at least, Conjecture 5.29 could perhaps be extended to arbitrary finite fields.

5.7 Error Correction and Decoding

Let $[s]$ be a (n, δ)-CW sequence over an alphabet $\mathcal{A}$. The position location application that motivated our study of code window sequences requires that the following problem be solved: given that our position sensing device has read a particular $w \in \mathcal{A}^n$, what is the most likely position of the device? The process of solving this problem can be divided into two stages:

(1) performing error detection or correction on the input w, and

(2) given an error-free w', determining the position of w' in $[s]$.

Working over $\mathbb{F}_q$, let $\mathcal{C}'$ be the code consisting of all n-windows of $[s]$, and let $\mathcal{C}$ be the linear code generated by these words. We have seen that if $\mathrm{lc}[s] \geqslant n$ then $\mathcal{C}$ consists simply of $\mathbb{F}_q^n$, and that if $\mathrm{lc}[s] = k < n$ then $\mathcal{C}$ is an $[n, k]$ code. In the former case $\mathcal{C}$ is of little use to us, and we must find some means of error detection or correction in $\mathcal{C}'$ directly. The latter case is more tractable: if the number of errors is small then it is reasonable to provide error correction in $\mathcal{C}$ by means of a look-up table of syndromes and coset leaders, and error detection up to the limits of the code may be achieved by means of syndromes for any number of errors ([38, Chapter 1, §4]).

By Remark 5.12, we know that $\mathcal{C}' \subsetneq \mathcal{C}$, and so this procedure is only an approximation to true nearest-neighbour decoding in $\mathcal{C}'$. However, in the case when $[s]$ is an m-sequence the only codeword in $\mathcal{C}$ but not in $\mathcal{C}'$ is the all-zero codeword, and in the case when $[s]$ is formed as described in Theorem 5.37 there are only two codewords in $\mathcal{C}$ not in $\mathcal{C}'$. Thus in these cases error detection or correction in $\mathcal{C}$ forms a particularly close approximation to error detection or

correction in $\mathcal{C}'$.

Suppose now that error correction on w has been performed (or that error detection used to remove invalid versions of w) and that we have an n-window w' of $[s]$, whose position in $[s]$ must be determined to complete stage (2) above. Of course, different approaches to this problem are required depending on the structure of the sequence $[s]$. Given the trends observed in the results of the computer searches in Section 5.5, we concentrate on the sequences studied in Section 5.6. Suppose that $[s]$ is an m-sequence. In Section 1.1 we saw how the decoding problem for $[s]$ is then equivalent to the discrete logarithm problem, which is difficult in general. This is also the case for those sequences formed as in Theorem 5.37, but in this case a look-up table similar to the tables $L^{(i)}(j)$ of Chapter 2 is also required.

This is similar to the situation for position location without error control, before it was demonstrated in [42] and in Chapters 2–4 of this thesis that some combinatorial constructions for de Bruijn sequences do admit efficient decoding algorithms. Thus we are motivated to try to find combinatorial constructions for code window sequences, whose structure allows efficient decoding. This may require the development of new techniques, or the modification of existing techniques currently used for constructing de Bruijn sequences. Section 5.8 provides an example of the latter.

5.8 A Product Construction

We have already generalised the idea of a de Bruijn sequence to that of a code window sequence. The following definition generalises spaced de Bruijn

sequences in an analogous fashion. This will allow us to generalise the method of constructing de Bruijn sequences given in Theorem 4.3 to give a construction for code window sequences.

DEFINITION 5.40. Let v, δ, n, N and c be positive integers, with $n \geqslant v \geqslant \delta$, and let $[s]$ be a sequence of least period n over an alphabet of size c. We say that $[s]$ is a (v, δ, N) *spaced code window sequence* (hereafter a (v, δ, N)-SCW sequence) if and only if for any integers j_1 and j_2 with $0 \leqslant j_1, j_2 < \mathrm{lcm}(n, N)$ and $j_1 \equiv j_2 \pmod{N}$, the j_1th and j_2th v-windows of $[s]$ differ in at least δ places. □

That is, for each congruence class k modulo N, the set of v-windows of $[s]$ starting at a position $j \equiv k \pmod{N}$ form a code with minimum (Hamming) distance δ.

The following result is a straightforward generalisation of Theorem 4.3.

THEOREM 5.41. *Let $[s]$ be a (v, δ)-CW sequence with least period N over an alphabet $\mathcal{C}$ of size c, and let $[t]$ be a (v, δ, N)-SCW sequence with least period n over an alphabet $\mathcal{D}$ of size d. Then the sequence $[r] = [s] \times [t]$ is a (v, δ)-CW sequence with least period $\mathrm{lcm}(n, N)$ over the alphabet $\mathcal{C} \times \mathcal{D}$ of size cd.*

PROOF. Let j_1 and j_2 be integers, with $0 \leqslant j_1, j_2 < \mathrm{lcm}(n, N)$ and let $w_1, w_2 \in (\mathcal{C} \times \mathcal{D})^v$ be the j_1th and j_2th v-windows of $[r]$, respectively.

Suppose that $j_1 \not\equiv j_2 \pmod{N}$. By projecting w_1 and w_2 onto $\mathcal{C}$ and by the definition of $[s]$, it is clear that w_1 and w_2 are at distance at least δ. However, if $j_1 \equiv j_2 \pmod{N}$ then as $j_1, j_2 < \mathrm{lcm}(n, M)$ we have $j_1 \not\equiv j_2 \pmod{n}$; projecting onto $\mathcal{D}$ and the definition of $[t]$ give that w_1 and w_2 are again at distance at

least δ. □

REMARK 5.42. The construction described in Theorem 5.41 may be generalised further, in a similar way that Theorem 4.24 is a generalisation of Theorem 4.3. This would provide a construction for sets of sequences all of whose n-windows are at least a given minimum distance. This would involve generalising the definition of perfect multi-factors, in a straightforward way. However, the purpose of this Section is to give an example of how a combinatorial technique that has previously been used to construct de Bruijn sequences may be adapted to the construction of code window sequences, and this is more easily achieved without the extra complexity that working on sets of sequences rather than individual sequences imposes. Thus this generalisation is left to the interested reader. □

The following result gives a very simple means of constructing spaced code window sequences, and thus demonstrates that they exist.

LEMMA 5.43. *Let v, δ and n be positive integers, and let $[s]$ be a (v, δ)-*CW *sequence with period n. Then $[s]$ is also a (v, δ, N)-*SCW *sequence for all N co-prime to n.*

PROOF. Apply the Chinese Remainder Theorem. □

We shall now give a less trivial and more useful construction for spaced code window sequences.

CONSTRUCTION 5.44. Let $[s]$ be a (v, δ)-CW sequence of least period n over an alphabet $\mathcal{C}$ of size c, and let f and g be positive integers with $\gcd(g, n) = 1$. Let $N = fv + g$. For each k in the range $0 \leqslant k < n$, define the c-ary N-tuple

$t^{(k)} = (t_0^{(k)}, t_1^{(k)}, \ldots, t_{N-1}^{(k)})$ by

$$\begin{aligned}(t_0^{(k)}, t_1^{(k)}, \ldots, t_{fv-1}^{(k)}) &= (s_k, s_{k+1}, \ldots, s_{k+v-1})^f \quad \text{and} \\ (t_{fv}^{(k)}, t_{fv+1}^{(k)}, \ldots, t_{N-1}^{(k)}) &= (s_{k+v}, s_{k+v+1}, \ldots, s_{k+v+g-1}).\end{aligned}$$

(Recall that the f in the exponent denotes the f-fold concatenation of the given v-tuple with itself.) Then, define the sequence $[t] = (t_i)_{i\in\mathbb{Z}}$ of period nN by

$$t_i = t_a^{(b)}$$

where

$$a = i \bmod N \quad \text{and} \quad b = g \times \left\lfloor \frac{i}{N} \right\rfloor \bmod n. \tag{5.10}$$

□

THEOREM 5.45. *Let $[t]$ be a sequence as defined in Construction 5.44. Then $[t]$ is a (v, δ, N)-*SCW *sequence over $\mathcal{C}$ with least period nN.*

PROOF. Let j_1 and j_2 be distinct integers with $j_1 \equiv j_2 \pmod N$ and $0 \leqslant j_1, j_2 < nN$ and let w_1 and w_2 be the j_1th and j_2th v-windows of $[t]$, respectively. We are required to show that w_1 and w_2 are at distance at least δ.

Define

$$b_i = g \times \left\lfloor \frac{j_i}{N} \right\rfloor \bmod n$$

for $i = 1, 2$. As $j_1 \neq j_2$ and $j_1 \equiv j_2 \pmod N$ we have that $\lfloor j_1/N \rfloor \neq \lfloor j_2/N \rfloor$, and since $j_1, j_2 < nN$ and $\gcd(g, n) = 1$ we deduce that $b_1 \not\equiv b_2 \pmod n$.

Let $k = j_1 \bmod N$ and define $\ell_i = b_i + (k \bmod v)$ for $i = 1, 2$. Firstly, consider the case $0 \leqslant k < N - v$. Then by the definition of $[t]$, w_i is the kth v-window

of $t^{(b_i)}$, for $i = 1, 2$. Suppose further that $k < (f-1)v$; by the definition of the $t^{(b_i)}$ we have that

$$\begin{aligned} w_i &= (s_{\ell_i}, s_{\ell_i+1}, \ldots, s_{b_i+v-1}, s_{b_i}, s_{b_i+1}, \ldots, s_{\ell_i-1}) \\ &= E^{k \bmod v}(s_{b_i}, s_{b_i+1}, \ldots, s_{b_i+v-1}) \end{aligned}$$

for $i = 1, 2$. Now suppose instead that $(f-1)v \leqslant k < N - v$. In a similar way we have

$$w_i = (s_{\ell_i}, s_{\ell_i+1}, \ldots, s_{\ell_i+v-1}), \quad \text{for } i = 1, 2.$$

Now consider that case $N - v \leqslant k < N$. Then w_i consists of the last $N - k$ terms of $t^{(b_i)}$ followed by the first $v - N + k$ terms of $t^{((b_i+g) \bmod n)}$, by the choice of b in equation (5.10). But, by the design of the $t^{(b)}$ the last v terms of $t^{(b_i)}$ and the first v terms of $t^{((b_i+g) \bmod n)}$ coincide. Thus

$$w_i = E^{k \bmod v}(s_{b_i+g}, s_{b_i+g+1}, \ldots, s_{b_i+g+v-1}), \quad \text{for } i = 1, 2.$$

In each of these cases, by the fact that $b_1 \not\equiv b_2 \pmod{n}$ (and so $\ell_1 \not\equiv \ell_2 \pmod{n}$), we see that w_1 and w_2 appear, perhaps shifted cyclicly, as distinct v-windows of $[s]$. By the properties of $[s]$, we deduce that w_1 and w_2 are at distance at least δ.

□

The decoding problem for spaced code window sequences may be defined similarly to Definition 4.4. In the case of a code window sequence $[t]$ formed from a spaced de Bruijn sequence $[s]$ using Construction 5.44, it is simple to show how the decoding problem for $[t]$ may be reduced to the decoding problem for $[s]$.

We now employ Construction 5.44 to construct an infinite family of spaced code window sequences, in an iterative fashion. The sequences arising are easily decoded by repeatedly applying the principle described above.

THEOREM 5.46. *Let* $[s]$ *be a* (v, δ)-CW *sequence with least period* n *over an alphabet of size* c*, and suppose there exist integers* f, g *such that* $n = fv + g$ *and* $\gcd(g, n) = 1$*. Then there exists a* (v, δ)-CW *sequence over an alphabet of size* c^i *having least period* n^i *for all* $i \geqslant 1$.

PROOF. We use induction upon i, and note that the case $i = 1$ is immediate.

Fix i. We assume by the inductive hypothesis that there exists a (v, δ)-CW sequence $[t]$ of least period n^i over an alphabet of size c^i. Notice that $\gcd(g, n^i) = 1$, since $\gcd(g, n) = 1$ by assumption. Then, by Theorem 5.45 there exists a (v, δ, n^i)-SCW sequence $[t]$ of least period n^{i+1} over the same alphabet. By Theorem 5.41 the sequence product $[s] \times [t]$ is a (v, δ)-CW sequence of least period n^{i+1} over an alphabet of size c^{i+1}, as required. □

5.9 Code Window Arrays

We may generalise the position location scheme of Section 1.1 to two dimensions. In such a scheme, a device moving relative to a plane surface reads a sub-array or window of elements from a larger array of information encoded on the surface. (We think of the device maintaining a constant orientation relative to the surface.) As in the one-dimensional case, we desire that each of these sub-arrays occur at a unique position in the overall array and thus allow the device to identify its position. Numerous previous authors have considered such schemes, in particular see [8]. There has been considerable progress in the construction of arrays suitable for this purpose ([17], [28], [37], [43], [47], [48], [49], [50]). However, most of these schemes involve constructing arrays in which every or almost every sub-array of a given size appears, and so these schemes are unable

to cope with sensor read errors.

A scheme of this kind using these arrays is central to the operation of a recently developed medical instrument for measuring the shape of the cornea ([59]). This device projects an array of the type described above onto the cornea, and obtains measurements on the shape of the cornea by analysing small sub-arrays of reflected light. Accurate identification of these sub-arrays is essential to determining which part of the cornea is being analysed. However, the designers of this system remark that "in the process of recovering the pattern from the input image a significant number of bits will be wrongly interpreted. Thus it is necessary to evaluate to what extent (despite misinterpretations) it is still possible to come to a unique identification of the sub-pattern (and thus to come to a complete fault correction)." Their solution is to obtain error-correction by examining sub-arrays much larger than those which, in an error-free environment, would be strictly necessary to determine a position uniquely. The consequence of this is a significant reduction in the resolution of the device.

Achieving error control simply by examining larger than necessary sub-arrays from an array designed to contain nearly every sub-array of a certain size seems likely to be inefficient. There is therefore some motivation to find more efficient schemes, based on arrays whose sub-arrays of a given size are designed to form a code with a given minimum distance. Clearly, this is a large area for future research, which we do not attempt to explore fully here. However, we demonstrate that an existing construction for generating arrays containing every window of a certain size may be modified to give arrays of the kind we seek. This involves an application of some of the techniques developed earlier in this Chapter.

DEFINITION 5.47. A $(p, q; m, n; \delta)$ *code window array* (henceforth denoted a $(p, q; m, n; \delta)$-CW array) is an infinite array $\mathcal{W} = (W_{i,j})_{i,j \in \mathbb{Z}}$ that has period p vertically, period q horizontally, and the property that any two $m \times n$ sub-arrays appearing at distinct positions (modulo the period) differ from each other in at least δ places. □

DEFINITION 5.48. Let $[t]$ be an (n, δ)-CW sequence over $\mathbb{Z}_c$. If $[t]$ has the additional property that for all $a \in \mathbb{Z}_c$, there do not exist two distinct n-windows x and y of $[t]$ with $\mathrm{d}(x + a_{|n|}, y) < \delta$, then we say that $[t]$ has property Y. □

The following construction is adapted from the technique of [17].

CONSTRUCTION 5.49. Let $\mathcal{S} = \{[s^{(0)}], [s^{(1)}], \ldots, [s^{(\ell-1)}]\}$ be an (m, p, c)-PF, with $[s^{(j)}] = (s_i^{(j)})_{i \in \mathbb{Z}}$ for each j in the range $0 \leqslant j \leqslant \ell - 1$. Let q, n and δ be integers, with $q > n > \delta$, and let $[t] = (t_i)_{i \in \mathbb{Z}}$ be an (n, δ)-CW sequence over $\mathbb{Z}_{c^m}$ of least period q having property Y. Define the array $\mathcal{W} = (W_{i,j})_{i,j \in \mathbb{Z}}$ by $W_{i,j} = s_y^{(x)}$ where $x = \lfloor t_j / p \rfloor$ and $y = (t_j \bmod p) + i$. □

THEOREM 5.50. *With the notation of Construction 5.49, the array $\mathcal{W}$ is a c-ary $(p, q; m, n; \delta)$-CW array.*

PROOF. That $\mathcal{W}$ has vertical least period p follows from the fact that all of the sequences $[s^{(i)}]$ have period p, and that this must also be the least period of at least one of these sequences. The horizontal least period of $\mathcal{W}$ follows from the fact that $[t]$ has least period q.

Pick distinct coordinate pairs (u, v) and $(u', v') \in \mathbb{Z} \times \mathbb{Z}$, with $0 \leqslant u, u' \leqslant p-1$ and $0 \leqslant v, v' \leqslant q-1$. Let $\mathcal{B}$ and $\mathcal{B}'$ denote the $m \times n$ sub-arrays of $\mathcal{W}$ occurring at positions (u, v) and (u', v') respectively. That is, if $\mathcal{B} = (B_{i,j})_{0 \leqslant i \leqslant m-1, 0 \leqslant j \leqslant n-1}$,

then $B_{i,j} = W_{u+i,v+j}$, with $\mathcal{B}'$ defined similarly.

For each i in the range $0 \leqslant i \leqslant n-1$ let b_i and b'_i be m-tuples representing the ith column of $\mathcal{B}$ and $\mathcal{B}'$, respectively. Notice that for each i in the range $0 \leqslant i \leqslant n-1$, we have $b_i = [s^{(x)}]_y^m$ where $x = \lfloor t_{v+i}/p \rfloor$ and $y = (t_{v+i} \bmod p) + u$. Similar expressions hold for b'_i.

It suffices to show that $\mathcal{B}$ and $\mathcal{B}'$ differ in at least δ places.

Suppose otherwise. Since $\delta < n$ there is at least one column in which $\mathcal{B}$ and $\mathcal{B}'$ are identical. Thus we may let k be an integer in the range $0 \leqslant k \leqslant n-1$ such that $b_k = b'_k$.

Now suppose additionally that $v = v'$. Then b_k and b'_k are drawn from the same column of $\mathcal{W}$, derived from $[s^{(x)}]$ where $x = \lfloor t'_{v+k}/p \rfloor$. Since $\mathcal{S}$ is a perfect factor, b_k appears as a window in $[s^{(x)}]$ at a unique position (modulo p), and so we deduce that $u = u'$, a contradiction. Thus we may assume $v \neq v'$.

For each m-tuple w, let the integer r_w be defined by $r_w = fp + g$, where f and g are the unique integers in the ranges $0 \leqslant f \leqslant \ell - 1$ and $0 \leqslant g \leqslant p-1$ with $w = [s^{(f)}]_g^m$. For each i in the range $0 \leqslant i \leqslant n-1$, we have $r_{b_i} = fp + g$, where $f = \lfloor t_{v+i}/p \rfloor$ and $g = (t_{v+i} \bmod p) + u$. Thus $r_{b_i} = t_{v+i} + u$. Similarly, $r_{b'_i} = t_{v'+i} + u'$. Now if $\mathcal{B}$ and $\mathcal{B}'$ differ in less than δ places, there there exist less than δ possible values for i in the range $0 \leqslant i \leqslant n-1$ for which $b_i \neq b'_i$, and hence for which $r_{b_i} \neq r_{b'_i}$ and $t_{v+i} + u \neq t_{v'+i} + u'$. Thus $\mathrm{d}([t]_v^n + u_{|n|}, [t]_{v'}^n + u'_{|n|}) < \delta$, contradicting property Y for $[t]$. □

Construction 5.49 can only be used to form code window arrays if we can find code window sequences with property Y. Firstly, notice that the binary

sequences arising from Theorem 5.37 have this property. This is clear from the fact that these sequences are self-dual. However, Construction 5.49 requires such a sequence over a larger alphabet than the binary alphabet to be of any real use. Fortunately, this may be achieved using a technique from Section 5.8, as our final result demonstrates.

This result uses sequences over $\mathbb{Z}_c^i$, for various values of i. We define addition in $\mathbb{Z}_c^i$ by first mapping the terms into $\mathbb{Z}_{c^i}$. The mapping used is the natural one, in which elements of $\mathbb{Z}_c^i$ are seen as i-digit base c representations of elements of $\mathbb{Z}_{c^i}$. It is through this definition of addition that the property Y for sequences over $\mathbb{Z}_c^i$ is defined.

THEOREM 5.51. *Let $[s]$ be a (u,δ)-CW sequence over $\mathbb{Z}_c$ with property Y, having least period N. Suppose that there exist integers f and g with $N = fu + g$ and $\gcd(g, N) = 1$. There there exists a (u,δ)-CW sequence $[s^{(i)}]$ over $\mathbb{Z}_{c^i}$ with property Y having least period N^i, for all $i \geqslant 1$.*

PROOF. The sequences $[s^{(i)}]$ are constructed by repeatedly applying Construction 5.44 and Theorems 5.45 and 5.41, in an identical manner to that used in the proof of Theorem 5.46.

The resulting sequences are (u,δ)-CW sequences by Theorem 5.46. It remains to show that each has property Y. This may be achieved by induction upon i, the case $i = 1$ being immediate.

Fix i, and let $[s^{(i)}]$ be a (u,δ)-CW sequence of least period N^i over $\mathbb{Z}_c^i$ having property Y. As in the proof of Theorem 5.46, we have that $\gcd(g, N^i) = 1$, and so we may apply Construction 5.44 to obtain a (u,δ,N)-SCW sequence. The

proof of Theorem 5.45 depended on showing that for any two n-windows w_1 and w_2 of $[t]$ at positions that are congruent modulo N, these two windows could be formed by re-arranging distinct n-windows of $[s^{(i)}]$. Since $[s^{(i)}]$ has property Y, the same method also shows that there exists no $a \in \mathbb{Z}_c^i$ such that $\mathrm{d}(w_1 + a_{|n|}, w_2) < \delta$.

The sequence $[s^{(i+1)}]$ is formed by taking $[s] \times [t]$. Consider two windows v_1 and v_2 of $[s^{(i+1)}]$, occurring at positions p_1 and p_2 respectively. If $p_1 \not\equiv p_2$, then since $[s]$ has property Y, there cannot exist a constant $a \in \mathbb{Z}_c^{i+1}$ giving $\mathrm{d}(v_1 + a_{|n|}, v_2) < \delta$. Alternatively, if $p_1 \equiv p_2 \bmod N$, then by the property of $[t]$ just described, the same result holds. Theorem 5.51 follows. □

Further Work

Construction 3.4 combined two techniques to yield de Bruijn sequences that could be decoded by an algorithm of very low complexity. However, some tables of information were required by this algorithm, and these could grow quickly with the span n of the sequence. In the binary case, we gave a slight variation on this construction for which these storage tables were not required (Theorem 3.18). Coupled with the results on de Bruijn sequences with composite alphabet size presented in Chapter 4, this gives a means of constructing easily decoded de Bruijn sequences of arbitrary span over any even-size alphabet.

A naturally arising question is to ask whether these techniques can be extended to arbitrary alphabet sizes. The method of Chapter 4 does not have any restriction on alphabet size, so the question arising is really whether we can construct de Bruijn sequences over alphabets of odd prime size and of arbitrary span, that may be efficiently decoded without the use of storage tables. In particular, was the restriction to binary alphabets made in the latter part of Chapter 3 really necessary, or can the method used in Construction 3.16 and Theorem 3.18 be modified to work with any prime alphabet size?

The author believes that such a generalisation is not possible. Theorem 3.18 depends on the ability to recursively compute the information that was previously stored in tables. The method may seem a little obscured, wrapped in the Lemmas of Section 3.4, but what is central is actually Remark 3.8: this shows how, in the binary case, the double sum $\eta_j[r]$ of elements from a sequence $[r]$ is equal to a sum of alternate elements from $[r]$. The double sum relates to the

sum of terms from a sequence formed by applying the operator D_0^{-1} to $[r]$, and the alternating sum is easily found when $[r]$ is formed by interleaving. This does not hold if the alphabet is not binary. For an alphabet $\mathbb{Z}_c$, the double sum would instead be expressed in terms of c sums involving every cth element of $[r]$, each starting at a different point. These would not relate to the interleaving of two sequences as before. The obvious solution is to replace the interleaving of two sequences by an interleaving involving c sequences. It is not at all obvious how a de Bruijn sequence could be constructed in this way. Even if it could, where previously the interleaving construction doubled the span of the sequence used, the new construction would multiply the span by a factor of c. This defeats the method previously used to ensure every possible span was attainable, by using the interleaving and GL constructions in turn. Instead, up to $c-1$ GL constructions may be required between each interleaving construction. The double sums used previously are then no longer sufficient to provide storage-free decoding. We must conclude that this approach seems doomed to failure.

Given the techniques that have now been developed, the general decoding problem for de Bruijn sequences could be considered 'solved' given a construction for de Bruijn sequences that may be decoded easily without the use of storage tables, over alphabets of odd prime size and of arbitrary odd span (the problem for even spans following from this using the interleaving construction). Such a construction remains a desirable goal.

A problem we have not discussed is the encoding problem for a de Bruijn sequence. That is, given a means of constructing a sequence, how efficiently can one compute what the ith n-window of the sequence would be, for arbi-

trary i? For prime power alphabet sizes, the de Bruijn sequences derived from m-sequences are quick to encode, since this problem may be reduced to computing the appropriate power of the companion matrix corresponding to the minimal polynomial of the m-sequences. For other alphabets, it may be worth noting that the technique of Chapter 4 seems to lend itself to this problem. It also seems plausible that the interleaving and GL constructions, even when combined, may allow a kind of recursive encoding algorithm. Details are left as an open problem for the interested reader.

A related problem to that of encoding is the rapid term-wise generation of de Bruijn sequences. As for encoding, m-sequences may be employed to give de Bruijn sequences that may be rapidly generated over prime power sized alphabets. Arbitrary alphabet sizes may be achieved using the method at the end of Remark 4.19. It seems plausible that this method may be improved upon by finding a means of rapidly generating the spaced de Bruijn sequences constructed in Chapter 4. Alternatively, there exists an efficient algorithm ([3]) that may be used to rapidly generate those binary de Bruijn sequences that arise from Lempel's cycle joining construction. It may be of interest to investigate whether or not this algorithm can be adapted to generate those sequences over arbitrary alphabets that may be formed using the GL construction of Chapter 2.

Chapter 5 studies the construction of sequences with at least a given minimum distance between n-windows. A number of open problems arise. An obvious problem concerns the existence of primitive polynomials whose coefficients have the properties described in Section 5.6. Another is to determine upper bounds on the length such a sequence may have, since the bounds of Lemma 5.2

are trivial.

Notice that we have concentrated throughout on periodic sequences. As discussed in Section 1.1, these have the advantage that they may be applied in both cyclic and linear position location systems. It is simple to show that if every n-tuple appears as a window in a finite, aperiodic sequence, then the first and last $(n-1)$-windows of the sequence are identical. Thus in this case the sequence is equivalent to the (periodic) de Bruijn sequences we have studied. However, when all possible n-tuples do not appear as windows, such as when the n-windows are required to form an error control code, this is not necessarily the case. Thus it may be worthwhile to sacrifice periodicity and investigate the window properties of finite sequences, to see what advantages, if any, such sequences may offer.

In Section 5.8 we gave an example of how a combinatorial technique previously applied to constructing de Bruijn sequences may also be applied to our code window sequences. Clearly, there are many other existing constructions which may, or may not, be suitable for similar adaptation. Of particular interest would be to adapt the interleaving technique to the formation of code window sequences. Considering Construction 3.1 in detail, whilst the interleaving stage itself seems relatively straightforward, it seems difficult to adapt the earlier stage in which the starting sequence $[t]$ is adapted, by the insertion and deletion of certain digits, to form the sequences $[a]$ and $[b]$. This is because $[t]$ contains $0_{|n|}$ and $1_{|n|}$, and these are the only n-windows of $[t]$ affected by these operations. If $[t]$ had a minimum distance of at least 2 between its n-windows, then for reasons like those in Remark 5.12, it could not have any n-windows

of consecutive equal digits. Thus the insertion or deletion of individual digits would affect many more n-windows of $[t]$, and perhaps also affect the distance properties of these windows. It would be interesting to determine whether these problems may be overcome.

Perhaps the most promising and potentially useful subject for future research is that introduced in Section 5.9. We have given one application in which the arrays described therein could be profitably employed, and so construction of these arrays is a desirable goal. In particular, the author notes that techniques based on [17], including Construction 5.49, suffer from the drawback that for reasonably-sized windows the overall array has a high ratio of width to height. This is because the value p in Construction 5.49 is bounded by c^m, whilst $[t]$ may have period q of the order of p^n. The method of [37] using m-sequences has similar drawbacks. Despite this, the author feels that the structure inherent in the arrays constructed by this latter method makes them a particularly strong candidate for further investigation.

Bibliography

[1] T. Aardenne-Ehrenfest and N. G. de Bruijn, "Circuits and Trees in Oriented Linear Graphs", *Simon Stevin* **28** 1951, 203–17, also in *Classic Papers in Combinatorics* (I. Gessel and G. Rota eds.) Birkhauser, Boston, 1987, 149–163.

[2] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, 1974.

[3] F. S. Annexstein, "Generating de Bruijn Sequences: An Efficient Implementation" *IEEE Trans. Computers* **46** (2) 1997, 198–200.

[4] T. C. Bartee and P. W. Wood, "Coding for Tracking Radar Ranging", MIT Lincoln Lab., Lexington, Mass., Tech. Report 318, June 11 1963.

[5] S. R. Blackburn, T. Etzion and K. G. Paterson, "Permutation Polynomials, de Bruijn Sequences and Linear Complexity", *J. Combin. Theory Ser. A* **76** 1996, 55–82.

[6] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, Elsevier, 1976.

[7] N. G. de Bruijn, "A Combinatorial Problem", *Nederl. Akad. Wetensch., Proc.* **49** 1946, 758–64.

[8] J. Burns and C. J. Mitchell, "Coding Schemes for Two-Dimensional Position Sensing", *Cryptography and Coding III* (M. J. Ganley ed.), Oxford University Press, 1993.

[9] A. H. Chan, R. A. Games and E. L. Key, "On the Complexities of de Bruijn Sequences", *J. Combin. Theory Ser. A* **33** 1982, 233–46.

[10] F. Chung, P. Diaconis and R. Graham, "Universal Cycles for Combinatorial Structures", *Discrete Mathematics* **110** 1992, 43–59.

[11] S. D. Cohen, "Primitive Elements and Polynomials with Arbitrary Trace", *Discrete Mathematics* **83** 1990, 1–7.

[12] P. M. Cohn, *Algebra*, volume 1, 2nd ed., Wiley, 1982.

[13] T. Etzion, "On the Distribution of de Bruijn Sequences of Low Complexity", *J. Combin. Theory Ser. A* **38** (2) 1985, 241–53.

[14] T. Etzion, "An Algorithm for Constructing m-ary de Bruijn Sequences", *J. Algorithms* **7** 1986, 331–40.

[15] T. Etzion, "An Algorithm for Generating Shift-Register Cycles", *Theoretical Computer Science* **44** 1986, 209–224.

[16] T. Etzion, "On the Distribution of de Bruijn CR-Sequences", *IEEE Trans. Inform. Theory* **32** (3) 1986, 422-3.

[17] T. Etzion, "Constructions for Perfect Maps and Pseudo-Random Arrays", *IEEE Trans. Inform. Theory* **34** (5) 1988, 1308–16.

[18] T. Etzion and A. Lempel, "Algorithms for the Generations of Full-Length Shift-Register Sequences", *IEEE Trans. Inform. Theory* **IT-30** (3) 1984, 480–4.

[19] T. Etzion and A. Lempel, "On the Distribution of de Bruijn Sequences of Given Complexity", *IEEE Trans. Inform. Theory* **IT-30** (4) 1984, 611–4.

[20] T. Etzion and A. Lempel, "Construction of de Bruijn Sequences of Minimal Complexity", *IEEE Trans. Inform. Theory* **IT-30** (5) 1984, 705–9.

[21] C. Flye-Sainte Marie, "Solution to Problem number 58", *l'Intermediare des Mathematiciens* **1** 1894, 107–10.

[22] H. Fredricksen, "A Survey of Full Length Nonlinear Shift Register Cycle Algorithms", *Siam Review* **24** (2) 1982, 195–221.

[23] S. A. Fredricsson, "Pseudo-Randomness Properties of Binary Shift Register Sequences", *IEEE Trans. Inform. Theory* **IT-21** (1) 1975, 115–20.

[24] R. A. Games, "There are no de Bruijn Sequences of Span n with Complexity $2^{n-1}+n+1$", *J. Combin. Theory Ser. A* **34** 1983, 248–251.

[25] R. A. Games, "A Generalized Recursive Construction for de Bruijn Sequences", *IEEE Trans. Inform. Theory* **IT-29** (6) 1983, 843–50.

[26] I. J. Good, "Normal Recurring Decimals", *J. London Math. Soc.* **21** 1947, 167–169.

[27] W. B. Han, "The Coefficients of Primitive Polynomials over Finite Fields", *Mathematics of Computation* **65** (213) 1996, 31–40.

[28] G. H. Hurlbert, C. J. Mitchell and K. G. Paterson, "On the Existence of de Bruijn Tori with Two by Two Windows", *J. Combin. Theory Ser. A* **76** (2) 1996, 213–30.

[29] V. Klee, "The use of circuit codes in analog-to-digital conversion", in *Graph Theory and its Applications* (B. Harris ed.) Academic Press, New York, 1970.

[30] P. V. Kumar and V. K. Wei, "Minimum Distance of Logarithmic and Fractional Partial m-Sequences", *IEEE Trans. Inform. Theory* **38** (5) 1992, 1474–82.

[31] A. Lempel, "m-ary Closed Sequences", *J. Combin. Theory* **10** 1971, 253–8.

[32] A. Lempel, "On a Homomorphism of the de Bruijn Graph and its Application to the Design of Feedback Shift Registers", *IEEE Trans. Computers* **C-19** (12) 1970, 1204–9.

[33] R. Lidl and H. Niederreiter, *Finite Fields,* Encyclopedia of Mathematics and its Applications **20**, Cambridge University Press, 1983.

[34] J. H. Lindholm, "An Analysis of the Pseudo-Randomness Properties of Subsequences of Long m-Sequences", *IEEE Trans. Inform. Theory* **IT-14** (4) 1968, 569–76.

[35] J. L. Massey, "Shift Register Synthesis and BCH Decoding", *IEEE Trans. Inform. Theory* **IT-15** 1969, 122–7.

[36] J. L. Massey and R. Liu, "Equivalence of Non-Linear Shift Registers", *IEEE Trans. Inform. Theory* **IT-10** 1964, 378–9.

[37] F. J. MacWilliams and N. J. A. Sloane, "Pseudo-Random Sequences and Arrays", *Proc. of the IEEE* **64** (12) 1976, 1715–29.

[38] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, Elsevier, Amsterdam, 1977.

[39] C. J. Mitchell, "Constructing c-ary Perfect Factors", *Designs, Codes and Cryptography* **4** 1994, 341–68.

[40] C. J. Mitchell, "New c-ary Perfect Factors in the de Bruijn Graph", *Codes and Ciphers* (P. G. Farrell ed.), Formara Ltd., Southend, 1995, Proceedings of the fourth IMA Conference on Cryptogtraphy and Coding, Cirencester, December 1993, 299–313.

[41] C. J. Mitchell, "De Bruijn Sequences and Perfect Factors", *SIAM J. Discrete Math.* **10** (2) 1997, 270–81.

[42] C. J. Mitchell, T. Etzion and K. G. Paterson, "A Method for Constructing Decodable de Bruijn Sequences", *IEEE Trans. Inform. Theory* **42** (5) 1996, 1472–8.

[43] C. J. Mitchell and K. G. Paterson, "Decoding Perfect Maps", *Designs, Codes and Cryptography* **4** 1994, 11–30.

[44] C. J. Mitchell and K. G. Paterson, "Perfect Factors from Cyclic Codes and Interleaving", *SIAM J. Discrete Math.* (to appear).

[45] O. Moreno, "On Primitive Elements of Trace Equal to 1 in GF(2^m)", *Discrete Mathematics* **41** 1982, 53–6.

[46] A. Odlyzko, "Discrete Logarithms in Finite Fields and their Cryptographic Significance", *Advances in Cryptology: Proc. EUROCRYPT '84* (T. Beth, N. Cot and I. Ingemarsson eds.) Springer-Verlag, 1985, 224–314.

[47] K. G. Paterson, *On Sequences and Arrays with Specific Window Properties*, Ph.D. Thesis, Royal Holloway and Bedford New College, University of London, 1993.

[48] K. G. Paterson, "Perfect Factors in the de Bruijn Graph", *Designs, Codes and Cryptography* **5** 1995 115–38.

[49] K. G. Paterson, "New Classes of Perfect Maps I", *J. Combin. Theory Ser. A* **73** (2) 1996, 302–34.

[50] K. G. Paterson, "New Classes of Perfect Maps II", *J. Combin. Theory Ser. A* **73** (2) 1996, 335–45.

[51] K. G. Paterson and M. J. B. Robshaw, "Storage Efficient Decoding for a Class of Binary de Bruijn Sequences", *Discrete Mathematics* **138** 1995, 327–41.

[52] W. W. Peterson and E. J. Weldon, *Error-Correcting Codes*, 2nd ed., MIT Press, 1972.

[53] E. M. Petriu, "Absolute-Type Position Transducers Using a Pseudorandom Encoding", *IEEE Trans. Instrumentation and Measurement* **IM-36** (4) 1987, 950–5.

[54] E. M. Petriu, "New Pseudorandom/Natural Code Conversion Method", *Electronics Letters* **24** 1988, 1358–9.

[55] E. M. Petriu and J. S. Basran, "On the Position Measurement of Automated Guided Vehicles Using Pseudorandom Encoding", *IEEE Trans. Instrumentation and Measurement* **38** (3) 1989, 799–803.

[56] E. M. Petriu, J. S. Basran and F. C. A. Groen, "Automated Guided Vehicle Position Recovery", *IEEE Trans. Instrumentation and Measurement* **39** (1) 1990, 254–8.

[57] O. Pretzel, *Error-Correcting Codes and Finite Fields*, Clarendon Press, Oxford, 1992.

[58] S. K. Stein, "The Mathematician as an Explorer", *Scientific American*, May 1961, 149–58.

[59] F. M. Vos, G. L. van der Heijde, H. J. W. Spoelder, I. H. M. van Stokkum and F. C. A. Groen, "A New Instrument to Measure the Shape of the Cornea Based on Pseudorandom Color Coding", *IEEE Trans. Instrumentation and Measurement* **46** (4) 1997, 794–7.

[60] S. Wainberg and J. K. Wolf, "Subsequences of Pseudorandom Sequences", *IEEE Trans. Communications* **COM-18** (5) 1970, 606–12.

Appendix A

We give the minimal generating cycles for all the optimal (n, δ)-CW sequence sequences found by direct computer search, as described in Section 5.5. These were obtained by a 'depth first' search technique, written in C++. Two sequences were considered to be equivalent if one could be transformed into the other by any combination of shifting, reversing of order or complementing every bit. One sequence from each equivalence class (chosen from those having least linear complexity) is listed. Where several inequivalent sequences were all optimal, each equivalence class is represented.

The Tables contain the following information:

(1) a unique reference number for use in the main text;

(2) the window size n (the minimum distance between these windows varies between tables, and is stated in the caption);

(3) the least period of the sequence;

(4) one period of the sequence itself;

(5) the minimal polynomial of the sequence;

(6) the linear complexity of the sequence;

(7) the design distance between n-windows of the sequence as may be predicted from the minimal polynomial using Theorem 5.22 (applicable only in the cases where the linear complexity of the sequence is less than n), and

(8) the 'type' of the sequence, as described in Section 5.5.

TABLE A.1. Results of computer search for optimal $(n, 2)$-CW sequences, with complexity $< n$.

Ref.	n	Period	Sequence $[s]$ and Minimal Polynomial	lc$[s]$	Design Dist.	Type
1	5	7	0011101 $x^3 + x^2 + 1$	3	2	M
2	6	14	00001101111001 $x^5 + x^4 + x^3 + 1 =$ $(x + 1)^2(x^3 + x^2 + 1)$	5	1	D
3	7	31	0000110101001000101111101100111 $x^5 + x^4 + x^3 + x + 1$	5	2	M
4	7	31	0000110010011111011100010101101 $x^5 + x^4 + x^3 + x^2 + 1$	5	2	M
5	8	63	000001011111100101010001100111101110101101001101100010010000111 $x^6 + x^4 + x^3 + x + 1$	6	2	M

TABLE A.2. Results of computer search for optimal $(n, 2)$-CW sequences, with complexity $\geqslant n$.

Ref.	n	Period	Sequence $[s]$ and Minimal Polynomial	lc$[s]$	Type
6	5	10	0001011101 $x^6 + x^5 + x + 1 =$ $(x+1)^2(x^4 + x^3 + x^2 + x + 1)$	6	S
7	6	15	000110111100101 $(x^{15} - 1)/(x+1)(x^2 + x + 1)$	12	O
8	7	30	000001001100011101111100101101 $x^{30} + 1$	30	H
9	8	63	000001110010110101000001001000101 011101001111110111100011010011 $(x^{63} + 1)/(x+1)(x^3 + x^2 + 1)$	59	O
10	8	63	000001011011110000111010101110011 111110110010001100010011010101 $(x^{63} + 1)/(x+1)(x^2 + x + 1)$	60	O
11	8	63	000001100010010111001110101011101 001101111110110010001111000010 1 $(x^{63} + 1)/(x+1)$	62	H
12	8	63	000001100011101010110110011010 01 111110111100001011100100010 0101 $(x^{63} + 1)/(x+1)$	62	H
13	8	63	000001101111110011100001010011 00 100101110101011000100011110 1101 $(x^{63} + 1)/(x+1)$	62	H
14	8	63	000001100010010111000010100110 11 111100111010101100100011110 1101 $(x^{63} + 1)/(x+1)$	62	H
15	8	63	000001100011110111111001000100 10 111000010100111010101100110 1101 $(x^{63} + 1)/(x+1)$	62	H

TABLE A.3. Results of computer search for optimal $(n, 3)$-CW sequences, with complexity $< n$.

Ref.	n	Period	Sequence $[s]$ and Minimal Polynomial	lc$[s]$	Design Dist.	Type
16	6	7	0010111 $x^3 + x + 1$	3	3	M
17	7	14	00001001111011 $x^5 + x^2 + x + 1 = (x + 1)^2(x^3 + x + 1)$	5	1	D
18	8	15	000100110101111 $x^4 + x + 1$	4	3	M
19	9	31	0000110010011111011100010101101 $x^5 + x^4 + x^3 + x^2 + 1$	5	3	M
20	9	31	0000111001101111101000100101011 $x^5 + x^4 + x^2 + x + 1$	5	3	M
21	10	62	0000001000110110101100111100010 1111110111001001010011000011101 $x^7 + x^3 + x^2 + 1 = (x + 1)^2(x^5 + x^3 + 1)$	7	1	D

TABLE A.4. Results of computer search for optimal $(n, 3)$-CW sequences, with complexity $\geqslant n$.

Ref.	n	Period	Sequence $[s]$ and Minimal Polynomial	lc$[s]$	Type
22	7	8	00001101 $x^8 + 1$	8	H
23	8	16	0000011011111001 $x^9 + x^8 + x + 1 = (x + 1)^9$	9	S
24	8	16	0000111011110001 $x^9 + x^8 + x + 1 = (x + 1)^9$	9	S
25	8	16	0001001011101101 $x^9 + x^8 + x + 1 = (x + 1)^9$	9	S
26	8	16	0001101011100101 $x^9 + x^8 + x + 1 = (x + 1)^9$	9	S
27	8	16	0000011010111001 $x^{16} + 1$	16	H
28	8	16	0000111011010001 $x^{16} + 1$	16	H
29	9	29	0000011001111101110001010100 1 $(x^{29} + 1)/(x + 1)$	28	H
30	10	45	0000000110111101001111101110001 0 0101011001101 $x^{45} + 1$	45	H

TABLE A.5. Results of computer search for optimal $(n, 4)$-CW sequences, with complexity $< n$.

Ref.	n	Period	Sequence $[s]$ and Minimal Polynomial	lc$[s]$	Design Dist.	Type
31	10	15	000111101011001 $x^4 + x^3 + 1$	4	4	M
32	11	31	00001110011011111010001001010 11 $x^5 + x^4 + x^2 + x + 1$	5	4	M
33	12	62	00000010001101101011001111000 10 11111101110010010100110000111 01 $x^7 + x^3 + x^2 + 1 = (x + 1)^2(x^5 + x^3 + 1)$	7	2	D

TABLE A.6. Results of computer search for optimal $(n, 4)$-CW sequences, with complexity $\geqslant n$.

Ref.	n	Period	Sequence $[s]$ and Minimal Polynomial	lc$[s]$	Type
34	9	11	00011101101 $(x^{11} + 1)/(x + 1)$	10	H
35	10	22	0000111011011110001001 $x^{12} + x^{11} + x + 1 =$ $(x + 1)^2(x^{10} + x^9 + \cdots + 1)$	12	S
36	11	28	0000110100011011110010111001 $x^{15} + x^{14} + x + 1 =$ $(x + 1)^3(x^3 + x + 1)^2(x^3 + x^2 + 1)^2$	15	S
37	12	40	0000001100101101110010011010 10 1111100101 $(x^{40} + 1)/(x + 1)$	39	H

TABLE A.7. Results of computer search for optimal $(n, 5)$-CW sequences, with complexity $< n$.

Ref.	n	Period	Sequence $[s]$ and Minimal Polynomial	lc$[s]$	Design Dist.	Type
38	11	12	000001101011 $x^6 + x^5 + x^4 + x^2 + x + 1 =$ $(x+1)^2(x^4 + x^2 + 1)$	6	2	O
39	11	12	000011011101 $x^8 + x^4 + 1$	8	1	O
40	11	12	000011101001 $x^8 + x^6 + x^2 + 1 =$ $(x+1)^4(x^4 + x^2 + 1)^2$	8	1	O
41	11	12	001001110101 $x^8 + x^4 + 1 =$ $(x^4 + x^2 + 1)^4$	8	1	O
42	11	12	000111001001 $x^{10} + x^9 + x^7 + x^6 + x^4 + x^3 + x + 1 =$ $(x+1)^4(x^2 + x + 1)^3$	10	1	O
43	11	12	000011001101 $x^{10} + x^9 + x^7 + x^6 + x^4 + x^3 + x + 1 =$ $(x+1)^4(x^2 + x + 2)^3$	10	1	O
44	12	15	000111101011001 $x^4 + x^3 + 1$	4	5	M
45	13	31	0000111001101111101000100101011 $x^5 + x^4 + x^2 + x + 1$	5	5	M
46	14	62	00000011110100110001101011101101111111000010110011100101000100 1 $x^7 + x^6 + x^3 + 1 =$ $(x+1)^2(x^5 + x^4 + x^3 + 1)$	7	3	D

TABLE A.8. Results of computer search for optimal $(n, 5)$-CW sequences, with complexity $\geqslant n$.

Ref.	n	Period	Sequence $[s]$ and Minimal Polynomial	lc$[s]$	Type
47	10	11	00011101101 $(x^{11}+1)/(x+1)$	10	H
48	11	22	00001110111011110001001 $x^{12}+x^{11}+x+1=$ $(x+1)^2(x^{10}+x^9+\cdots+1)$	12	S
49	12	26	00000110010101111100110101 $x^{14}+x^{13}+x+1=$ $(x+1)^2(x^{12}+x^{11}+\cdots+x+1)$	14	S
50	12	26	00001101110101111001000101 $x^{14}+x^{13}+x+1=$ $(x+1)^2(x^{12}+x^{11}+\cdots+x+1)$	14	S
51	13	28	0000001100101011111100110101 $x^{15}+x^{14}+x+1=$ $(x+1)^3(x^3+x+1)^2(x^3+x^2+1)^2$	15	S
52	13	28	0000010011101011111011000101 $x^{15}+x^{14}+x+1=$ $(x+1)^3(x^3+x+1)^2(x^3+x^2+1)^2$	15	S
53	13	28	0000011001010011111001101011 $x^{15}+x^{14}+x+1=$ $(x+1)^3(x^3+x+1)^2(x^3+x^2+1)^2$	15	S
54	13	28	0000011001101011111001100101 $x^{15}+x^{14}+x+1=$ $(x+1)^3(x^3+x+1)^2(x^3+x^2+1)^2$	15	S
55	13	28	0000011101101011111000100101 $x^{15}+x^{14}+x+1=$ $(x+1)^3(x^3+x+1)^2(x^3+x^2+1)^2$	15	S
56	13	28	0000110100011011110010111001 $x^{15}+x^{14}+x+1=$ $(x+1)^3(x^3+x+1)^2(x^3+x^2+1)^2$	15	S
57	13	28	0000011101110101111001000101 $(x+1)^2(x^3+x+1)^2(x^3+x^2+1)^4$	20	O

(Continued.)

TABLE A.8. (Continued.)

Ref.	n	Period	Sequence $[s]$ and Minimal Polynomial	lc$[s]$	Type
58	13	28	0000100010111101101011101001 $(x^{28}+1)/(x+1)$	27	H
59	13	28	0000100101000101101111011101 $(x^{28}+1)/(x+1)$	27	H
60	13	28	0000010011101001111011000101 $x^{28}+1$	28	H
61	13	28	0000011001101011111000100101 $x^{28}+1$	28	H
62	13	28	0000001100101001111100110101 $x^{28}+1$	28	H
63	13	28	0000011101101011101000100101 $x^{28}+1$	28	H
64	14	40	0000011001111011011101100001111 1000100101 $(x^{40}+1)/(x+1)$	39	H
65	14	40	0000010001111011011101100001111 1000100101 $x^{40}+1$	40	H

Appendix B

We list the primitive polynomials described in Section 5.6.

For polynomials over $\mathbb{F}_2$, we adopt the notation used in the Peterson Tables ([52, Appendix C]). Each of the decimal digits listed should be converted into its binary equivalent, with the least significant digit to the right. This binary expansion represents a list of the coefficients of the required polynomial, with the highest order coefficient first.

We represent polynomials over other fields by a list of their coefficients, with the highest order coefficient first. For odd prime fields $\mathbb{F}_p$, the field elements are simply integers modulo p.

Table B.1. Primitive polynomials over $\mathbb{F}_2$ not having two consecutive zero coefficients.

Degree	Coefficients	Degree	Coefficients	Degree	Coefficients
3	13	18	1573657	33	137537353353
4	—	19	3732735	34	372753752655
5	75	20	5267667	35	765266657267
6	155	21	16653273	36	1357326533377
7	253	22	25332733	37	2726755655753
8	537	23	55372765	38	7332733576675
9	1725	24	126775673	39	15765265525755
10	3575	25	255765727	40	36765257376673
11	6673	26	577757675	41	57335557657335
12	16533	27	1335337277	42	127357757333733
13	27537	28	3335275655	43	353372735257737
14	75653	29	7373777535	44	555527655675535
15	177775	30	15772532565	45	1325753327266753
16	373665	31	37777777767		
17	525327	32	67577575265		

TABLE B.2. Primitive polynomials over $\mathbb{F}_2$ not having three consecutive equal coefficients.

Degree	Coefficients	Degree	Coefficients	Degree	Coefficients
3	13	18	1315323	33	146653224533
4	23	19	2644655	34	231226633233
5	45	20	5114511	35	546311555245
6	155	21	13225555	36	1544514546313
7	253	22	22222223	37	3126646326315
8	453	23	52325453	38	5152466522451
9	1131	24	115532451	39	13251152654631
10	2553	25	315325515	40	23266224465313
11	4445	26	652452245	41	62246646646653
12	13131	27	1451553251	42	144452322462225
13	24623	28	3253114513	43	255155551551453
14	65453	29	6332263255	44	546555131322655
15	111511	30	14622324465	45	1122552523326245
16	255125	31	22551252231		
17	525251	32	52646453223		

TABLE B.3. Primitive polynomials over $\mathbb{F}_3$ not having any zero coefficients.

Degree	Coefficients	Degree	Coefficients
1	11	11	111222221221
2	112	12	1122222121222
3	1211	13	12111222121211
4	11222	14	111222121122122
5	121111	15	1121122111122121
6	1211212	16	12121212121222112
7	12212221	17	121212121121212211
8	121221212	18	1121111221111122122
9	1111112221	19	11122222221222122221
10	11222121212	20	111121112121211112122

TABLE B.4. Primitive polynomials over $\mathbb{F}_5$ not having any zero coefficients.

Degree	Coefficients	Degree	Coefficients
1	12	11	144222113142
2	112	12	1434323214322
3	1143	13	12223413221132
4	14143	14	114231422114213
5	114442	15	1223211344311233
6	1234242	16	11323111113413443
7	12144112	17	133321211213333312
8	134141322	18	1344421444131431223
9	1243442443	19	12122431441413242123
10	11231433343	20	111211443443422122112

We represent the elements of $\mathbb{F}_4$ by $\{0, 1, \alpha, \alpha^2\}$, where $\alpha^2 = \alpha + 1$.

TABLE B.5. Primitive polynomials over $\mathbb{F}_4$ not having any zero coefficients.

Degree	Coefficients	Degree	Coefficients
1	11	6	$1\alpha\alpha^2 1\alpha^2\alpha 1$
2	111	7	$1\alpha^2\alpha\alpha^2\alpha\alpha^2\alpha^2 1$
3	1111	8	$1\alpha\alpha 1\alpha^2\alpha^2\alpha^2\alpha^2 1$
4	$1\alpha 111$	9	$1\alpha^2\alpha^2\alpha^2 1\alpha\alpha\alpha 11$
5	$11\alpha^2 111$	10	$11\alpha\alpha\alpha 1\alpha^2 11\alpha 1$

We represent the elements of $\mathbb{F}_8$ by $\{0, 1, \alpha, \alpha^2, \ldots, \alpha^6\}$, where $\alpha^3 = \alpha + 1$.

TABLE B.6. Primitive polynomials over $\mathbb{F}_8$ not having any zero coefficients.

Degree	Coefficients	Degree	Coefficients
1	11	6	$1\alpha\alpha^6\alpha^3\alpha^6\alpha^6 1$
2	111	7	$1\alpha^3\alpha^6\alpha^4 1\alpha^6\alpha^4 1$
3	1111	8	$1\alpha\alpha^2\alpha^4\alpha^6\alpha^4\alpha^6\alpha^5 1$
4	11111	9	$1\alpha^3 1\alpha^5\alpha^6\alpha^2\alpha^6 1\alpha 1$
5	$111\alpha\alpha 1$	10	$1\alpha^2\alpha^5\alpha\alpha^2 1\alpha\alpha^4 1\alpha^5 1$