ELSEVIER

Discrete Applied Mathematics 101 (2000) 231–245

DISCRETE APPLIED MATHEMATICS

A scheme for the synchronization of variable length codes

S. Perkins, D.H. Smith *

Department of Mathematics and Computing, University of Glamorgan, Pontypridd, Mid Glamorgan CF37 1DL, UK

Received 17 January 1997; revised 9 June 1998; accepted 10 May 1999

Abstract

A synchronization scheme is necessary when variable length codes are used in the presence of errors. In this paper we present a scheme for synchronizing the data stream without allowing slippage. We achieve this by inserting a number of distinct keywords, each consisting of a synchronizing sequence and an explicit or implicit cyclic count, into the data at intervals. We present decoding algorithms for this scheme and prove their effectiveness given limits on the maximum number of errors per cycle of keywords.

Keywords: Variable length codes; Synchronization; Decoding

1. Introduction

Variable length codes may offer increased efficiency for the transmission or storage of data. They have been extensively studied in the literature [7]. Many data compression codes [5] are variable length codes. This means that there is no fixed relationship between the number of symbols input and the number of symbols output.

A major problem of variable length codes is that if random or burst errors cause a decoding failure, all subsequent data may be incorrectly decoded. The following simple Huffman code example illustrates the potential problem.

Example 1. Suppose that the message $DECABDECAB \ldots DECAB$ (with the string $DECAB$ repeated thereafter) is encoded using the Huffman code with codewords $A=01$, $B=10$, $C=11$, $D=000$ and $E=001$. Suppose that the second bit of the transmitted sequence S is received in error.

$$S = 0000011101100000001110110\ldots,$$

$$R = 0100011101100000001110110\ldots$$

* Corresponding author.
E-mail address: dhsmith@glam.ac.uk (D.H. Smith)

0166-218X/00/$ - see front matter
PII: S0166-218X(99)00196-1

When R is decoded the message obtained is $ADCBCDDCBC\ldots DDCBC$ (with the string $DDCBC$ repeated thereafter). The decoder never regains synchronization, all subsequent data is erroneous.

A method for synchronization is necessary so that once resynchronization occurs, data can again be decoded correctly. We shall refer to a synchronization scheme as *weak* if it gives no indication of the amount of data that has been lost or is erroneous.

Example 2. Suppose that the message $DECABDECAB\ldots DECAB$ is again encoded, this time using the Huffman code with codewords $A = 00$, $B = 10$, $C = 11$, $D = 010$ and $E = 011$. Suppose that the second bit of the transmitted sequence S is received in error.

$$S = 0100111100100100111110010\ldots,$$

$$R = 0000111100100100111110010\ldots$$

When R is decoded the message obtained is $AACCABDECAB\ldots DECAB$. The decoder regains synchronization, the first two symbols DE of the transmitted sequence have changed to the three symbols AAC in the decoded sequence.

In this example synchronization is re-established but the positions of the decoded symbols in the data stream are no longer correct. Thus the third symbol in the encoded sequence is received as the fourth symbol in the decoded sequence, etc. We refer to such a loss of position as *slippage*. If the synchronization scheme maintains the position of the decoded symbols in the data stream, which may be necessary in some applications, the synchronization scheme will be referred to as *strong* [3,8]. For example, if the message and decoded message are:

$$Message = BCABDE\ ABCEDADEC\ldots,$$

$$Decoded\ Message = BCA\ EABECCEDADEC\ldots,$$

then synchronization is lost after symbol three and regained at symbol nine. There are five symbols between these positions in both sequences. Strong synchronization is regained at position nine.

The aim of synchronization schemes is thus not to recover all of the data. It is to ensure that the decoding of subsequent data is correct after the loss of data due to error. Additionally, for strong synchronization, it should be possible to deduce the position of the recovered data in the data stream.

There are a number of ways to achieve weak synchronization; the first of these is to use self-synchronizing codes. These codes contain a sequence, called a synchronizing sequence, which always allows the decoder to recover synchronization [4,6,7]. A special class of these are synchronous codes which have at least one of their codewords as a synchronizing sequence [1,2]. Another way to achieve synchronization of a binary

stream is to add redundancy to the data in the form of a synchronizing sequence distinct from the data.

A scheme for strong synchronization has been implemented and described in [8]. The scheme is largely satisfactory, even with a high random error rate. It may be less satisfactory in the presence of burst errors. In this paper we propose a scheme for strong synchronization whereby a number of distinct keywords, each consisting of a synchronizing sequence and an explicit or implicit cyclic count, are placed cyclically into the data stream at intervals (not necessarily at fixed intervals). Clearly errors may hit any of these keywords and on decoding they may not all be received correctly. Some may have been deleted (become unrecognizable) or even changed into a different keyword. Also some may be inserted, as errors could hit the data stream creating a spurious keyword. Spurious keywords might also occur naturally in the data stream. We propose an algorithm that takes in the received sequence of keywords and outputs the keywords that will be assumed to be correct. Given certain restrictions on the number of errors allowed in a received cycle of keywords, we show that through the use of this algorithm we will never erroneously assume that data belongs to a previous or to a future cycle. This will prevent all future data from being misinterpreted. We also suggest a modified algorithm and prove a similar result for this algorithm. Finally, we discuss further possible improvements.

2. The synchronization scheme

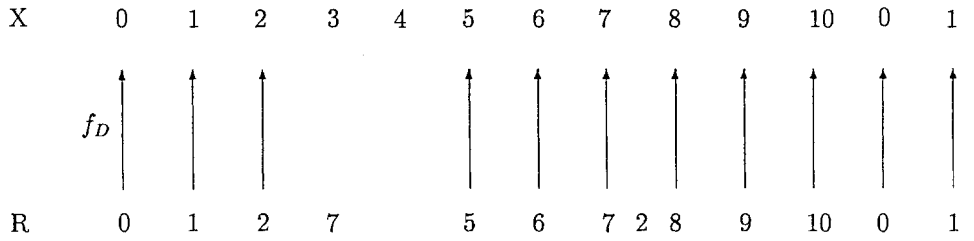
We suppose that there are N synchronizing keywords $c_0, c_1, c_2, \ldots, c_{N-1}$ which are entered cyclically into the data stream, with the variable length encodings of a number of data values inserted between synchronizing keywords.

$$sync_1 \quad data_1 \quad sync_2 \quad data_2 \ldots sync_N \quad data_N \quad sync_1 \quad data_{N+1} \quad sync_2 \quad data_{N+2} \ldots .$$

To ensure correct data recovery on resynchronization after errors have occurred, encoding of data between keywords must be independent of previous data encoding. An adaptive data compression code, for example, would need to be re-initialized after each keyword. Otherwise, an incorrect data block $data_i$ causes the next block $data_{i+1}$ to be incorrectly decoded.

Keywords are written by our algorithm if they are interpreted as correct by the algorithm. We refer to such a correct interpretation as a *match*. Data is then decoded independently by an appropriate method only if it lies between two matched keywords; otherwise data is assumed to be incorrect and may be lost. However, if each block of data $data_i$ corresponds to a fixed number of encoded values, and the decoding of keywords is successful, then a count of the number of missing or erroneous data values is available. Thus strong synchronization can be maintained. For this reason, in the remainder of this paper we focus only on the decoding of the received sequence of synchronizing keywords.

If $X = x_1, x_2, \ldots, x_n$ is the encoded sequence of synchronizing keywords we have $x_i = sync_{i \bmod N} = c_{(i-1) \bmod N}$. The sequence X is the expected received sequence if no errors occur. Also let $R = r_1, r_2, \ldots, r_m$ be the actual received sequence of synchronizing keywords. Keywords of the encoded sequence X may have been changed or lost and spurious keywords may have been inserted. A *decoding* of R is a mapping f_D from the set R' of matched elements of R to the set of elements of X.



Thus on decoding, unless there is a decoding error, when a synchronizing keyword is written its position in the original sequence is known.

Suppose that a decoding algorithm writes r_i and the immediately preceding received keyword written by the algorithm was r_{i-s} where $s \in \{1, 2, \ldots, i-1\}$. If

$$f_D(r_{i-s}) = x_k = c_{(k-1) \bmod N},$$

then

$$f_D(r_i) = x_j,$$

where $j > k$ is the smallest integer such that $x_j = r_i$. We shall call an algorithm with this property *immediate*.

Definition 1. Suppose that r_i is the correctly received synchronizing keyword x_j. We shall say that decoding has *slipped forward* if $f_D(r_i) \in \{x_{j+N}, x_{j+2N}, x_{j+3N}, \ldots\}$ and that decoding has *slipped back* if $f_D(r_i) \in \{x_{j-N}, x_{j-2N}, x_{j-3N}, \ldots\}$.

Three possible types of keyword decoding error can prevent strong synchronization from being correctly recovered. The first type of error is when no further matches are made and synchronization is never re-established. This could only occur in extreme error environments. The other two types of decoding error are slipping forward and slipping back. These can cause decoding to remain permanently out of strong synchronization. If the decoding algorithm is immediate and the matches made are strings of (cyclically) consecutive keywords, then these errors do not occur if at least one correctly received keyword x_i in the match satisfies $f_D(x_i) = x_i$. This follows because a (cyclically) consecutive set of keywords of R' is mapped to a (cyclically) consecutive set of keywords of X.

3. Algorithm A

In the following algorithm we shall identify sets of d or more consecutive synchronizing keywords ($d \geqslant 3$), which we shall refer to as *matches*. Matches will not, however, be written by the algorithm if they have certain overlaps with the previous match. The steps in the algorithm are the following:

1. scan the sequence until d consecutive keywords have been detected,
2. write the matched keywords,
3. extend the match (and write the additional matched keywords) if possible,
4. skip over overlapping keywords.

Algorithm A avoids identifying strings of consecutive keywords with values which overlap in up to $f+1$ positions with the end of the previous identified string. This prevents slipping forward in certain circumstances.

```
INPUT(d); -- GET CHOSEN MATCH LENGTH
INPUT(f); -- GET CHOSEN OVERLAP VALUE
INPUT(N); -- GET CYCLE LENGTH
i:=1; -- NEXT RECEIVED WORD TO BE MATCHED
k:=1; -- LAST MATCHED RECEIVED WORD (INITIALISED TO 1)
WHILE NOT end of sequence LOOP
     consecutive:= TRUE;
     j:=i;
     WHILE consecutive AND j < i + d - 1 LOOP
     --CHECK CONSECUTIVE TO LENGTH d
          IF for some p, r_j = c_p AND r_{j+1} ≠ c_{(p+1) mod N}
          THEN consecutive := FALSE;
          END IF;
          j:=j + 1;
     END LOOP;
     IF consecutive THEN i:=i + d - 1; k:=i;
     --WRITE MATCHED KEYWORDS
          FOR l IN i - d + 1..i LOOP
               WRITE(r_l);
          END LOOP;
     ELSE i:=j;
     END IF;
     WHILE consecutive LOOP --EXTEND MATCH IF POSSIBLE
          IF for some p, r_i = c_p AND r_{i+1} = c_{(p+1) mod N}
          THEN WRITE(r_{i+1}); k:=k + 1;
          END IF;
          IF for some p, r_i = c_p AND r_{i+1} ≠ c_{(p+1) mod N}
          THEN consecutive:= FALSE;
```

```
            END IF;
            i:=i+1;
        END LOOP;
        --NOW SKIP OVER OVERLAPPING KEYWORDS
        WHILE k > 1 AND r_k = c_p for some p AND
            r_i ∈ {c_p, c_((p−1) mod N), c_((p−2) mod N), ..., c_((p−f) mod N)} LOOP
            i:=i+1;
        END LOOP;
END LOOP;
```

4. Vulnerability of Algorithm A to slippage

Suppose that for some s_1

$$S = c_{s_1}, c_{s_1+1 \bmod N}, \ldots, c_{s_1+N-1 \bmod N}$$

is an expected cycle of synchronizing keywords. Let $\sigma(S)$ denote the number of deletion, change or insertion errors affecting S, where insertions are only counted if they occur strictly between c_{s_1} and $c_{s_1+N-1 \bmod N}$. We will also use this notation for a substring of S.

We shall say that a match is *correct* if it contains at least one correctly received keyword x_i such that $f_D(x_i) = x_i$.

Now suppose that decoding of R takes place using Algorithm A as an immediate algorithm. Let $d \geqslant 3$ be the match length used in the algorithm and let $f = d - 2$. We shall see why $f \geqslant d - 2$ is necessary in Section 5.

Theorem 1. *If $\sigma(S) < \min\{\lfloor N/d \rfloor, d\}$ for each cycle S of synchronizing keywords, then at least one correct match is made per cycle and decoding can neither slip back nor slip forward.*

Proof. A match is spurious if it is not correct, i.e. if it contains no correctly received keyword. However, a spurious match requires d consecutive errors, and so cannot occur if $\sigma(S) < d$.

We show first that a correct match is identified in the first cycle $x_1, x_2, \ldots, x_N$ of keywords. Assume that this is not the case. Since the first match cannot be skipped over because of overlap, we must have

$$\sigma(x_{(i-1)d+1}, x_{(i-1)d+2}, \ldots, x_{id}) \geqslant 1 \quad (i = 1, 2, \ldots, \lfloor N/d \rfloor),$$

otherwise one of the sets of d consecutive keywords would be identified as a match. Then at least $\lfloor N/d \rfloor$ errors would have affected the cycle, contradicting $\sigma(S) < \lfloor N/d \rfloor$.

Since the first match occurs in the first cycle it clearly cannot be a slip back. Again, as it is the first match and the algorithm is immediate, it cannot be a slip forward.

Consider the case where the first slippage made by Algorithm A is a slip back. Let the previous correct match end with

$$f_D(r_s) = x_t.$$

Suppose that

$$f_D(r_q) = x_w, f_D(r_{q+1}) = x_{w+1}, \ldots, f_D(r_{q+d'}) = x_{w+d'},$$
$$(d' \geqslant d-1, w > t, q > s)$$

and that we should correctly decode

$$f_D(r_q) = x_{w+\tau N}, f_D(r_{q+1}) = x_{w+\tau N+1}, \ldots, f_D(r_{q+d'}) = x_{w+\tau N+d'}$$

for some fixed $\tau \geqslant 1$.

Consider the sets of keywords

$$\{x_{t+(i-1)d+1}, x_{t+(i-1)d+2}, \ldots, x_{t+id}\} \quad (i = 1, 2, \ldots, \lfloor N/d \rfloor).$$

As $f = d-2$ none of these $\lfloor N/d \rfloor$ sets of consecutive keywords, when correctly received, can be skipped over because of overlap. Also, since $t + id < w + N$, none can be unaffected by error, or a correct match after x_t would occur, contrary to assumption. Thus, we have

$$\sigma(x_{t+(i-1)d+1}, x_{t+(i-1)d+2}, \ldots, x_{t+id}) \geqslant 1 \quad (i = 1, 2, \ldots, \lfloor N/d \rfloor),$$

so $\sigma(S) \geqslant \lfloor N/d \rfloor$. Thus the case where the first slippage made by Algorithm A is a slip back cannot occur.

Now consider the case where the first slippage made by Algorithm A is a slip forward. Again, let the previous correct match end with

$$f_D(r_s) = x_t.$$

Suppose that we should correctly decode

$$f_D(r_q) = x_w, \ f_D(r_{q+1}) = x_{w+1}, \ldots, f_D(r_{q+d'}) = x_{w+d'} \quad (d' \geqslant d-1, w > t, q > s),$$

but in fact,

$$f_D(r_q) = x_{w+\tau N}, f_D(r_{q+1}) = x_{w+\tau N+1}, \ldots, f_D(r_{q+d'}) = x_{w+\tau N+d'}$$

for some fixed $\tau \geqslant 1$. However, this can only occur if there is a spurious match involving d or more keywords r_i with $s < i < q$. We have already seen that such spurious matches are not possible.

Having shown that neither backward nor forward slippage can occur, it remains to show that following a correct match ending with

$$f_D(r_s) = x_t,$$

then a further correct match is identified within one cycle. Assume that this is not the case and consider the sets

$$\{x_{t+(i-1)d+1}, x_{t+(i-1)d+2}, \ldots, x_{t+id}\} \quad (i = 1, 2, \ldots, \lfloor N/d \rfloor).$$

If there is at least one error for each set we again contradict $\sigma(S) < \lfloor N/d \rfloor$. The only remaining case to consider is when

$$\sigma(x_{t+(i-1)d+1}, x_{t+(i-1)d+2}, \ldots, x_{t+id}) \geqslant 1$$

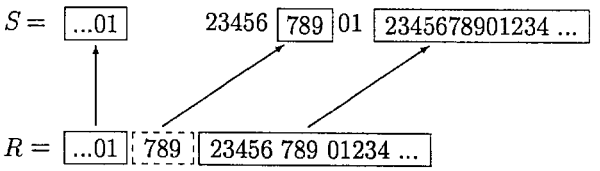
for all but one value of i in $\{1,2,\ldots,\lfloor N/d \rfloor\}$, but for this remaining value of i the match is not identified because of overlap with the match ending at $f_D(r_s) = x_t$. However, if $f = d-2$ this cannot occur. □

5. Examples

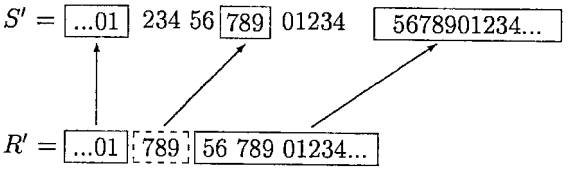
In this section we show by example that the bound in Theorem 1 is tight. In Examples 3 and 4 we take $\sigma(S) \geqslant \min\{\lfloor N/d \rfloor, d\}$ for each cycle S of synchronizing codewords and take $f = d-2$ as stated. In Example 5 we take $\sigma(S) < \min\{\lfloor N/d \rfloor, d\}$ and let $f < d-2$.

Example 3. Consider a sequence S with cycles of length $N = 10$ and decoding match length $d = 3$. Therefore Algorithm A has overlap $f = 1$. Suppose $\sigma(S) = 3$, one greater than Theorem 1 claims to correct. We show that our algorithm cannot deal with this situation.

Suppose we send sequence S and receive sequence R. In sequence R there are three insertion errors, the first 789 listed in R below. On decoding we accept runs of length three or greater. Hence we accept 789 as correct and slip forward a cycle as shown below:



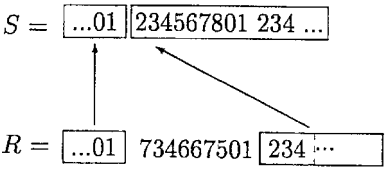
As an alternative example, suppose we send sequence S' and receive sequence R' below. R' contains three change errors (2 becomes 7, 3 becomes 8 and 4 becomes 9). Again the cycle slips forward.



Example 4. Consider a sequence S with cycles of length $N = 9$ and decoding match length $d = 3$. Therefore Algorithm A has overlap $f = 1$. Suppose $\sigma(S) = 3$, one greater

than Theorem 1 claims to correct. We show that our algorithm cannot deal with this situation.

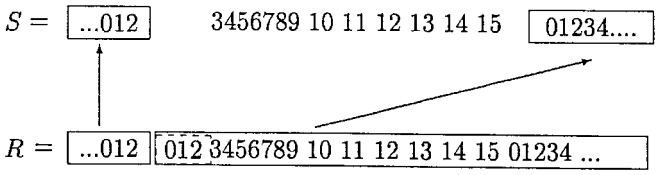
Suppose we send sequence S and receive sequence R below. In sequence R there are three change errors (2 becomes 7, 5 becomes 6 and 8 becomes 5). On decoding we accept runs of length three or greater. The second 01 and the second 1 are ignored due to overlap with the previous match. Hence 234 is taken as correct and we slip back a cycle as shown below:



Next, we show that f cannot be smaller than $d-2$.

Example 5. Consider a sequence S with cycles of length $N=16$, decoding match length $d=4$ and overlap $f=1$ in place of $f=2$ as required by Algorithm A. Suppose we have $\sigma(S)<4$. We show that our algorithm cannot deal with this situation.

Suppose we receive sequence R when sequence S was sent. In sequence R there are three insertion errors, the second 012 listed in R below. On decoding we accept runs of length four or greater. As $f=1$ we accept 012 as correct and slip forward a cycle as shown below:



6. A modified algorithm — Algorithm B

Suppose that $\Sigma = x_i, x_{i+1 \bmod N}, x_{i+2 \bmod N}, \ldots, x_{i+d-1 \bmod N}$ with $d \geqslant 3$ and $\sigma(\Sigma)=1$. If the single error is a change or deletion error affecting the first or last element of Σ, there remains a sequence $x_{i+1 \bmod N}, x_{i+2 \bmod N}, \ldots, x_{i+d-1 \bmod N}$ or $x_i, x_{i+1 \bmod N}, x_{i+2 \bmod N}, \ldots, x_{i+d-2 \bmod N}$ of $d-1$ consecutive keywords. Otherwise, Σ may contain a single insertion, a single change not affecting the ends or a single deletion not affecting the ends. Lines 9–22 of the pseudocode of Algorithm A can be replaced by pseudocode that recognizes any of these received sequences and writes the $d-1$ or d keywords which can be assumed to be correct. Such a match cannot be spurious. We will refer to such a

modified algorithm as Algorithm B. Pseudocode for Algorithm B is contained in the appendix. Let S be defined as in Section 4.

Theorem 2. *Suppose that* $\sigma(S) < \min\{2\lfloor N/d \rfloor - 1, d-1\}$ *for each cycle* S *of synchronizing keywords. Then if Algorithm B is used with* $f = d-2$, *at least one correct match is made per cycle and decoding can neither slip back nor slip forward.*

Proof. The proof is almost identical to the proof of Theorem 1. A spurious match may now only require $d-1$ consecutive errors, but still cannot occur as $\sigma(S) < d-1$. When showing that a correct match is identified in the first cycle, we now have that

$$\sigma(x_{(i-1)d+1}, x_{(i-1)d+2}, \ldots, x_{id}) \geqslant 2 \quad (i = 1, 2, \ldots, \lfloor N/d \rfloor)$$

which still gives a contradiction. When showing that a step back cannot occur none of the $\lfloor N/d \rfloor$ sets of d keywords can be a match. We must have

$$\sigma(x_{t+(i-1)d+1}, x_{t+(i-1)d+2}, \ldots, x_{t+id}) \geqslant 2 \quad (i = 1, 2, \ldots, \lfloor N/d \rfloor - 1).$$

We can only claim

$$\sigma(x_{t+(i-1)d+1}, x_{t+(i-1)d+2}, \ldots, x_{t+id}) \geqslant 1 \quad \text{for } i = \lfloor N/d \rfloor$$

as if

$$\sigma(x_{t+(i-1)d+1}, x_{t+(i-1)d+2}, \ldots, x_{t+id}) = 1,$$

this set of consecutive keywords may be skipped over because of overlap. Again, when showing that following a correct match a further match is identified within one cycle:

$$\sigma(x_{t+(i-1)d+1}, x_{t+(i-1)d+2}, \ldots, x_{t+id}) \geqslant 2$$

except when $i = \lfloor N/d \rfloor$ when we have

$$\sigma(x_{t+(i-1)d+1}, x_{t+(i-1)d+2}, \ldots, x_{t+id}) \geqslant 1. \qquad \square$$

For Algorithm A, d is best chosen so that $\lfloor N/d \rfloor$ is approximately equal to d. For Algorithm B, d is best chosen so that $2\lfloor N/d \rfloor - 1$ is approximately equal to $d-1$. Thus, for moderately large values of N the number or errors that can be tolerated per cycle is increased by a factor of about $\sqrt{2}$.

It would be possible to recognise the effect of more than one error per set Σ of d keywords, and obtain further increases in the number of errors per cycle that could be tolerated. However, the algorithm will become increasingly complex and the benefit will only be seen for very long cycle lengths.

It is possible that an algorithm could take a more global approach and look forward a complete cycle before making a decision about matches. This approach does have the potential to significantly increase the number of errors per cycle that could be tolerated. However, an algorithm which both deals with every possible error pattern and for which results similar to those presented here can be proved remains elusive.

7. Implementation issues

7.1. Data decoding

It has already been noted that the encoding of a block $data_i$ of data must be independent of previous data blocks. The actual decoding rule for the data will depend on the nature and encoding of the data. For this reason, and because it is independent of keyword decoding, it will not be considered here. However, although it was stated that data is only decoded if it lies between two matched keywords, there are in fact three options:

1. Decode data only if it lies between two matched keywords. Accept the data block $data_i$ only if the correct number of data values are obtained. Otherwise discard the data and treat the data block as missing.
2. Decode data only if it lies between two matched keywords. If too few data values are decoded, write them from the beginning of the block, adding an appropriate number of missing values. If too many data values are obtained truncate to the number of items known to be encoded as $data_i$.
3. Data could also be decoded following a correctly matched keyword when the next identified keyword is not consecutive. The data would be truncated when the correct number of values for a block is obtained or when the next keyword is identified.

The first option is safer, in that fewer data values will be decoded incorrectly, but the second and third options may lead to more correct data values being written.

7.2. Parameters of the synchronization scheme

If the number of data values between synchronizing words is too large, a large volume of data may be lost before synchronization is re-established. If it is too small the overhead of the keywords becomes significant. Once this number is fixed, N and d should be chosen with N approximately equal to d^2 (Algorithm A), and with N as large as possible without the length of the keywords becoming an unacceptable overhead.

7.3. Buffer size and delay

The keyword decoding algorithm is essentially a linear search and does not cause unacceptable delay. The main disadvantage of the scheme is the need for sufficient memory to buffer up to $d-1$ blocks of data. These blocks can then be written if the match is established. However, if matches have length greater than d, the first $d-1$ data blocks can be written and there is no need to buffer more than one data block at a time when the match is extended.

7.4. Error control

Depending on the application, it might be necessary to apply error control coding to the data, additional to any already used on the channel. However, using an error

control code for the keywords increases the probability of insertion errors and is not generally helpful.

7.5. Comparison of performance with existing schemes

A comparison of the probability of loss of strong synchronisation of this scheme with that presented in [8,9], and with other possible schemes will appear in a future paper. The schemes presented in [8,9] are applied to RDAT DDS tapes operating in hostile environments. The type of scheme presented here compares favourably, subject to the memory requirements for buffering.

8. Conclusion

A synchronization scheme has been presented for variable length codes used in the presence of errors. The scheme and its decoding algorithm are designed to prevent slippage in the data stream. A variation of the scheme has also been considered. The effectiveness of the schemes has been proved, given limits on the number of errors affecting keywords in each cycle. The probability of loss of strong synchronization in the presence of random errors compares favourably to existing schemes, subject to the memory requirements for buffering.

Appendix A. — Pseudocode for Algorithm B

```
INPUT(d);   --GET CHOSEN MATCH LENGTH
INPUT(f);   --GET CHOSEN OVERLAP VALUE
INPUT(N);   --GET CYCLE LENGTH
i:=1;   --NEXT RECEIVED WORD TO BE MATCHED
k:=1;   --LAST MATCHED RECEIVED WORD (INITIALISED TO 1)
WHILE NOT end of sequence LOOP
--CHECK FOR d − 1 CONSECUTIVE KEYWORDS
   matched:=TRUE;
   j:=i;
   WHILE matched AND j < i + d − 2 LOOP
      IF for some p, r_j = c_p AND r_{j+1} ≠ c_{(p+1) mod N} THEN matched:=FALSE;
      END IF;
      j:=j + 1;
   END LOOP;
   IF matched THEN i:=i + d − 2; k:=i;
--WRITE d − 1 CONSECUTIVE MATCHED KEYWORDS
      FOR l IN i − d + 2..i LOOP
         WRITE(r_l);
```

```
    END LOOP;
  END IF;
--OTHERWISE CHECK FOR d − 1 KEYWORDS FROM d CONSECUTIVE
--KEYWORDS WITH ONE INTERNAL DELETION
  IF not matched THEN j:=i; jumps:=0; deletions:=0; matched:=TRUE;
    WHILE matched AND j < i + d − 2 LOOP
      IF for some p, r_j = c_p AND r_{j+1} ≠ c_{(p+1) mod N} THEN
        jumps:=jumps + 1;
      END IF;
      IF for some p, r_j = c_p AND r_{j+1} = c_{(p+2) mod N} THEN
        deletions:=deletions + 1;
      END IF;
      IF jumps > 1 OR (jumps = 1 AND deletions = 0) THEN
        matched:=FALSE;
      END IF;
      j:=j + 1;
    END LOOP;
    IF jumps = 1 AND deletions = 1 THEN matched:=TRUE;
    ELSE matched:=FALSE;
    END IF;
    IF matched THEN i:=i + d − 2; k:=i;
--WRITE d − 1 MATCHED KEYWORDS
      FOR l IN i − d + 2..i LOOP
        WRITE(r_l);
      END LOOP;
    END IF;
  END IF;
--OTHERWISE CHECK FOR d KEYWORDS WHICH ARE CONSECUTIVE EXCEPT
--FOR ONE INTERNAL CHANGE ERROR
  IF not matched THEN j:=i; jumps:=0; changes:=0; matched:=TRUE;
    WHILE matched AND j < i + d − 1 LOOP
      IF for some p, r_j = c_p AND r_{j+1} ≠ c_{(p+1) mod N} THEN
        jumps:=jumps + 1;
      END IF;
      IF j ≠ i + d − 2 AND for some p, r_j = c_p AND
      r_{j+2} = c_{(p+2) mod N} THEN
        changes:=changes + 1; position:=j + 1;
      END IF;
      IF jumps > 2 OR (jumps = 2 AND changes = 0) THEN
        matched:=FALSE;
      END IF;
      j:=j + 1;
```

```
    END LOOP;
    IF jumps = 2 AND changes = 1 THEN matched:=TRUE;
    ELSE matched:=FALSE;
    END IF;
    IF matched THEN i:=i + d − 1; k:=i;
--WRITE d − 1 MATCHED KEYWORDS EXCLUDING CHANGED WORD
      FOR l IN i − d + 1..i LOOP
        IF l ≠ position THEN WRITE(r_l); END IF;
      END LOOP;
    END IF;
  END IF;
--OTHERWISE CHECK FOR d + 1 KEYWORDS WHICH ARE CONSECUTIVE
--EXCEPT FOR ONE INTERNAL INSERTION
  IF not matched THEN j:=i; jumps:=0; insertions:=0; matched:=TRUE;
    WHILE matched AND j < i + d LOOP
      IF for some p, r_j = c_p AND r_{j+1} ≠ c_{(p+1) mod N} THEN jumps:=jumps + 1;
      END IF;
      IF j ≠ i + d − 1 AND for some p, r_j = c_p AND
        r_{j+2} = c_{(p+1) mod N} THEN
        insertions:=insertions + 1; position:=j + 1;
      END IF;
      IF jumps > 2 OR (jumps = 2 AND insertions = 0) THEN
        matched:=FALSE;
      END IF;
      j:=j + 1;
    END LOOP;
      IF (jumps = 1 OR jumps = 2) AND insertions = 1 THEN matched:=TRUE;
      ELSE matched:=FALSE;
      END IF;
      IF matched THEN i:=i + d; k:=i;
--WRITE d MATCHED KEYWORDS EXCLUDING INSERTION
        FOR l IN i − d..i LOOP
          IF l ≠ position THEN WRITE(r_l); END IF;
        END LOOP;
      END IF;
   END IF;
--IF NO MATCH DETERMINE START OF SEARCH FOR NEW MATCH
--START NEW SEARCH AFTER FIRST JUMP
   IF not matched THEN consecutive:=TRUE;
      WHILE consecutive AND j < i + d − 2 LOOP
        IF for some p, r_j = c_p AND r_{j+1} ≠ c_{(p+1) mod N} THEN
          consecutive:=FALSE;
```

```
                END IF;
                j:=j + 1;
            END LOOP;
            i:=j;
        END IF;
        WHILE matched LOOP --EXTEND MATCH IF POSSIBLE
            IF for some p, r_i = c_p AND r_{i+1} = c_{(p+1) mod N} THEN
                WRITE(r_{i+1}); k:=k + 1;
            END IF;
            IF for some p, r_i = c_p AND r_{i+1} ≠ c_{(p+1) mod N} THEN
                consecutive:=FALSE;
            END IF;
            i:=i + 1;
        END LOOP;
    --NOW SKIP OVER OVERLAPPING KEYWORDS
        WHILE k > 1 AND r_k = c_p for some p AND
            r_i ∈ {c_p, c_{(p−1) mod N}, c_{(p−2) mod N}, ..., c_{(p−f) mod N}} LOOP
            i:=i + 1;
        END LOOP;
    END LOOP;
```

References

[1] A.E. Escott, S. Perkins, Binary Huffman equivalent codes with a short synchronizing codeword, IEEE Trans. Inform. Theory 44 (1) (1998) 346–351.

[2] T.J. Ferguson, J.H. Rabinowitz, Self-synchronizing Huffman codes, IEEE Trans. Inform. Theory 30 (4) (1984) 687–693.

[3] W-M. Lam, S.R. Kulkarni, Extended synchronizing codewords for binary prefix codes, IEEE Trans. Inform. Theory 42 (3) (1996) 984–987.

[4] J.C. Maxted, J.P. Robinson, Error recovery for variable length codes, IEEE Trans. Inform. Theory 31 (6) (1985) 794–801.

[5] M. Nelson, The Data Compression Book. M & T Publishing, North America, Prentice-Hall International (UK) Ltd, England, 1991.

[6] B. Rudner, Construction of minimum-redundancy codes with an optimum synchronizing property, IEEE Trans. Inform. Theory 17 (4) (1971) 478–487.

[7] J.J. Stiffler, Theory of Synchronous Communications, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[8] O.D.J. Thomas, D.H. Smith, A. Ryley, Synchronized lossless data compression for DDS tape storage, in: K.A.S. Immink, A.J. Han Vinck (Eds.), Proceedings of the 1995 International Symposium on Synchronization, Saalbau, Germany 1995, pp. 12–15.

[9] O.D.J. Thomas, D.H. Smith, A. Ryley, An Adaptive error-tolerant and lossless data compressor for DDS, J. Inform. Recording 23 (1997) 547–557.