Decoding Perfect Maps

Chris J. Mitchell* Kenneth G. Paterson†

19th March 1993

Abstract

Perfect Maps are two-dimensional arrays in which every possible sub-array of a certain size occurs exactly once. They are a generalisation of the de Bruijn sequences to two dimensions and are of practical significance in certain position location applications. In such applications the decoding problem, i.e. resolving the position of a particular sub-array within a specified Perfect Map, is of great significance. In this paper new constructions for (binary) Perfect Maps and 2^k-ary de Bruijn sequences are presented. These construction methods, although not yielding Perfect Maps for new sets of parameters, are significant because the Maps they yield can be efficiently decoded.

1 Introduction

Perfect maps, i.e. two-dimensional arrays in which every possible rectangular sub-array (of fixed size) occurs precisely once, have been studied for some 30 years (see, for example, Reed and Stewart's 1962 paper, [23]). A number of construction methods have been devised, [6, 7, 12] and the existence question has recently been completely answered [15].

A number of possible applications exist for such arrays, perhaps the most obvious of which is their use for two-dimensional position location. The basic idea is that, if such a map is written in some way onto a planar surface, then any device capable of examining an appropriately sized rectangular sub-array will be able to precisely determine its position on the

*Computer Science Department, Royal Holloway, University of London, Egham Hill, Egham, Surrey TW20 0EX, England.

†Mathematics Department, Royal Holloway, University of London. Funded by SERC CASE award No. 90C/11574

surface. Brief mention is made of such an application by Reed and Stewart, [23], and a more detailed description of applications of this type can be found in Burns and Mitchell, [3]. Before proceeding also note that, in the one-dimensional case, similar position-detection applications have been suggested for de Bruijn and m-sequences by a number of authors (see, for example, Bondy and Murty, [2], Petriu et al., [17, 18, 19, 20, 21, 22] and Arazi, [1]).

It is worth pointing out that the position detection application does not require the 'extreme' property of Perfect Maps, namely that each sub-array occurs exactly once. The key requirement is that each sub-array occurs at most once. This logic has led some authors to apply the term Perfect Map in a rather looser way to a much larger class of arrays (see, for example, Reed and Stewart, [23]). Of particular importance in this context are the *pseudorandom arrays* (see, for example, Nomura et al., [14], MacWilliams and Sloane, [13] and Etzion, [6]) which have the property that each sub-array, apart from the all-zero sub-array, occurs exactly once. Other arrays with the property that each sub-array occurs at most once have been constructed by Dénes and Keedwell, [5] and Etzion, [6].

Although the existence problem for such arrays has been addressed by a number of authors, the problem of *decoding* these arrays has virtually been ignored (with the exception of a small amount of work on decoding de bruijn sequences), even though its solution is essential for the position location application. By a decoding algorithm we mean an algorithm for computing the position of a given sub-array within a Perfect Map. In any instance the 'brute force' method is available: we could store a complete look-up table of sub-arrays and their positions. However such a method rapidly becomes infeasible because of prohibitive memory requirements as the arrays increase in size. The only other results known to the authors on the array decoding problem are those of Lloyd and Burns, [11] who present a decoding technique for a certain special class of (binary) pseudorandom arrays.

This paper is concerned with the construction of a class of Perfect Maps for which a decoding algorithm exists which is significantly more efficient than 'brute force' decoding. Whilst constructions and possible applications exist for c-ary Perfect Maps with $c > 2$, where a c-ary array is one with its elements drawn from the set $\{0, 1, \ldots, c-1\}$, in this paper we are primarily concerned with the binary case, i.e. where each array contains only elements from the set $\{0, 1\}$.

We start by giving a description of the method of construction of these arrays, which actually form a subset of the Perfect Maps of Etzion [6]. His constructions for Perfect Maps rely on ordering special sets of sequences and their shifts to form the columns of an array. We give an explicit description of these ordering and shifting processes in terms of 2^k-ary de Bruijn sequences.

Our description naturally leads to a decoding method which reduces the problem to the decoding of 2^k-ary de Bruijn sequences and other related sequences. We next give new constructions for classes of such sequences which allow efficient decoding. More specifically, we present a decoding algorithm which reduces the decoding of one of these 2^k-ary de Bruijn sequences to repeated decodings of a binary de Bruijn sequence of the same span. We then briefly consider some techniques for the decoding of binary de Bruijn sequences. We conclude by combining the decoding of our Perfect Maps and that of our sequences to produce an efficient Perfect Map decoding algorithm. We indicate the complexity of the proposed decoding technique and compare it with the 'brute force' approach. We first give the necessary definitions and briefly review the known results on Perfect Maps.

2 Formal definitions and notation

In this paper we consider c-ary m by n integer arrays, which we write as

$$A = (a_{ij}), \ \ (0 \leq i \leq m-1, 0 \leq j \leq n-1)$$

where each entry a_{ij} satisfies $0 \leq a_{ij} \leq c-1$.

If A is an m by n c-ary array, then we define its u by v sub-arrays to be the set of c-ary arrays

$$A_{st} = (a_{ij}^{(st)}, 0 \leq i \leq u-1, 0 \leq j \leq v-1), \ \ 0 \leq s \leq m-1, 0 \leq t \leq n-1$$

defined by

$$a_{ij}^{(st)} = a_{i+s,j+t}$$

where $i+s$ is computed modulo m and $j+t$ is computed modulo n. Note that throughout this paper we consider arrays and sequences 'periodically', i.e. we treat them as if they are written onto the surface of a torus or a circle respectively.

Using the notation of Fan et al., [7], we can then define a $(m, n; u, v)$-*Perfect Map*, or simply a $(m, n; u, v)$-PM to be a c-ary m by n array ($m \geq u$, $n \geq v$) with the property that each possible u by v c-ary array occurs exactly once in the set of u by v sub-arrays $\{A_{st} : 0 \leq s \leq m-1, 0 \leq t \leq n-1\}$. Note that $(1, n; 1, v)$-PMs are simply the well-known de Bruijn sequences.

We immediately have the following result relating the parameters of a Perfect Map.

Lemma 2.1 *If A is a c-ary $(m, n; u, v)$-PM then*

(i) $m > u$ *or* $m = u = 1$,

(ii) $n > v$ *or* $n = v = 1$, *and*

(iii) $mn = c^{uv}$.

Proof (i) is immediate on observing that if $m = u$ then any $m \times n$ array must contain the all-zero $u \times v$ sub-array either not at all or at least u times. (ii) is similar and (iii) follows directly from the definition of perfect map. □

As an immediate corollary we can deduce that, for the binary case (i.e. $c = 2$), we must have $m = 2^k$ and $n = 2^{uv-k}$ for some integer k.

It has recently been shown by Paterson [15], that, in the binary case, the necessary conditions of the previous lemma are in fact sufficient for the existence of Perfect Maps.

We conclude these remarks by briefly reviewing the one-dimensional analogue of Perfect Maps and Pseudorandom Arrays. As we have already observed, a $(1, n; 1, v)$-PM is simply a *de Bruijn sequence*. We then immediately have the following well-known result, [4, 9, 24].

Theorem 2.2 (De Bruijn, Good, Rees) *A c-ary span v de Bruijn sequence (i.e. a sequence of length* $n = c^v$ *with entries from* $\{0, 1, \ldots, c-1\}$ *in which every distinct c-ary v-tuple occurs exactly once) exists for every c and v* ($c \geq 2$ *and* $v \geq 1$).

In fact, the precise number of distinct de Bruijn sequences is also known. Many construction methods have been devised for de Bruijn sequences, see, for example, [8].

The one-dimensional analogue of a pseudorandom array is what we call a *pseudorandom sequence*, and is a c-ary sequence of length $n = c^v - 1$ with the property that every c-ary v-tuple occurs, with the exception of the all-zero v-tuple. The following result is well-known (see, for example, [3]).

Lemma 2.3 *There exists a* $(c-1)$*-to-one correspondence between the set of c-ary span v de Bruijn sequences and the set of c-ary span v pseudorandom sequences.*

Given a span v de Bruijn sequence, the corresponding pseudorandom sequence is derived by deleting one of the zeros from the unique v-tuple of zeros. Hence pseudorandom sequences exist for every choice of c and v. Note that we shall use the term *derived pseudorandom sequence* throughout to refer to the sequence obtained from a de Bruijn sequence by this zero deletion process.

We shall be mainly interested in 2^k-ary de Bruijn and pseudorandom sequences; we often choose to represent the elements of such sequences by their binary expansions i.e. by binary k-tuples.

3 A construction method for Perfect Maps

Before describing the method of construction we require one additional piece of notation. Given a finite sequence $C = (c_i)$, $(0 \leq i \leq n-1)$, and a non-negative integer k, we define $\mathbf{T}_k(C)$ to be the *cyclic shift* of C by k places. I.e. if we write $(d_i) = \mathbf{T}_k(C)$ then

$$d_{i+k} = c_i, \quad (0 \leq i \leq n-1)$$

where $i + k$ is calculated modulo n.

We also need to consider the existence of *Perfect Factors*, introduced by Etzion, [6]. A (u, k)-Perfect Factor consists of a collection of 2^{u-k} binary sequences (*cycles*) of length 2^k, with the property that every binary u-tuple occurs in a unique sequence in the collection. Note that a (u, u)-Perfect Factor is simply a binary, span u de Bruijn sequence. Etzion (Theorem 4 of [6]) established the following key result.

Theorem 3.1 *If u and k are positive integers satisfying*

$$u < 2^k \leq 2^u$$

then there exists a (u, k)-Perfect Factor.

We can now describe the construction method central to this paper.

Construction 3.2 *Suppose u, v and k are positive integers satisfying*

$$u + 1 \leq 2^k \leq 2^u$$

and

$$uv \geq 2k + 1.$$

Suppose $C_0, C_1, \ldots C_{2^{u-k}-1}$ are the 2^{u-k} cycles of length 2^k of a (u, k)-Perfect Factor (such a Perfect Factor exists by Theorem 3.1). Let (r_i), $(0 \leq i < 2^{uv-k})$, be $2^{k(v-1)}$ repetitions of a 2^{u-k}-ary span v de Bruijn sequence (such sequences always exist by Theorem 2.2). Suppose also that (s_i), $(0 \leq i < 2^{uv-k})$, is $2^{v(u-k)}$ repetitions of a 2^k-ary span $v-1$ pseudorandom sequence for which the first $v-2$ elements are all zeros, preceded by $2^{v(u-k)}$ zeros (again such sequences always exist by Theorem 2.2 and Lemma 2.3). Finally, define the sequence (w_i), $(0 \leq i < 2^{uv-k})$ by

$$w_i = \sum_{j=0}^{i-1} s_j \bmod 2^k,$$

where $w_0 = 0$.

Now define a $2^k \times 2^{uv-k}$ array by letting it have column i $(0 \leq i \leq 2^{uv-k}-1)$ be equal to $\mathbf{T}_{w_i}(C_{r_i})$, i.e. the ith column consists of the cycle C_{r_i} of the chosen Perfect Factor cyclically shifted by w_i places.

Before proving that the above construction yields Perfect Maps, we need the following.

Lemma 3.3 *The sequence (w_i) defined in Construction 3.2 satisfies*

$$w_{2^{uv-k}-1} + s_{2^{uv-k}-1} \equiv w_0 \equiv 0 \pmod{2^k}.$$

Proof By definition

$$w_{2^{uv-k}-1} \equiv \sum_{j=0}^{2^{uv-k}-2} s_j \pmod{2^k}.$$

But (s_i) is nothing more than $2^{v(u-k)}$ repetitions of a 2^k-ary span $v-1$ pseudorandom sequence (preceded by a collection of zeros). Hence

$$\sum_{j=0}^{2^{uv-k}-2} s_j = 2^{v(u-k)}S - s_{2^{uv-k}-1}$$

where S is the sum of the elements of such a pseudorandom sequence. It should be clear that a 2^k-ary span $v-1$ pseudorandom sequence contains each of the elements of the set $\{0, 1, \ldots, 2^k-1\}$ precisely $2^{k(v-2)}$ times each and so

$$S = 2^{k(v-2)}(1 + 2 + \ldots + (2^k - 1)) = 2^{kv-k-1}(2^k - 1)$$

and thus

$$w_{2^{uv-k}-1} + s_{2^{uv-k}-1} \equiv 2^{v(u-k)}2^{kv-k-1}(2^k-1) \equiv 2^{uv-k-1}(2^k-1) \pmod{2^k}.$$

But $uv - k - 1 \geq k$ by definition and the result follows. □

Theorem 3.4 *Suppose u, v and k are positive integers satisfying*

$$u + 1 \leq 2^k \leq 2^u$$

and

$$uv \geq 2k + 1.$$

Then a $2^k \times 2^{uv-k}$ array A obtained using Construction 3.2 is a $(2^k, 2^{uv-k}; u, v)$-PM.

Remark Observe that the condition

$$uv \geq 2k + 1$$

is trivially satisfied given $v > 1$ (except for the single case $v = 2$ and $u = k$).

Proof Suppose D is a $u \times v$ binary array — we need to show that this array occurs somewhere within the $2^k \times 2^{uv-k}$ array A. Consider the v columns $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{v-1}$ of D. Each of these columns is a u-tuple and hence each $\mathbf{d}_i$ occurs within (a unique) one of the cycles $C_0, C_1, \ldots C_{2^{u-k}-1}$ (since they form a (u,k)-Perfect Factor); suppose that $\mathbf{d}_i$ occurs in C_{e_i} $(0 \leq i \leq v-1)$. Moreover suppose that the first element of $\mathbf{d}_i$ is located at element f_i of C_{e_i} $(0 \leq f_i \leq 2^k - 1,\ 0 \leq i \leq v-1)$.

Suppose we can find a set of v consecutive columns of A equal to

$$\mathbf{T}_{g_0}(C_{e_0}), \mathbf{T}_{g_1}(C_{e_1}), \ldots, \mathbf{T}_{g_{v-1}}(C_{e_{v-1}})$$

where $g_0, g_1, \ldots, g_{v-1}$ is some sequence satisfying

$$g_i - g_0 \equiv f_i - f_0 \pmod{2^k},\ (1 \leq i \leq v-1).$$

Then it should be clear that these v columns will contain a copy of D. We conclude the proof of the theorem by exhibiting the existence of such a set of columns within A.

First note that, from the definition of Construction 3.2, the sequence (r_i), $(0 \leq i < 2^{uv-k})$, is $2^{k(v-1)}$ repetitions of a 2^{u-k}-ary span v de Bruijn sequence. Now, by definition of de Bruijn sequence, this means that each repetition of the de Bruijn sequence making up (r_i) will contain the v-tuple $(e_0, e_1, \ldots, e_{v-1})$ exactly once. Hence, ignoring for the moment the shift values, the array A will contain the v-tuple of columns $(C_{e_0}, C_{e_1}, \ldots, C_{e_{v-1}})$ exactly $2^{k(v-1)}$ times at even intervals (of length $2^{v(u-k)}$). Suppose the first occurrence of this v-tuple of columns is at column numbers $t, t+1, \ldots, t+v-1$ (where t satisfies $0 \leq t < 2^{v(u-k)}$), and hence all subsequent appearances are at column numbers $2^{v(u-k)}y + t, 2^{v(u-k)}y + t + 1, \ldots, 2^{v(u-k)}y + t + v - 1$ $(0 \leq y < 2^{k(v-1)})$, and where column numbers are reduced modulo 2^{uv-k} as necessary.

The shift values applied to each appearance of the v-tuple of columns $(C_{e_0}, C_{e_1}, \ldots, Ce_{v-1})$ are determined by the sequence (w_i), $(0 \leq i < 2^{uv-k})$. We are thus interested in the values of

$$w_{dy+t}, w_{dy+t+1}, \ldots, w_{dy+t+v-1},\ (0 \leq y < 2^{k(v-1)}),$$

where $d = 2^{v(u-k)}$. More specifically, we are concerned with the values of

$$w_{dy+t+1} - w_{dy+t}, \ldots, w_{dy+t+v-1} - w_{dy+t},\ (0 \leq y < 2^{k(v-1)}),$$

since we need to find such a tuple of values which equals

$$f_1 - f_0, \ldots, f_{v-1} - f_0$$

for some y $(0 \leq y < 2^{k(v-1)})$. But, by definition of (w_i), the above tuples of values are equal to

$$s_{dy+t}, s_{dy+t} + s_{dy+t+1}, \ldots, s_{dy+t} + s_{dy+t+1} + \cdots + s_{dy+t+v-2}$$

for every y ($0 \leq y < 2^{k(v-1)}$). Note that this applies even when $dy+t+j \geq 2^{uv-k}$ since $w_{2^{uv-k}-1} + s_{2^{uv-k}-1} \equiv 0 \pmod{2^k}$ by Lemma 3.3.

Next observe that (s_i) consists of d repetitions of a pseudorandom sequence of period $2^{k(v-1)} - 1$ (preceded by d zeros), and, by definition, this pseudorandom sequence will contain every 2^k-ary non-zero $(v-1)$-tuple of values precisely once. Now since $2^{k(v-1)} - 1$ is co-prime to d, the tuples

$$s_{dy+t}, s_{dy+t+1}, \ldots, s_{dy+t+v-2}$$

will range through the entire set of (non-zero) 2^k-ary $(v-1)$-tuples as y ranges from 1 up to $2^{k(v-1)}-1$ (this remains true even when $dy+t+j \geq 2^{uv-k}$ since we assumed that the first $v-2$ elements of the pseudorandom sequence used to constitute (s_i) are all zeros). Hence the tuples

$$s_{dy+t}, s_{dy+t} + s_{dy+t+1}, \ldots, s_{dy+t} + s_{dy+t+1} + \cdots + s_{dy+t+v-2}$$

will also range through the entire set of (non-zero) 2^k-ary $(v-1)$-tuples as y ranges from 1 up to $2^{k(v-1)} - 1$. Finally observe that, when $y = 0$, the tuple

$$s_{dy+t}, s_{dy+t+1}, \ldots, s_{dy+t+v-2}$$

will be identically zero (by definition — again noting that this holds since the first $v-2$ elements of the pseudorandom sequence are all zero) and hence the tuple

$$s_t, s_t + s_{t+1}, \ldots, s_t + s_{t+1} + \cdots + s_{t+v-2}$$

will also be identically zero.

Thus, as y ranges from 0 up to $2^{k(v-1)} - 1$, the $(v-1)$-tuples

$$w_{dy+t+1} - w_{dy+t}, \ldots, w_{dy+t+v-1} - w_{dy+t}$$

will range through every possible 2^k-ary $(v-1)$-tuple. Hence there will exist precisely one such tuple which equals

$$f_1 - f_0, \ldots, f_{v-1} - f_0.$$

The result now follows. □

Example 3.5 *As an example of the above construction method, consider the case $u = 3$, $v = 2$, $k = 2$ (and hence we construct a $(4, 16; 3, 2)$-PM). We first need a $(3, 2)$-Perfect Factor — which will contain $2^{3-2} = 2$ cycles of length $2^2 = 4$. The following is an example:*

$$C_0 = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}, \quad C_1 = \begin{pmatrix} 0 & 1 & 1 & 1 \end{pmatrix}.$$

We also need a 2-ary span 2 de Bruijn sequence and a 4-ary span 1 pseudorandom sequence, examples of which are provided by

$$\begin{pmatrix} 0 & 0 & 1 & 1 \end{pmatrix}$$

and

$$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$$

respectively. Hence (r_i), which consists of 4 repetitions of the de Bruijn sequence is

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

and (s_i), consisting of 4 repetitions of the pseudorandom sequence preceded by 4 zeros is

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

and hence (w_i) is

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 3 & 2 & 3 & 1 & 0 & 1 & 3 & 2 & 3 & 1 \end{pmatrix}.$$

Using (r_i) and (w_i) as indicated in Construction 3.2 we arrive at the following $(4, 16; 3, 2)$-PM:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Construction 3.2 above generalises Theorem 1 of Ma, [12], whose construction corresponds to the special case $k = u$. Construction 3.2 yields Perfect Maps for every parameter set satisfying the conditions of Theorem 3.4. These are exactly the conditions on the parameters imposed by Etzion in [6]; indeed our Perfect Maps are actually a subset of those produced by the method of Etzion. The main significance of the above construction is that it enables us to find means of decoding these arrays which are significantly more efficient than brute force techniques.

4 The decoding technique

We now describe a decoding algorithm for the above class of Perfect Maps. Suppose A is a $(2^k, 2^{uv-k}; u, v)$-PM obtained from Construction 3.2. The decoding algorithm described here requires the use of decoding algorithms for

the three combinatorial objects used to construct A; we start by defining the function of these algorithms—efficient implementations will be considered in subsequent sections.

Let $C_0, C_1, \ldots, C_{2^{u-k}-1}$ be the (u,k)-Perfect Factor used to construct A. Now suppose that the function

$$F_1 : \{0,1\}^u \to \{0,1,\ldots,2^{u-k}-1\}$$

maps a binary u-tuple onto the number of the cycle of the Perfect Factor which contains it, i.e., given a binary u-tuple $\mathbf{h}$, $F_1(\mathbf{h}) = i$ if and only if C_i contains $\mathbf{h}$. The related function

$$F_2 : \{0,1\}^u \to \{0,1,\ldots,2^k-1\}$$

indicates the position of a u-tuple within the cycle in which it is to be found. More formally, if the binary u-tuple $\mathbf{h}$ is located in a cycle

$$(c_0, c_1, \ldots, c_{2^k-1})$$

such that the first element of $\mathbf{h}$ corresponds to c_j, then $F_2(\mathbf{h}) = j$.

We define functions G and H_1 to respectively be decoding algorithms for the 2^{u-k}-ary span v de Bruijn sequence and the 2^k-ary span $v-1$ pseudorandom sequence used to construct A. Hence the function

$$G : \{0,1,\ldots,2^{u-k}-1\}^v \to \{0,1,\ldots,2^{(u-k)v}-1\}$$

is defined so that, given a 2^{u-k}-ary v-tuple $\mathbf{a}$, $G(\mathbf{a})$ is equal to the position within the de Bruijn sequence at which $\mathbf{a}$ is to be found. Similarly, the function

$$H_1 : \{0,1,\ldots,2^k-1\}^{v-1} \to \{0,1,\ldots,2^{k(v-1)}-2\}$$

is defined so that, given a 2^k-ary $(v-1)$-tuple $\mathbf{b}$, $H_1(\mathbf{b})$ is equal to the position within the pseudorandom sequence at which $\mathbf{b}$ is to be found.

Finally we define the function

$$H_2 : \{0,1,\ldots,2^{k(v-1)}-2\} \to \{0,1,\ldots,2^k-1\}$$

by

$$H_2(j) = \begin{cases} 0 & \text{if } j = 0 \\ \sum_{i=0}^{j-1} \alpha_j \bmod 2^k & \text{if } 1 \le j \le 2^{k(v-1)}-2 \end{cases}$$

where (α_i), $(0 \le i \le 2^{k(v-1)}-2)$ is the 2^k-ary span $v-1$ pseudorandom sequence used in the construction of A.

We now describe how the functions F_1, F_2, G, H_1 and H_2 can be used to decode A. Suppose D is a $u \times v$ binary array with columns $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{v-1}$. Then the following algorithm can be used to locate the position of D within A.

Algorithm 4.1 *The algorithm involves three main steps, as follows.*

1. *Let*
$$g = G(F_1(\mathbf{d}_0), F_1(\mathbf{d}_1), \ldots, F_1(\mathbf{d}_{v-1}))$$
and
$$h = H_1(F_2(\mathbf{d}_0) - F_2(\mathbf{d}_1), F_2(\mathbf{d}_1) - F_2(\mathbf{d}_2), \ldots, F_2(\mathbf{d}_{v-2}) - F_2(\mathbf{d}_{v-1}))$$
(unless
$$F_2(\mathbf{d}_1) - F_2(\mathbf{d}_0) = F_2(\mathbf{d}_2) - F_2(\mathbf{d}_1) = \cdots = F_2(\mathbf{d}_{v-1}) - F_2(\mathbf{d}_{v-2}) = 0$$
in which case set $h = -1$).

2(a). *If $h = -1$ then let $y = g$.*

2(b). *If $h \geq 0$ then, using the Euclidean Algorithm (or by any other means), obtain the unique solution y' (in the range 0 to $2^{uv-k} - 2^{v(u-k)} - 1$) to the simultaneous congruences:*
$$y' \equiv g \pmod{2^{v(u-k)}},$$
$$y' \equiv h \pmod{2^{k(v-1)} - 1};$$
finally let $y = y' + 2^{v(u-k)}$.

The value y will now be equal to the number of the column of A in which the first column of D appears.

3(a). *If $h = -1$ then let*
$$x = F_2(\mathbf{d}_0).$$

3(b). *If $h \geq 0$ then suppose $y' = \lambda(2^{k(v-1)} - 1) + h$. Then let*
$$x = \lambda 2^{kv-k-1}(2^k - 1) + H_2(h) + F_2(\mathbf{d}_0) \bmod 2^k.$$

The value x will now be equal to the number of the row of A in which the first row of D appears. This completes the decoding process.

Proof of correctness Suppose X and Y are the numbers of the row and column in A (respectively) in which the first row and first column of D appear.

1. It is elementary to verify that $Y \equiv g \pmod{2^{v(u-k)}}$. The interpretation of h is a little more complex. If $h = -1$ then the first column of D lies in one of columns $0, 1, \ldots, 2^{v(u-k)} - 1$ of A. If $h \geq 0$, then we know that the first column of D lies in one of columns $2^{v(u-k)}, 2^{v(u-k)} + 1, \ldots, 2^{uv-k} - 1$ of A and that, if we let $Y' = Y - 2^{v(u-k)}$, then $Y' \equiv h \pmod{2^{k(v-1)} - 1}$.

2(a). The reason for this step should be clear from the discussion under (1) above.

2(b). Since $Y \equiv g \pmod{2^{v(u-k)}}$ it immediately follows that we also have $Y' \equiv g \pmod{2^{v(u-k)}}$. Hence the algorithm correctly solves for Y' and hence for Y.

3(a). In this case it should be clear that the column of A containing the first column of D will be an unshifted version of the cycle from the Perfect Factor, and hence $X = F_2(\mathbf{d}_0)$.

3(b). The value of λ indicates the number of times a complete copy of the pseudorandom sequence has been added to the shift value applied to the column of A containing the first column of D. The value of h indicates the number of extra elements of the pseudorandom sequence also added to this shift value. Hence

$$X = \lambda S + H_2(h) + F_2(\mathbf{d}_0) \bmod 2^k,$$

where S represents the sum of the elements in the pseudorandom sequence. But, as in the proof of Lemma 3.3,

$$S = 2^{kv-k-1}(2^k - 1)$$

and hence $x = X$. □

Example 4.2 *As an example of this decoding method we reconsider the* 4×16 *array constructed in Example 3.5. Suppose*

$$D = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}.$$

Then we have

$$\mathbf{d}_0 = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}, \quad \mathbf{d}_1 = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}.$$

Now $F_1(\mathbf{d}_0) = F_1(111) = 1$ *and* $F_1(\mathbf{d}_1) = F_1(000) = 0$. *Hence* $g = G(10) = 3$. *Similarly,* $F_2(\mathbf{d}_0) = F_2(111) = 1$ *and* $F_2(\mathbf{d}_1) = F_2(000) = 0$, *and hence* $h = H_1(1-0) = H_1(1) = 0$. *We therefore have*

$$y' \equiv 3 \pmod 4$$

and

$$y' \equiv 0 \pmod 3,$$

and hence $y' = 3$. *This gives* $y = y' + 4 = 7$.

To compute x *observe that* $y' = 3 = 1 \times 3 + 0$, *i.e.* $\lambda = 1$. *In addition observe that* $H_2(h) = H_2(0) = 0$. *Hence*

$$x = \lambda 2^{kv-k-1}(2^k - 1) + H_2(h) + F_2(\mathbf{d}_0) = 1 \times 2 \times 3 + 0 + 1 = 3 \bmod 4.$$

Hence the top left hand corner of D *is to be found in column 7, row 3 of* A.

5 Construction and decoding of 2^k-ary de Bruijn sequences

In this section we present a new construction which builds a 2^k-ary span v sequence from many copies of a binary span v sequence in a well defined way. The technique allows us to specify a decoding algorithm which reduces the decoding of a 2^k-ary sequence to k decodings of the binary sequence. The constructions and algorithms given here can easily be generalised to sequences over alphabets of arbitrary size, but we omit the details. We also briefly consider the decoding problem for binary de Bruijn sequences.

Construction 5.1 *Let b_1, b_2 and v be positive integers. Let $S^1 = (s_i^1)$ be a 2^{b_1}-ary span v de Bruijn sequence and let $S^2 = (s_i^2)$ be a 2^{b_2}-ary span v de Bruijn sequence. Suppose that both sequences begin with v zeros. Let $(S^2)'$ denote the pseudorandom sequence derived from S^2.*

Construct a binary array A with b_1+b_2 rows and $2^{(b_1+b_2)v}$ columns as follows: the first b_1 rows of A consist of the binary representations of $2^{b_2 v}$ copies of S^1, and the last b_2 rows of A consists of the binary representation of a run of $2^{b_1 v}$ zeros followed by $2^{b_1 v}$ copies of $(S^2)'$.

Theorem 5.2 *Given positive integers b_1, b_2 and v, let A be an array obtained from sequences S^1, S^2 using Construction 5.1. Let $S = (s_i)$ be the $2^{b_1+b_2}$-ary sequence where s_i has a binary representation with least significant bit the first entry in the ith column of A, second least significant bit the second entry in the ith column of A, and so on. Then S is a $2^{b_1+b_2}$-ary span v de Bruijn sequence beginning with v zeros.*

Note Because of the ordering of bits specified in the Theorem, the b_1 least significant bits of the binary representations of the terms of (S) are derived solely from the terms of sequence S^1. Likewise the b_2 most significant bits of the binary representations of the terms of (S) are derived solely from the terms of sequence S^1.

Proof Given an arbitrary $2^{b_1+b_2}$-ary v-tuple, we need to find an i such that the v-tuple occurs in positions $i, i+1, \ldots, i+v-1$ of the sequence derived from A. Equivalently, given an arbitrary binary $(b_1+b_2) \times v$ matrix B, we must find an i such that B occurs as the submatrix of A occupying columns $i, i+1, \ldots, i+v-1$.

We write $B = \begin{pmatrix} C \\ D \end{pmatrix}$ where C is an $b_1 \times v$ array and D is an $b_2 \times v$ array. Since S^1 is a de Bruijn sequence, S^1 contains a v-tuple corresponding to C in some positions $x, x+1, \ldots, x+v-1$ with $0 \leq x \leq 2^{b_1 v} - 1$. We consider two cases.

When D is the all-zero matrix, B will occur beginning at position x in A since the first $2^{b_1 v}$ columns of A are formed from one copy of the de Bruijn sequence S^1 and a run of $2^{b_1 v}$ zeros, and since $(S^2)'$ begins with $v-1$ zeros.

Suppose now that D is not identically zero. Then the sequence $(S^2)'$ contains a v-tuple corresponding to D in some positions $y, y+1, \ldots, y+v-1$, with $0 \leq y \leq 2^{b_2 v} - 2$. By the Chinese Remainder Theorem we can solve the system

$$\begin{aligned} i &\equiv x \pmod{2^{b_1 v}} \\ i &\equiv y \pmod{2^{b_2 v} - 1} \end{aligned}$$

uniquely for i with $0 \leq i < 2^{b_1 v}(2^{b_2 v} - 1)$. Finally we deduce from the construction of A that B occurs beginning in column $i + 2^{b_1 v}$ of A. □

Example 5.3 *As an example of the above construction method, consider the case $b_1 = b_2 = 1$ and $v = 2$. Let*

$$S^1 = S^2 = \begin{pmatrix} 0 & 0 & 1 & 1 \end{pmatrix}$$

and then

$$(S^2)' = \begin{pmatrix} 0 & 1 & 1 \end{pmatrix}.$$

Construction 5.1 now gives

$$A = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{pmatrix}$$

and we obtain the following 4-ary span 2 de Bruijn sequence:

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 2 & 3 & 1 & 2 & 2 & 1 & 3 & 2 & 0 & 3 & 3 \end{pmatrix}.$$

It is clear how Construction 5.1 can be used repeatedly to generate a 2^k-ary de Bruijn sequence from binary sequences of the same span. There are many ways to combine intermediate sequences to produce a final sequence; we introduce a family of binary trees to describe the possibilities.

Let k be a positive integer and define a set of binary trees T_k as follows. The highest vertex of each tree T is labelled k, and the two vertices directly below a vertex labelled t have labels that are positive integers summing to t. The lowest vertices of each tree T (called the leaves) all have labels equal to 1. We shall distinguish between the left and right subtrees at a vertex of T. We next define a tree traversing algorithm applicable to the trees in T_k which indicates how Construction 5.1 is used on intermediate sequences to form a final 2^k-ary de Bruijn sequence.

Inputs to the algorithm are: a tree T in T_k and a span v binary de Bruijn sequence S. In traversing the tree, we store a sequence at each vertex and the algorithm terminates with a 2^k-ary span v de Bruijn sequence stored at the vertex labelled k.

Algorithm 5.4 *(Sequence Generation)*

store the sequence S at each vertex of T labelled with a 1;
set current vertex to be that with label k;
repeat while (there is no sequence stored at the vertex labelled k)

> *if (both vertices below current vertex contain sequences) then*
>
> > *join the two sequences using Construction 5.1 with sequence S^1 equal to the sequence at the head of the left subtree and sequence S^2 equal to the sequence at the head of the right;*
> > *if (current vertex does not have label k)*
> >
> > > *move up the tree to the next higher vertex;*
>
> *else*
>
> > *go down leftmost subtree whose head vertex does not already contain a sequence;*

Note that the left-right ordering of branches in a particular tree is of importance in determining the final sequence obtained with the algorithm. The set of trees T_k together with the above algorithm then defines the set of possible repeated applications of Construction 5.1 to produce a 2^k-ary sequence from the sequence S.

We do not discuss here the best choice of tree leading to the fastest or most memory efficient production of sequences; rather we next concentrate on how the trees in T_k can be used to define a decoding algorithm for 2^k ary sequences. Below we give a recursive algorithm having inputs a tree T in T_k, a span v binary de Bruijn sequence S and a 2^k-ary v-tuple which we represent as a $k \times v$ binary matrix, Z, having rows $\mathbf{z_1}, \ldots, \mathbf{z_k}$. Our algorithm computes the position of Z in the sequence obtained using tree T and sequence S in Algorithm 5.4. In traversing the tree T, we store intermediate positions at the vertices — the algorithm terminates on returning the position x of Z. A secondary algorithm is invoked to decode the binary v-tuples $\mathbf{z_i}$ in S — thus decoding the 2^k-ary sequence is reduced to k decodings of a binary sequence. We briefly discuss below some possible secondary algorithms.

Algorithm 5.5 *(Sequence Decoding)*

set current vertex to be that with label k;
set $i = 1$;
x =decode(current vertex);
procedure decode(current vertex)

if (current vertex label is 1) then

let x be the decoding of the binary v-tuple $\mathbf{z_i}$ in S obtained using a secondary algorithm;
increment i;
return (x);

else

let b_l, b_r denote the labels of left and right vertices below current vertex respectively;
let x_l = decode(left vertex);
let x_r = decode(right vertex);
if $(x_r = 0)$ then

return(x_l);

else

using the Euclidean Algorithm solve the system

$$x \equiv x_l \pmod{2^{b_l v}}$$

$$x \equiv x_r - 1 \pmod{2^{b_r v} - 1}$$

with $0 \leq x < 2^{b_l v}(2^{b_r v} - 1)$;
return($x + 2^{b_l v}$);

The correctness of this algorithm follows immediately from the proof of Theorem 5.2.

We now discuss the efficiency of Algorithm 5.5. In traversing a tree T in T_k, the algorithm makes at most $k-1$ uses of the Euclidean algorithm, which can be implemented very efficiently. The main overhead is therefore in making k uses of the secondary algorithm to decode sequence S, a span v binary de Bruijn sequence. The simplest (and fastest) technique for decoding such a sequence is to store a complete look-up table, requiring the storage of $v2^v$ bits. Recent work in [16] has shown that a restricted class of de Bruijn sequences can be decoded using 2^v bits of storage and time linear in v. However, for speed and simplicity, our calculations in subsequent sections will be based on the storage for a complete look-up table. In memory critical applications, the work of [16], or a Feedback shift register implementation analogous to the work of Petriu et al. ([19, 22]), can be used to obtain further reductions in the quoted storage requirements.

6 A combined decoding technique and its efficiency

In this section we combine Algorithm 4.1 and the results and algorithms of Section 5 to produce a decoding algorithm for Perfect Maps which is time efficient and has markedly better storage requirements than the brute force decoding method.

We start by considering efficient implementations for each of the functions F_1, F_2, G, H_1 and H_2 of Algorithm 4.1. In order to use the results of Section 5, we restrict the de Bruijn and pseudorandom sequences used in Construction 3.2 to be of the type produced in Section 5. We therefore consider a smaller class of Perfect Maps than that of Construction 3.2.

F_1 This function requires a look-up table having 2^u entries, each of which contains a $(u-k)$-bit number; hence the function requires $2^u(u-k)$ bits of space.

F_2 Similarly this function requires $2^u k$ bits of space.

G This function is a decoder for a 2^{u-k}-ary span v de Bruijn sequence. Using the techniques of section 5, an efficient decoder G can be obtained using around $v2^v$ bits of storage (there is some extra storage associated with the tree used to define the 2^{u-k}-ary sequence).

H_1 As for G, this function can be implemented using about $(v-1)2^{v-1}$ bits of storage.

H_2 Recall that $H_2(j)$ is defined as

$$H_2(j) = \begin{cases} 0 & \text{if } j = 0 \\ \sum_{i=0}^{j-1} \alpha_i \bmod 2^k & \text{if } 1 \le j \le 2^{k(v-1)} - 2 \end{cases}$$

where (α_i) is a 2^k-ary span $v-1$ pseudorandom sequence beginning with $v-2$ zeros. Suppose that (α_i) is derived from a span $v-1$ de Bruijn sequence (β_i) commencing with $v-1$ zeros. Then in fact

$$H_2(j) = \sum_{i=0}^{j} \beta_i \bmod 2^k \text{ for } 0 \le j \le 2^{k(v-1)} - 2$$

Suppose further that (β_i) is derived from an application of Algorithm 5.4 with tree T and span $v-1$ binary de Bruijn sequence $S = (s_i)$. Then (β_i) is made up in a well defined way from the repetition of span $v-1$ de Bruijn and pseudorandom sequences whose sums over a period can be calculated by arguments similar to that of Lemma 3.3. We use the structure of (β_i) to reduce the problem of computing $H_2(j)$ to the problem of summing along intermediate sequences. The reduction is made clear in the following:

Lemma 6.1 *Suppose Construction 5.1 is used to produce a $2^{b_l+b_r}$-ary span $v-1$ de Bruijn sequence (β_i) from 2^{b_l}-ary and 2^{b_r}-ary sequences $S^l=(s_i^l)$ and $S^r=(s_i^r)$ respectively. Let $H_2(j)$ be as above and define $H_2^l(j)=\sum_{i=0}^{j} s_i^l \bmod 2^{b_l+b_r}$ and $H_2^r(j)=\sum_{i=0}^{j} s_i^r \bmod 2^{b_l+b_r}$. Then if*

$$p_l = j \bmod 2^{b_l(v-1)},$$

$$q_l = j \operatorname{div} 2^{b_l(v-1)},$$

$$p_r = (j-2^{b_l(v-1)}) \bmod 2^{b_r(v-1)}-1, \quad \textit{and}$$

$$q_r = (j-2^{b_l(v-1)}) \operatorname{div} 2^{b_r(v-1)}-1$$

we have

$$H_2(j)=\begin{cases} H_2^l(j) & \textit{if } 0\le j\le 2^{b_l(v-1)}-1 \\ H_2^l(p_l)+q_l 2^{b_l(v-1)-1}(2^{b_l}-1) & \\ +2^{b_l}(H_2^r(p_r+1)+q_r 2^{b_r(v-1)-1}(2^{b_r}-1)) \bmod 2^{b_l+b_r} & \textit{otherwise} \end{cases}$$

Proof The least significant b_l bits of the binary representation of $H_2(j)$ are the b_l least significant bits of the sum of q_l repetitions of S^l and the first p_l terms of S^l. Using the same argument as in Lemma 3.3, the sum of the terms of S^l is $2^{b_l(v-1)-1}(2^{b_l}-1)$. When $j\ge 2^{b_l(v-1)}$, the most significant b_r bits of $H_2(j)$ arise from the most significant b_r bits of the previous sum added to the sum of q_r copies of S^r and the terms $s_1^r, s_2^r, \ldots, s_{p_r+1}^r$ of S^r. When $0\le j\le 2^{b_l(v-1)}-1$, the most significant b_r bits of $H_2(j)$ arise solely from the most significant b_r bits of the previous sum. □

Since β_i is produced using Algorithm 5.4, we know that the sums along intermediate sequences may also be obtained by combining sums along lower intermediate sequences using Lemma 6.1. We can therefore reduce the calculation of $H_2(j)$ until we are faced with the problem of repeated summing along the binary sequence S. This is achieved by defining a table U by

$$U(j) = \sum_{i=0}^{j} s_i \bmod 2^k \text{ for } 0\le j\le 2^{v-1}-1$$

requiring storage of $k2^{v-1}$ bits. We give a recursive algorithm whose inputs are the tree T, table U and index j and which returns the value of $H_2(j)$. The correctness of the algorithm follows from Lemma 6.1.

Algorithm 6.2 *(calculation of $H_2(j)$)*

$H_2(j) = sum(j, vertex\ with\ label\ k)$;

procedure sum(i, current vertex)

if (current vertex has label 1) then

return (U(i));

else

let b_l, b_r denote the labels of left and right vertices below current vertex respectively;

if ($0 \leq i \leq 2^{b_l(v-1)} - 1$) then

return (sum(i, left vertex));

else

let $p_l = i \ mod\ 2^{b_l(v-1)}$;

let $q_l = i \ div\ 2^{b_r(v-1)}$;

let $p_r = (i - 2^{b_l(v-1)}) \ mod\ 2^{b_r(v-1)} - 1$;

let $q_r = (i - 2^{b_l(v-1)}) \ div\ 2^{b_r(v-1)} - 1$;

let H_2^l =sum(p_l, left vertex);

let H_2^r =sum($p_r + 1$, right vertex);

return ($H_2^l + q_l 2^{b_l(v-1)-1}(2^{b_l} - 1)$

$+2^{b_l}(H_2^r + q_r 2^{b_r(v-1)-1}(2^{b_r} - 1)) \bmod 2^k$);

Combining the storage for the above implementations whilst ignoring any storage associated with trees T, we deduce that the total space, S_A required for Algorithm 4.1 is given in bits by

$$S_A = 2^u(u-k) + 2^u k + v2^v + (v-1)2^{v-1} + k2^{v-1}$$

which simplifies to

$$S_A = u2^u + (k + 3v - 1)2^{v-1} \tag{1}$$

We note that each use of Algorithm 4.1 requires v uses of F_1 and F_2 (simple look-ups) and one use each of G, H_1 and H_2 (requiring only simple modular arithmetic and multiple look-ups during a tree traversal). Hence the time complexity of our decoder will be very small, and we need only consider its storage requirements.

The calculations for the 'brute force' or direct look-up table approach are somewhat simpler. In this case, if we suppose the look-up table is to be used to decode a $(2^k, 2^{uv-k}; u, v)$-PM, the look-up table will contain as entries all the possible $u \times v$ sub-arrays, and each entry will contain the required positional information. Hence the table will contain 2^{uv} entries, each of uv bits. Thus the total space, S_B required for the brute force decoding algorithm is given by

$$S_B = uv2^{uv}. \tag{2}$$

Hence, by comparing Equations 1 and 2, and noting that $k \leq u$, it is clear that Algorithm 4.1 coupled with the above implementations of functions F_1, F_2, G, H_1 and H_2 will be better than the brute force approach when u and v are not too small.

As an example consider the values of S_A and S_B for the case $u = 5$, $v = 5$ and $k = 4$, i.e. for a $(16, 2^{21}; 5, 5)$-PM. Then we have

$$\begin{aligned} S_A &= u2^u + (k + 3v - 1)2^{v-1} \\ &= 5 \times 2^5 + 18 \times 2^4 \\ &= 160 + 288 = 448 \end{aligned}$$

Clearly at this level, the storage associated with trees in our algorithms should be taken into account. In any case

$$\begin{aligned} S_B &= uv2^{uv} \\ &= 5 \times 5 \times 2^{5\times 5} \\ &= 25 \times 33554432 = 838,860,800 \end{aligned}$$

and hence, in this case, Algorithm 4.1 gives a huge saving in space (from 105 Mbytes to 56 bytes).

7 Areas for further study

The method of construction and the associated decoding algorithm described in this paper raise a number of further questions. We consider some of them briefly.

- First and foremost amongst these is whether efficient decoding algorithms for Perfect Maps of parameters different from those described here can be devised. It would certainly be of practical significance to produce decoding algorithms for 'square' Perfect Maps of the parameters which have been constructed by Fan et al., [7], and for the Perfect Maps constructed in [15].
- It appears likely that the construction and decoding algorithm presented in this paper can be generalised in a number of different ways, in particular to cover the c-ary and multi-dimensional cases. In this latter respect note that three-dimensional Perfect Maps have previously been constructed by Iványi, [10].
- Another possible generalisation of the above construction might be to generate 'Sub-Perfect Maps', i.e. arrays where each u by v sub-array occurs at most once (as mentioned in the introduction). One possible approach of this type would involve replacing the Perfect Factor

(C_i) used in Construction 3.2 with a 'Sub-Perfect' Factor, i.e. a set of sequences of equal length in which every v-tuple occurs at most once. It is certainly true that Etzion's construction, [6], does admit generalisations and modifications of this type.

Acknowledgements

The second author would like to acknowledge the support of Hewlett-Packard Ltd. under the SERC CASE studentship scheme.

References

[1] B. Arazi. Position recovery using binary sequences. *Electronics Letters*, **20**:61–62, 1984.

[2] J.A. Bondy and U.S.R. Murty. *Graph theory with applications*. Elsevier, 1976.

[3] J. Burns and C.J. Mitchell. Coding schemes for two-dimensional position sensing. In M. Ganley, editor, *Cryptography and Coding III*. Oxford University Press (to appear).

[4] N.G. de Bruijn. A combinatorial problem. *Proceedings Nederlandse Akademie van Wetenschappen*, **49**:758–764, 1946.

[5] J. Dénes and A.D. Keedwell. A new construction of two-dimensional arrays with the window property. *IEEE Transactions on Information Theory*, **36**:873–876, 1990.

[6] T. Etzion. Constructions for perfect maps and pseudo-random arrays. *IEEE Transactions on Information Theory*, **34**:1308–1316, 1988.

[7] C.T. Fan, S.M. Fan, S.L. Ma, and M.K. Siu. On de Bruijn arrays. *Ars Combinatoria*, **19A**:205–213, 1985.

[8] H. Fredricksen. A survey of full length nonlinear shift register cycle algorithms. *SIAM Review*, **24**:195–221, 1982.

[9] I.J. Good. Normally recurring decimals. *Journal of the London Mathematical Society*, **21**:167–169, 1946.

[10] A.M. Iványi. Construction of three-dimensional perfect matrices. *Ars Combinatoria*, **29C**:33–40, 1990.

[11] S.A. Lloyd and J. Burns. Finding the position of a subarray in a pseudo-random array. In M. Ganley, editor, *Cryptography and Coding III*. Oxford University Press (to appear).

[12] S.L. Ma. A note on binary arrays with a certain window property. *IEEE Transactions on Information Theory*, **IT-30**:774–775, 1984.

[13] F.J. MacWilliams and N.J.A. Sloane. Pseudo-random sequences and arrays. *Proceedings of the IEEE*, **64**:1715–1729, 1976.

[14] T. Nomura, H. Miyakawa, H. Imai, and A. Fukuda. A theory of two-dimensional linear recurring arrays. *IEEE Transactions on Information Theory*, **IT-18**:775–785, 1972.

[15] K.G. Paterson. Perfect maps. *IEEE Transactions on Information Theory*, submitted.

[16] K.G. Paterson and M.J.B. Robshaw. Storage efficient decoding for a class of binary de Bruijn sequences. Preprint (Mathematics Department, Royal Holloway, University of London), December 1992.

[17] E.M. Petriu. Absolute-type pseudorandom shaft encoder with any desired resolution. *Electronics Letters*, **21**:215–216, 1985.

[18] E.M. Petriu. Absolute-type position transducers using a pseudorandom encoding. *IEEE Transactions on Instrumentation and Measurement*, **IM-36**:950–955, 1987.

[19] E.M. Petriu. New pseudorandom/natural code conversion method. *Electronics Letters*, **24**:1358–1359, 1988.

[20] E.M. Petriu. Scanning method for absolute pseudorandom position encoders. *Electronics Letters*, **24**:1236–1237, 1988.

[21] E.M. Petriu and J.S. Basran. On the position measurement of automated guided vehicles using pseudorandom encoding. *IEEE Transactions on Instrumentation and Measurement*, **38**:799–803, 1989.

[22] E.M. Petriu, J.S. Basran, and F.C.A. Groen. Automated guided vehicle position recovery. *IEEE Transactions on Instrumentation and Measurement*, **39**:254–258, 1990.

[23] I.S. Reed and R.M. Stewart. Note on the existence of perfect maps. *IRE Transactions on Information Theory*, **IT-8**:10–12, 1962.

[24] D. Rees. Note on a paper by I.J. Good. *Journal of the London Mathematical Society*, **21**:169–172, 1946.