Navigable Two-Dimensional Coloring Schemes

Sheelagh Lloyd, John Burns
Personal Systems Laboratory
HP Laboratories Bristol
HPL-94-56
June, 1994

binary sequences,
windowing sequences

The problem addressed in this paper is that of enabling position determination on a two-dimensional surface. The surface will be divided into cells of different colors, and a pen will be able to read the color of one cell at a time as it moves across the surface. Navigation is achieved by designing patterns which allow the position on the surface to be determined, given a small part of the pattern. This paper describes the theory of such coloring schemes and their decoding.

1. Introduction

The project which inspired this work aimed to provide passive position determination by writing some sort of pattern on a surface which can be sensed by a pen. We shall describe here the theory of designing the patterns which enable the determination of the position on the surface, given a small part of the pattern. The surface will be divided into cells of different colours, and the pen will be able to read the colour of one cell at a time as it moves across the surface. Many different geometries are possible, but we have restricted our attention to square grids, sometimes divided with what we call separators. The square cells are the data cells, and carry the pattern information, whereas separators simply serve to mark the transitions between data cells. There are two parts to the coding problem: to define patterns which have the property that it is possible to determine the position of a small part of the pattern, and to code these patterns onto a square grid.

In this paper, we describe the work on these codings to date. We first explain the basic properties that a coding must have for it to be useful, and then look at the consequences of allowing diagonal moves. The next section contains a catalogue of various codings, including a table showing the tradeoffs between the different properties. We then discuss the decoding problem, and then show how to find binary sequences with the required qualities.

2. Basic properties of codings

As mentioned in the introduction, our surface is divided into data cells and, depending on the coding, may have separators. The codings are written in the data cells, and the separators tell us that we have moved between data cells, and tells us the direction of that movement. We have considered schemes with various types of separators, and the three types are illustrated in Figures 1-3 respectively.

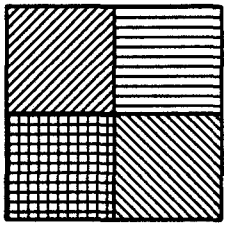

Figure 1

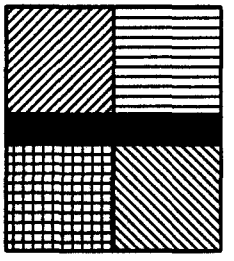

Figure 2

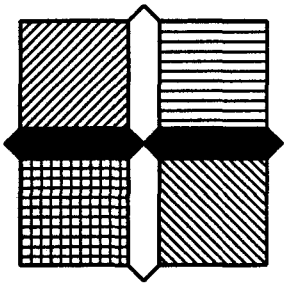

Figure 3

Obviously, the different separator schemes place different requirements on the coding schemes. For example, if there are no separators, then not only must adjacent cells be coloured with different colours, we must also provide within the coding a means for distinguishing horizontal movement from vertical movement. To some extent, we can simulate separators in the coding schemes themselves. For example, suppose we had a one separator (see Figure 2) coding scheme which used a total of *N* colours, so that there are *N-1* data colours and one separator colour. We can turn this into a no separator scheme with *2N-2* colours in the following way. The first, third and every odd numbered row is coloured as in the original scheme, so colours *0* to *N-2* are used. The even rows are coloured with colours *N-1* to *2N-3* by simply colouring each cell with its original value plus *N-1*. Then a vertical movement can be detected by the transition between a colour in the range *0* to *N-2* and a colour in the range *N-1* to *2N-3*, or vice versa. We have thus increased the resolution of the coding (packed in more data cells) at the expense of increasing the number of colours.

Although we began by considering the colours themselves encoding the underlying sequences, we soon found it more convenient to let the transitions between the cells, that is the changes in colour between neighbouring data cells, encode the information. This is not really a restriction, since we were able to cast our old schemes into the new framework, but it does require a certain change of view. In all our codings, therefore, the information which allows us to determine our position can be thought of as being encapsulated in the changes of colour between successive data cells.

Suppose that the transitions in a horizontal direction are given by the functions h_0 and h_1, and in a vertical direction by v_0 and v_1. This means that if the current cell is coloured with colour x, then the next cell in a horizontal direction is coloured with colour $h_0(x)$ or $h_1(x)$ according to the value of the corresponding sequence element. We shall now examine the conditions which will be needed on these functions for the coding to have the properties we desire. We recall that these properties are

- the coding must be well-defined
- decoding must be possible
- if there are no separators, then adjacent cells must be different colours

These conditions alone will lead us to some conclusions about the minimum number of colours necessary for certain types of codings.

Clearly, for the coding to be well-defined, we must have $h_i(v_j(x) = v_j(h_i(x))$ for all colours x and $i, j \in \{0,1\}$ for otherwise we would have a conflict when we tried to determine the colour of the cell diagonally adjacent to a cell coloured x. For the decoding to be possible, we must be able to distinguish between $h_0(x)$ and $h_1(x)$ and between $v_0(x)$ and $v_1(x)$. In other words, for all colours x, we must have $h_0(x) \neq h_1(x)$ and $v_0(x) \neq v_1(x)$.

If there are no separators, there must be no adjacent cells coloured with the same colour, and we must be able to distinguish horizontal movement from vertical movement, so we must have x, $h_0(x)$, $h_1(x)$, $v_0(x)$, $v_1(x)$ are all different for each colour x. If there are just horizontal separators, then we must have x, $h_0(x)$, $h_1(x)$ are all different for each colour x.

We are now able to establish the following statements about minimum numbers of colours.

Proposition 2.1 If there are no separators, then at least 5 colours are needed.

Proof

For any colour x, the colours x, $h_0(x)$, $h_1(x)$, $v_0(x)$, $v_1(x)$ are all different, and so there must be at least five colours.

Proposition 2.2 If there is just one set of separators, then at least 4 colours are needed.

Proof

Suppose for definiteness that the separators are horizontal. For any colour x, the colours x, $h_0(x)$, and $h_1(x)$ are all different, and so there must be at least three data colours. In addition, there must be one separator colour, which makes a total of four.

Proposition 2.3 If there are two sets of separators, then at least 4 colours are needed.

Proof

For any colour x, the colours $h_0(x)$ and $h_1(x)$ are different, and so there must be at least two data colours. In addition, there must be two separator colours, which gives a total of four colours.

We can in fact achieve these minimum numbers of colours, and we shall give examples of such codings in Sections 4.1.1, 4.1.4 and 4.2.2.

3. Allowing diagonal moves

We turn now to the situation where diagonal transitions are allowed. In this case, when such a move is made, there are two bits of information to be determined. We shall call these the x-bit and the y-bit respectively. There are several levels of information possible: we might just be able to tell that we have made a diagonal move or we might also know

- that the x-bit and y-bit are equal (or unequal) or
- the values of both the x-bit and y-bit.

In fact, there is another possibility: that we get either get the information that the x-bit and y-bit are equal or else we get the value of both bits. We shall now express these conditions in terms of the functions h_i and v_j.

The colours in the following set may occur a diagonal move away from x

$$\{h_0(v_0(x)), h_0(v_1(x)), h_1(v_0(x)), h_1(v_1(x))$$

Note that this set has at least two elements, because $h_0(v_j(x)) \neq h_1(v_j(x))$ and $h_i(v_0(x)) \neq h_i(v_1(x))$. The size of this set determines the amount of information we can expect to obtain once we know that a diagonal move has taken place. We shall say that a coding is r-diagonal (where $r = 2,3,4$) if the size of this set is equal to r.

The problem of detecting diagonal moves is different depending on the number of separators. If there are two sets of separators, then diagonal moves are automatically detected by the separators. At the other extreme, if there are no separators, then the diagonal values must not be the same as any horizontal or vertical ones: in other words we must have

$$\{x, h_0(x), h_1(x), v_0(x), v_1(x)\} \cap \{h_0(v_0(x)), h_0(v_1(x)), h_1(v_0(x)), h_1(v_1(x))\} = \varnothing$$

for each colour x. If we have one set of separators, say horizontal ones, then crossing a separator tells us that we have either made a vertical or a diagonal move, and so in this case we need

$$\{v_0(x), v_1(x)\} \cap \{h_0(v_0(x)), h_0(v_1(x)), h_1(v_0(x)), h_1(v_1(x))\} = \varnothing$$

for each colour x.

We are now able to establish some statements on the minimum numbers of colours.

Proposition 3.1 If there are no separators, then the following numbers of colours are required:

- 7 for a 2-diagonal coding
- 8 for a 3-diagonal coding
- 9 for a 4-diagonal coding.

Proof

For any colour x, we must have

$$\{x, h_0(x), h_1(x), v_0(x), v_1(x)\} \cap \{h_0(v_0(x)), h_0(v_1(x)), h_1(v_0(x)), h_1(v_1(x))\} = \varnothing$$

The first set has five elements, and the second set has r elements for an r-diagonal coding. We must therefore have at least $r+5$ colours for an r-diagonal coding.

Proposition 3.2 If there are just one set of separators, then the following numbers of colours are required:

- 5 for a 2-diagonal coding
- 6 for a 3-diagonal coding
- 7 for a 4-diagonal coding.

Proof

Suppose for definiteness that the separators are horizontal. For any colour x, we must have

$$\{v_0(x), v_1(x)\} \cap \{h_0(v_0(x)), h_0(v_1(x)), h_1(v_0(x)), h_1(v_1(x))\} = \varnothing$$

The size of the first set is 2, and the size of the second set is r. So we need at least $r+2$ data colours, and hence $r+3$ colours altogether for an r-diagonal coding.

Proposition 3.3 If there are two sets of separators, then the following numbers of colours are required:

- 4 for a 2-diagonal coding
- 5 for a 3-diagonal coding
- 6 for a 4-diagonal coding.

Proof

In this case, we need at least r data colours for an r-diagonal coding. In addition, there must be two separator colours, which gives a total of $r+2$ colours.

We can in fact achieve most of these minimum numbers of colours, and we shall give examples of such codings in Sections 4.1.2, 4.1.3, 4.1.9, 4.1.5, 4.1.10, 4.2.2, 4.1.6, and 4.2.3. There is one case where we cannot achieve the minimum given above.

Proposition 3.4 There is no 7 colour 2-diagonal no separator coding.

Proof

A 2-diagonal coding has the property that the set $\{h_0(v_0(x)), h_0(v_1(x)), h_1(v_0(x)), h_1(v_1(x))$ has exactly two elements. Now we know that h_0 has no fixed points and that $v_0(x) \neq v_1(x)$, so $h_0(v_0(x)) \neq h_0(v_1(x))$. Similarly, $h_0(v_0(x)) \neq h_1(v_0(x))$, so for the diagonal set to have only two elements, we must have $h_0(v_0(x)) = h_1(v_1(x))$ for all colours x. But we also know that $h_1 = h_0^{-1}$ and $v_1 = v_0^{-1}$, so if we denote $h_0 v_0$ by f, then we see that f^2 is the identity function. But all functions from a set of size 7 to itself whose square is the identity have a fixed point, so f must have a fixed point. So there exists x such that $h_0(v_0(x)) = x$, which contradicts the fact that we can detect diagonal moves. Hence there is no such coding.

In all of the above, we have been assuming tacitly that the only possible vertical transitions are $x \mapsto v_0(x), v_1(x)$. This will only be true if $v_0^{-1}, v_1^{-1} \in \{v_0, v_1\}$, and this will only happen if either $v_0 = v_1^{-1}$ or $v_i = v_i^{-1}$ for $i = 0,1$. If this is not the case, then the number of colours will inevitably be increased. On the other hand, some more information about direction will be available because if $v_0^{-1}(x) \notin \{v_0(x), v_1(x)\}$ and we observe the transition $x \mapsto v_0^{-1}(x)$ we will immediately be able to tell in which direction we are travelling.

If $v_0^{-1} \notin \{v_0, v_1\}$, then for at least some x, the diagonal set contains $h_0(v_0^{-1}(x))$ and $h_1(v_0^{-1}(x))$, which are distinct. If there are no vertical separators, then the diagonal set must have no intersection with the set $\{v_0(x), v_1(x), v_0^{-1}(x)\}$ for diagonal movements to be detected. So the minimum number of colours is increased by one. Furthermore, since the size of the diagonal set has potentially increased, there are more elements to be distinguished, and this may well increase the number of colours again. Since we are limited in the number of colours that we can provide, we have not considered this case further when diagonal moves are allowed.

4. Catalogue of codings

In this section, we shall describe various codings and their properties. We first present a table containing all the codings, their properties and the sections in which they can be found. The column labelled diagonal properties contains *0* if there is no diagonal detection, and *r* if the coding is *r*-diagonal.

number of separators	number of colours	diagonal properties	section number
0	5	0	4.1.1
0	8	2	4.2.1
0	8	3	4.1.2
0	9	4	4.1.3
1	4	0	4.1.4
1	4	0	4.1.8
1	5	2	4.1.9
1	6	3	4.1.5
1	7	4	4.1.10
2	4	2	4.2.2
2	5	3	4.1.6
2	6	4	4.2.3
2	7	4	4.1.7

Table 1 : Coding schemes

We shall group the descriptions of the codings according to the type of transition function used.

4.1 Addition modulo N

The first set of codings in this section have

$$h_0(x) = x + a \pmod N, \quad h_1(x) = x - a \pmod N, \quad v_0(x) = x + b \pmod N, \quad v_1(x) = x - b \pmod N$$

Note that we must always have $a, b \neq 0$, for else we cannot decode.

We deal first with the no separator case. Here we must have $a \neq b$, since we must be able to distinguish between horizontal and vertical moves.

4.1.1 No separator, five colour coding

$$h_0(x) = x + 1 \pmod 5, \quad h_1(x) = x - 1 \pmod 5, \quad v_0(x) = x + 2 \pmod N, \quad v_1(x) = x - 2 \pmod 5$$

This coding realises the minimum number of colours as dictated by Proposition 2.1. It has no diagonal detection.

4.1.2 No separator, eight colour coding

$$h_0(x) = x + 1 \pmod 8, \quad h_1(x) = x - 1 \pmod 8, \quad v_0(x) = x + 3 \pmod 8, \quad v_1(x) = x - 3 \pmod 8$$

This coding realises the minimum number of colours for a 3-diagonal coding as dictated by Proposition 3.1.

4.1.3 No separator, nine colour coding

$$h_0(x) = x + 1 \pmod 9, \quad h_1(x) = x - 1 \pmod 9, \quad v_0(x) = x + 3 \pmod 9, \quad v_1(x) = x - 3 \pmod 9$$

This coding realises the minimum number of colours for a 4-diagonal coding as dictated by Proposition 3.1.

We turn now to the case of one set of separators. As usual, we assume for definiteness that the separators are horizontal.

4.1.4 One separator, four colour coding

$$h_0(x) = x + 1 \pmod 3, \quad h_1(x) = x - 1 \pmod 3, \quad v_0(x) = x + 1 \pmod 3, \quad v_1(x) = x - 1 \pmod 3$$

This coding realises the minimum number of colours as dictated by Proposition 2.2. It has no diagonal detection.

4.1.5 One separator, six colour coding

$$h_0(x) = x + 1 \pmod 5, \quad h_1(x) = x - 1 \pmod 5, \quad v_0(x) = x + 1 \pmod 5, \quad v_1(x) = x - 1 \pmod 5$$

This coding realises the minimum number of colours for a 3-diagonal coding as dictated by Proposition 3.2.

We now look at the case of two sets of separators.

4.1.6 Two separators, five colour coding

$$h_0(x) = x + 1 \pmod 3,\quad h_1(x) = x - 1 \pmod 3,\quad v_0(x) = x + 1 \pmod 3,\quad v_1(x) = x - 1 \pmod 3$$

This coding realises the minimum number of colours for a 3-diagonal coding as dictated by Proposition 3.3.

4.1.7 Two separators, seven colour coding

$$h_0(x) = x + 1 \pmod 5,\quad h_1(x) = x - 1 \pmod 5,\quad v_0(x) = x + 2 \pmod 5,\quad v_1(x) = x - 2 \pmod 5$$

This coding is 4-diagonal. The reason that we cannot achieve the minimum number (6) of colours here is that we cannot find a and b mod 4 so that 0, a, $-a$, b and $-b$ are all distinct, so we must use five data colours. We shall have to wait until Section 4.2.3 for such a coding with six colours.

The second set of codings in this section have

$$h_0(x) = x + a \pmod N,\quad h_1(x) = x - a \pmod N,\quad v_0(x) = x \pmod N,\quad v_1(x) = x + b \pmod N$$

and are therefore only suitable for one or two separator codings.

Note that we must always have $a,b \neq 0$, for else we cannot decode.

4.1.8 One separator, four colour coding

$$h_0(x) = x + 1 \pmod 3,\quad h_1(x) = x - 1 \pmod 3,\quad v_0(x) = x \pmod 3,\quad v_1(x) = x + 1 \pmod 3$$

This coding realises the minimum number of colours as dictated by Proposition 2.2.

4.1.9 One separator, five colour coding

$$h_0(x) = x + 1 \pmod 4,\quad h_1(x) = x - 1 \pmod 4,\quad v_0(x) = x \pmod 4,\quad v_1(x) = x + 2 \pmod 4$$

This coding realises the minimum number of colours for a 2-diagonal coding as dictated by Proposition 3.2.

4.1.10 One separator, seven colour coding

$$h_0(x) = x + 1 \pmod 6,\ h_1(x) = x - 1 \pmod 6,\ v_0(x) = x \pmod 6,\ v_1(x) = x + 3 \pmod 6$$

This coding realises the minimum number of colours for a 4-diagonal coding as dictated by Proposition 3.2.

4.2 Bitwise exclusive or

The first set of codings in this section have

$$h_0(x) = x \oplus a,\ h_1(x) = x \oplus b,\ v_0(x) = x \oplus c,\ v_1(x) = x \oplus d$$

Note that we must always have $a,b,c,d \neq 0$ and $a \neq b$ and $c \neq d$, for else we cannot decode.

We deal first with the no separator case. Here we must have a, b, c and d all different, since we must be able to distinguish between horizontal and vertical moves.

4.2.1 No separator, eight colour coding

$$h_0(x) = x \oplus 1,\ h_1(x) = x \oplus 7,\ v_0(x) = x \oplus 4,\ v_1(x) = x \oplus 2$$

This coding is 2-diagonal.

The second set of codings in this section have

$$h_0(x) = x,\ h_1(x) = x \oplus a,\ v_0(x) = x,\ v_1(x) = x \oplus b$$

and so are only suitable for the case with two sets of separators.

4.2.2 Two separators, four colour coding

$$h_0(x) = x,\ h_1(x) = x \oplus 1,\ v_0(x) = x,\ v_1(x) = x \oplus 1$$

This coding realises the minimum number of colours for a 2-diagonal coding as dictated by Proposition 3.3.

4.2.3 Two separators, six colour coding

$$h_0(x) = x, \quad h_1(x) = x \oplus 1, \quad v_0(x) = x, \quad v_1(x) = x \oplus 2$$

This coding realises the minimum number of colours for a 4-diagonal coding as dictated by Proposition 3.3.

5. Decoding

There are two problems in decoding: separating the input into the x and y sequences, and tracking the position in each of those sequences. When diagonal moves are not allowed, or the coding is 4-diagonal there is no problem extracting the x and y sequences, but in other cases, there may be some considerable difficulty. We may, for example, have a 2-diagonal coding and be able to tell on a diagonal move only that the x and y bits are equal or unequal, without knowing their values. If there are very few diagonal moves, then it may be possible to deal with such codings using a variant of our main algorithm, which is described below. If, however, there will be many diagonal moves, then at the moment we can only recommend the use of a 4-diagonal coding, with its consequent increase in number of colours, since we have no good way of dealing with it otherwise.

We shall deal in this section with the problem of position tracking within a one-dimensional sequence, and assume that we have successfully extracted these sequences from the input information.

Suppose that we are concerned with the x dimension, and have a coding specified by the two functions h_0 and h_1. Then the only possible transitions are $x \mapsto h_0(x), h_1(x), h_0^{-1}(x), h_1^{-1}(x)$. We shall assume that the number of possible transitions does not depend on the starting point: that is that the size of the set $\{h_0(x), h_1(x), h_0^{-1}(x), h_1^{-1}(x)\}$ is independent of x. There are now essentially four cases to be considered:

1. all these four are different
2. $h_0 = h_0^{-1}, h_1 \neq h_1^{-1}$
3. $h_0 = h_0^{-1}, h_1 = h_1^{-1}$
4. $h_0 = h_1^{-1}$.

In the first case, we get both direction information and a bit of the sequence, that is, we can say something like *we are travelling from right to left and have read a 0.* In the second case, we either get a 0 and no direction information, or a 1 and some direction information. In the third case, we just get the bit of the sequence. In the fourth case, we can only say something like *either we are travelling from right to left and have read a 0, or else we are travelling from left to right and have read a 1.* For example, the coding in Section 4.1.1 is of the fourth type, the vertical coding of Section 4.1.9 is of the third type, and the vertical coding of Section 4.1.8 is of the second type. We have not described a coding of the first type, probably because it would require a larger number of colours, but an example would be $h_0(x) = x + 1 (\mathrm{mod}\, 5)$, $h_1(x) = x + 2 (\mathrm{mod}\, 5)$.

These different types of codings require different methods of decoding and different kinds of sequences to make position finding possible. We shall ignore for the moment the problem of changing direction, and concentrate simply on finding the position of a subsequence of length m within the sequence, when we do not know the direction of travel beforehand. In the first case, we obtain direction information immediately, and so the sequence needs only to have the windowing property, that is that each sequence of length m occurs at most once within the sequence. In the second case, providing there is at least one 1 in the subsequence, then we obtain direction information, and so we simply need a windowing sequence which does not contain the all zero window. In the third case, we get no direction information, and so the sequence must be orientable, that is each window of length m must occur at most once, and if a sequence occurs, then its reverse does not. In the fourth case, we again get no direction information, and this time the sequence must be

what we call complement-orientable: that is each window of length m must occur at most once, and if a sequence occurs, then its complement reversed does not. In Section 6, we shall discuss methods for finding sequences of these different types.

Suppose now that we have found our position in the sequence, and we now want to track any further movement (which may continue in the same direction or reverse at any point). Again, the tracking will depend crucially on the type of coding. If the coding is of the first type, then any reversal of direction will immediately be detected and so the position will always be known correctly (assuming, as always, that there are no errors in reading the colours). For all other types, there may be some uncertainty as to the direction, and we may not be able to detect a reversal until some time after it has happened. In these cases, therefore, we need to develop some method of dealing with this. The way we do this is by means of the algorithm described below.

We assume that we continue in the same direction and compare each bit we read with the expected bit of the sequence. When these bits do not match, we must have reversed somewhere previously. We look at the sequence to see where we could have reversed and only now discovered it, and set up candidate pointers at the positions we could be at now. We then continue reading bits and comparing them with those we would have expected. This enables us to discard candidate pointers when we read bits which contradict those expected. After a short time, all but one of the pointers will have been discarded and we will have re-established our position and direction.

It is clearly of interest to determine how long this process will take: in particular, how long will it be until a reversal is detected, and how long after that will it be before all but one of the candidate pointers is discarded. We call the first of these times the discovery time and the second the recovery time and we have the following result.

Proposition 5.1 The maximum discovery time for a Type 2, 3 or 4 sequence with window length m is *(m+1)/2*, and the maximum recovery time is *(m-1)*.

In order to prove this result, we first set up some notation. Suppose that the cells are coloured with the sequence of colours $c_0, c_1, \ldots, c_N$, and that the underlying sequence is denoted by $b_0, b_1, \ldots, b_{N-1}$, so that $c_{i+1} = h_{b_i}(c_i)$ for $i = 0, .., N-1$.

Suppose that we know initially that we are travelling in the direction of increasing *i*. There are two possibilities for reversal: it may occur inside a data cell or, if there are separators, then it may occur within a separator. These two cases will give rise to slightly different conditions.

We deal first with the case of reversal within the data cell c_t. The conditions for a reversal to have occurred at c_t and be detected when we thought we were at c_{t+s} are that

$$c_{t-1} = c_{t+1}, \ldots, c_{t-s+1} = c_{t+s-1}, c_{t-s} \neq c_{t+s},.$$

The conditions for a reversal to have occurred between c_t and c_{t+1} and be detected when we thought we were at c_{t+s} are that $c_t = c_{t+1}, \ldots, c_{t-s} = c_{t+s-1}, c_{t-s-1} \neq c_{t+s}$,

We now consider the recovery process. Suppose that we have detected a reversal when we thought we were at c_u. Then we look for pairs *(t,s)* such that $u = t + s$, and t and s satisfy one of the above sets of equations. If there is only one such pair, then the recovery time is zero. Otherwise, for each pair *(u-s,s)*, we look at the sequence $c_{u-2s}, c_{u-2s-1}, \ldots$ if the first set of equations is satisfied and at the sequence $c_{u-2s-1}, c_{u-2s-2}, \ldots$ if the second set is satisfied and compare them with the bits read. The recovery time is equal to the longest T for which two of these sequences match.

We first need a lemma.

Lemma 1 Suppose that $h_a(c) = h_b^{-1}(c)$. Then

- if the coding is Type 2, then $a = b = 0$
- if the coding is Type 3, then $a = b$
- if the coding is Type 4, then $a \neq b$.

The proof is immediate from the definition of the different types and we are now able to prove Proposition 5.1.

Proof of Proposition 5.1

We look first at the case of a Type 2 coding. Consider first reversals which take place within a data cell. We have $c_{t-r} = c_{t+r}$ for $1 \leq r \leq s-1$. In particular, we have $h_{b_{t-1}}^{-1}(c_t) = c_{t-1} = c_{t+1} = h_{b_t}(c_t)$, which implies that $b_{t-1} = b_t = 0$ by the lemma. In general, we have $h_{b_{t-r}}^{-1}(c_{t-r+1}) = c_{t-r} = c_{t+r} = h_{b_{t+r-1}}(c_{t+r-1})$, and, since $c_{t-r+1} = c_{t+r-1}$, we deduce that $b_{t-r} = b_{t+r-1} = 0$ for $1 \leq r \leq s-1$. So the sequence $b_{t-s+1},\ldots,b_{t+s-2}$ is a string of *2s-2* zeros. But the longest string of zeros allowed in a sequence for a Type 2 coding is equal to *m-1*, so we must have $s \leq (m+1)/2$, as required. When we go through the same process for reversals within a separator, we obtain a string of *2s-1* zeros, and hence no stronger restriction on s.

We may do the same for Types 3 and 4 codings, and obtain respectively a palindrome of length *2s-2* and a *complement-palindrome* (a sequence which is equal to its complemented reverse) of length *2s-2*, and thus obtain the same bound on s.

We look now at the recovery times. Here the three types are not distinguished as we shall only be using the windowing property, which they all share. Suppose that the recovery time is T. We shall assume that both the possible reversals take place in data squares, since this will give us the stronger bound on T and be applicable to both separator and non-separator cases. Then there exist s and r with $0 < r < s$ such that

$$c_{u-s-1} = c_{u-s+1}, \cdots, c_{u-2s-1} = c_{u-1}, c_{u-2s} \neq c_u$$
$$c_{u-r-1} = c_{u-r+1}, \cdots, c_{u-2r-1} = c_{u-1}, c_{u-2r} \neq c_u$$
$$c_{u-2s} = c_{u-2r}, c_{u-2s-1} = c_{u-2r-1}, \cdots, c_{u-2s-T+1} = c_{u-2r-T+1}$$

Consider the sequences $c_{u-r}, \cdots, c_{u-2r+1}, c_{u-2r}, \cdots, c_{u-2r-T+1}$ and $c_{u+r-2s}, \cdots, c_{u-2s+1}, c_{u-2s}, \cdots, c_{u-2s-T+1}$. We claim that they are equal. Clearly the last T elements are equal from the third equation above. We need only show then that $c_{u-r-i} = c_{u+r=2s-i}$ for $0 \leq i \leq r-1$. But for that range of values of r, we have

$$c_{u-r-i} = c_{u-r+i} = c_{u-s+(s-r+i)} = c_{u-s-(s-r+i)} = c_{u+r-2s-i}$$

So the sequences are equal. We may deduce from this that the sequences $b_{u-r}, \cdots, b_{u-2r+1}, b_{u-2r}, \cdots, b_{u-2r-T+2}$ and $b_{u+r-2s}, \cdots, b_{u-2s+1}, b_{u-2s}, \cdots, b_{u-2s-T+2}$ are equal. This gives us a repeated subsequence of length $r + T - 1$. By the windowing property, we deduce that $r+T-1 \leq m-1$, and hence that $T \leq m-1$.

6. Finding sequences

In this section, we shall present some methods of finding sequences with the properties we want. We recall that there are four types of codings, requiring four different types of sequences:

- Type 1: the sequence must be windowing
- Type 2: the sequence must be windowing and not contain the all zero window
- Type 3: the sequence must be orientable
- Type 4: the sequence must be complement-orientable.

The first two types of sequences have been well-studied in the literature: the first type are called deBruijn sequences and the second pseudo-random sequences. There are many results concerning the construction of such sequences, and we do not pursue the subject further here. The second two types do not seem to have received any attention before. It is relatively straightforward to perform computer searches to find such sequences, but it is prohibitively time-consuming for even moderate window lengths. It would be useful, therefore, as well as intellectually satisfying, to be able to construct these sequences algorithmically.

We shall express the problem of finding sequences in terms of finding a path through a directed graph; this is standard practice in the area of deBruijn sequences. If we are looking for sequences with window length m, we construct a directed graph, usually called the deBruijn graph, whose vertices are labelled with the sequences of length *(m-1)* and whose edges are labelled with the sequences of length m, as follows. An edge is drawn from a vertex labelled with $(a_0,a_1,\ldots,a_{m-2})$ to a different vertex labelled with $(b_0,b_1,\ldots,b_{m-2})$ if and only if $a_1 = b_0, a_2 = b_1, \ldots, a_{m-2} = b_{m-3}$. If this condition is satisfied, then the edge is labelled with $(a_0,a_1,\ldots,a_{m-2},b_{m-2}) = (a_0,b_0,\ldots,b_{m-3},b_{m-2})$.

We see, therefore, that a path through the graph can be thought of as a sequence where successive windows of the sequence correspond to the successive edge labels. All sequences of length m except the all zero and the all one sequence occur as edge labels. These do not occur because they would link a vertex to itself, which is not allowed.

6.1 Orientable sequences

We can obtain a crude upper bound on the length of an orientable sequence of window length m. The total number of windows of length m is 2^m. If m is even, then the number of palindromes of length m is equal to $2^{m/2}$, whereas if m is odd, then the number of palindromes is $2^{(m+1)/2}$. The remainder of the windows may be grouped into pairs, each one with its reverse. The maximum length of an orientable sequence is then

$$M = \begin{cases} 2^{m-1} - 2^{(m-2)/2} + m - 1 & \text{if } m \text{ is even} \\ 2^{m-1} - 2^{(m-1)/2} + m - 1 & \text{if } m \text{ is odd} \end{cases}$$

It seems, however, that this upper bound is never attained.

An orientable sequence can be thought of as a path through the deBruijn graph which has the property that if an edge occurs in the path, then the edge with the reverse label does not occur. In particular, no edge with a palindromic label may occur. There are well known methods for finding Eulerian paths (that is paths which use each edge exactly once) in a directed graph, but none of them translates over immediately to our problem. We have several algorithms for finding paths, but they are not guaranteed to find paths of maximal lengths. We shall describe these briefly.

A directed graph has an Eulerian path if it is connected, and each vertex (except possibly the start and end vertices) has the same number of edges coming into it as going out from it. This is indeed the case for the deBruijn graph. We know that we are not allowed to use edges with palindromic labels, so we must remove them. In order to preserve the Euler property, we must actually remove complete cycles containing palindromes. This is the first

step, therefore, to construct and remove cycles containing palindromes, and we have a number of ways of doing this. The second stage is to find a path through the remaining graph. A standard method is first to construct a spanning tree, mark all edges in this tree, and then trace a path backwards through the graph, always choosing an unmarked edge if possible. We have adapted this in a number of ways to our problem.

We can also construct orientable sequences of window length $2m+1$ recursively from orientable sequences of window length $2m$, using the Lempel homomorphism. This maps binary sequences of length t to binary sequences of length $t-1$ and is defined by $(x_0,\ldots,x_{t-1}) \mapsto (x_0 \oplus x_1,\ldots,x_{t-2} \oplus x_{t-1})$. The inverse image of $(y_0,\ldots,y_{t-2})$ is the set $\{(u_0,\ldots,u_{t-1}),(v_0,\ldots,v_{t-1})\}$ where $u_0 = 0$, $v_0 = 1$ and

$$u_{i+1} = \begin{cases} u_i & \text{if } y_i = 0 \\ \overline{u_i} & \text{if } y_i = 1 \end{cases}$$

and

$$v_{i+1} = \begin{cases} v_i & \text{if } y_i = 0 \\ \overline{v_i} & \text{if } y_i = 1 \end{cases}$$

If we have an orientable sequence of length m and window length $2m$ which starts with $2m-1$ zeros and ends with $2m-1$ ones, and look at its inverse image under the Lempel homomorphism, we obtain two sequences, one starting with $2m$ zeros and finishing with a string of $2m$ alternating zeros and ones, and the other equal to the complement of this. If we reverse the second sequence, and glue the two sequences together, amalgamating the string of alternating zeros and ones, to obtain a sequence of length $2M + 2 - 2m$ which is orientable with window length $2m+1$.

We present below a table of the longest orientable sequences that we have found so far for window lengths 4 to 16. Those for window length up to 7 have been verified to be maximal by exhaustive computer search.

window size	length of sequence
4	8
5	14
6	26
7	48
8	108
9	210
10	440
11	872
12	1860
13	3710
14	7400
15	15467
16	31766

Table 2 : Orientable sequence lengths

6.2 Complement-orientable sequences

In this case, we are able to construct maximal length sequences when the window length is odd, but are not able to do any better than in the orientable case if the window length is even.

6.2.1 Odd window length

If the window length m is odd, then a sequence of length m and the reverse of its complement cannot be equal. So the set of sequences of length m may be arranged in pairs by associating each sequence with the reverse of its complement, and any complement-orientable sequence may contain at most one of each pair as a subsequence. The number of these pairs is clearly 2^{m-1}. A sequence of length n contains $n-m+1$ subsequences of

length m, so if n is the length of a complement-orientable sequence with window length m, then we may deduce that $n-m+1 \leq 2^{m-1}$ and so $n \leq 2^{m-1}+m-1$.

If we could construct a windowing sequence whose subsequences each contain more zeros than ones, then it would automatically be complement-orientable, since the reverse of the complement of any subsequence would have more ones than zeros, and so not be a subsequence itself. We shall do this by looking at a subgraph of the deBruijn graph. This subgraph consists of all vertices and edges whose labels have at most *(m-1)/2* ones. Then any Eulerian path through this directed graph yields a maximal length complement-orientable sequence.

Let the indegree of a vertex be the number of edges coming into that vertex, and similarly let the outdegree of a vertex be the number of edges coming out of that vertex. Then it is well-known that a strongly connected graph has an Eulerian path if and only if the indegree of each vertex is equal to the outdegree of that vertex. A graph with an Eulerian path is called Eulerian. Let G_0 denote the subgraph of the deBruijn graph consisting of those edges and vertices with at most *(m-1)/2* ones in their label. We shall show that G_0 is Eulerian.

Proposition 6.1 If $\underline{a}$ is a vertex of G_0 with exactly *(m-1)/2* ones or $\underline{a} = (0,\ldots,0)$, then $\text{indegree}(\underline{a}) = \text{outdegree}(\underline{a}) = 1$. If $\underline{a}$ is any other vertex of G_0 then $\text{indegree}(\underline{a}) = \text{outdegree}(\underline{a}) = 2$.

Proof

If $\underline{a}$ has *(m-1)/2* ones and there is an edge from $\underline{a}$ to $\underline{b}$, then $b(m-2)=0$ since $(a(0),\ldots,a(m-2),b(m-2))$ has only *(m-1)/2* ones. Hence $b=(a(1),\ldots,a(m-2),0))$ and so there is only one edge coming out of $\underline{a}$. Similarly, there is only one edge going into $\underline{a}$. If $\underline{a}=(0,\ldots,0)$, then there is an edge from $\underline{a}$ to $(0,\ldots,0,1)$ and one from $(1,0,\ldots,0)$ to $\underline{a}$, and no others, since we do not include the edge joining $\underline{a}$ to itself.

If $\underline{a}$ has fewer than *(m-1)/2* ones, then there are edges from $\underline{a}$ to $(a(1),\ldots,a(m-2),0)$ and $(a(1),\ldots,a(m-2),1)$, and similarly edges from $(0,a(0),a(1),\ldots,a(m-2))$ and $(1,a(0),a(1),\ldots,a(m-2))$.

So we have proved the following:

Proposition 6.2 G_0 is Eulerian.

This means that we have shown the existence of a binary sequence whose subsequences of length m are precisely the sequences of length m which have at most *(m-1)/2* ones. We have therefore shown the existence of a complement-orientable sequence of maximal length.

It is straightforward to find an Eulerian path in a graph, and several algorithms exist. We present below an algorithm which finds such a path in this particular graph.

We claim that the following algorithm constructs a maximal length complement-orientable sequence. The successive windows of the sequence are given by $\underline{w}$.

- Set $\underline{w} = (w(0), w(1),\ldots, w(m-1))$ to be $(0,\ldots,0)$.
- If $(w(1),\ldots, w(m-1))$ contains fewer than *(m-1)/2* ones then
  - if we haven't already seen $(w(1),\ldots, w(m-1),1)$ then
    set $\underline{w} = (w(1),\ldots, w(m-1),1)$
  - else if we haven't already seen $(w(1),\ldots, w(m-1),0)$ then
    set $\underline{w} = (w(1),\ldots, w(m-1),0)$
  - else
    stop
- else
  - if we haven't already seen $(w(1),\ldots, w(m-1),0)$ then
    set $\underline{w} = (w(1),\ldots, w(m-1),0)$
  - else
    stop.

In other words, we add 1 to the sequence whenever we can and add 0 otherwise.

Proposition 6.3 The algorithm above yields a maximal length complement-orientable sequence.

Proof

By construction, the sequence generated is a window sequence, and each window has at most *(m-1)/2* ones. So by the remarks earlier, it is a complement-orientable sequence. We need only establish that it is of maximal length. This is equivalent to showing that it corresponds to an Eulerian path in the directed graph described earlier.

Suppose that an edge $\underline{e} = (e(0),\ldots,e(m-1))$ has not been included in the path constructed. We may assume that $e(m-1) = 0$, because if $(e(0),\ldots,e(m-2),1)$ has not been used, then certainly $(e(0),\ldots,e(m-2),0)$ has not been used, since it would have been used first. Now consider the vertex $\underline{a} = (e(1),\ldots,e(m-2),0)$. The edge $\underline{e}$ is an in-edge to this vertex. If $\underline{a}$ has indegree 1, then it has not been visited in the path, and so the edge $(e(1),\ldots,e(m-2),0,0)$ has not been used. If, on the other hand, $\underline{a}$ has indegree 2, then it has been visited at most once in the path. But in this case, it also has outdegree 2, so again the edge $(e(1),\ldots,e(m-2),0,0)$ has not been used. Repeating this argument shows us that all the edges $(e(0),\ldots,e(m-2),0)$, $(e(1),\ldots,e(m-2),0,0)$, $(e(2),\ldots,e(m-2),0,0,0),\ldots,(0,\ldots,0)$ are unused. But we know that $(0,\ldots,0)$ is in the path, so this is a contradiction. Hence all the edges have been used in the path.

6.2.2 Even window length

If the window length is even, then we could simply look at the subgraph of the deBruijn graph where each label has at most *(m-2)/2* ones, which would guarantee complement-orientableness. This, however, does not yield long sequences, because of all the sequences with *m/2* ones which are not complement-palindromes, but have been excluded from consideration by this method. We could use an adaptation of any of the methods used for

finding orientable sequences, but since we do not care whether the window length is odd or even, and we can easily construct maximal sequences when the window length is odd, we have not pursued this further.