Analysis of the computational and storage requirements for the minimum-distance decoding of convolutional codes

W-H. Ng, M.Sc., Ph.D., M.I.E.E.E., and R.M.F. Goodman, B.Sc., Ph.D.

Indexing terms: *Coding, Decoding*

Abstract

In this paper we present the analytical results of the computational requirement for the minimum-distance decoding of convolutional codes. By deriving upper bounds for the number of decoding operations required to advance one code segment, we show that many less operations are required than in the case of sequential decoding This implies a significant reduction in the severity of the buffer-overflow problem. Then, we propose several modifications which could further reduce the computational effort required at long back-up distance. Finally we investigate the trade-off between coding-parameters selection and storage requirement as an aid to quantitative decoder design. Examples and future aspects are also presented and discussed.

List of symbols

v = received sequence
ω = tentatively decoded sequence
t = test-error sequence
t_b = sequence consisting of the last b segments of t
$|t|$ = weight of t
b_t = maximum back-up distance in segments for a given $|t|$
b_t^* = required back-up distance in segments
P = permissible path
$T^*(b)$ = back-up distance threshold condition
K = constraint length of the code in segments
b_m = maximum back-up distance in segments over which direct mapping operates
N = upper bound on the number of computations
N_s = upper bound on the number of computations for sequential decoding
N^* = maximum number of computations
L = maximum decoding search length in segments
$d(k)$ = code distance over k segments
b_s = maximum back-up distance for the back-up search procedure, in segments
b_p = maximum back-up distance for the permissible path search procedure, in segments
n_j = number of paths of weight j in the lower-half initial code tree

1 Introduction

Sequential decoding is well known in achieving low b.e.r. with minimum E_b/N_0 requirement. Based on our previous work, we proposed a minimum-distance decoding scheme for convolutional codes,[1,2] which uses the distance and structure properties of convolutional codes to significantly reduce the computational effort. In this paper, we quantitatively assess the number of decoding operations required by the proposed algorithm and show that this is indeed much less than that required by sequential decoding. In addition, by dividing the decoding procedure into several regions in terms of required back-up distance, we suggest modifications to the algorithm which result in even further computational reductions.

In sequential decoding, the complexity is insensitive to the constraint length K and this allows the decoder to utilise a much longer code than that used by usual convolutional decoders to achieve an outstanding performance in extremely low b.e.r. region. But the progress of the decoding is highly variable, involving both forward extensions and back-up searches to find a tentatively decoded path satisfying the current metric conditions. Because data are transmitted at a constant rate, a buffer is required, and this then raises the possibility of buffer overflow. Indeed, with sequential decoding, it is the probability of buffer overflow that limits the effectiveness of the decoder. Any decoding scheme that reduces the number of computations required per message bit therefore reduces decoding delay and improves the output error rate achievable with a fixed buffer, or, conversely, reduces the size of buffer needed to achieve a given output error rate. Fig. 1 shows the distribution of computational effort for a typical Fano decoder operating at $R \approx R_{comp}$. The probability of the number of computations required per branch N_r exceeding the number of allowed computations per branch N_a is plotted, where the unit of computation is taken to be the examination of one branch of the code tree by the decoder. Hence, by utilising the minimum-distance decoding algorithm, a much more advantageous trade-off between buffer size and output bit error rate can be established than in the case of sequential decoding.

The basic strategy adopted in the minimum-distance decoding algorithm is to always seek a path at minimum distance from the received sequence, at every node extension. Compared with the sequential decoding, this enables us to achieve a significant reduction in the maximum number of decoding operations, in two main ways. First, a direct-mapping scheme is utilised to find directly the minimum-distance path in a single operation without the need for a back-up search. Secondly, when a back-up search is required, an efficient search procedure that directly identifies the possible nodes at which path divergence might have occurred is instigated.

In this paper, we analyse the maximum number of computations required to advance one branch when using the algorithm, and com-

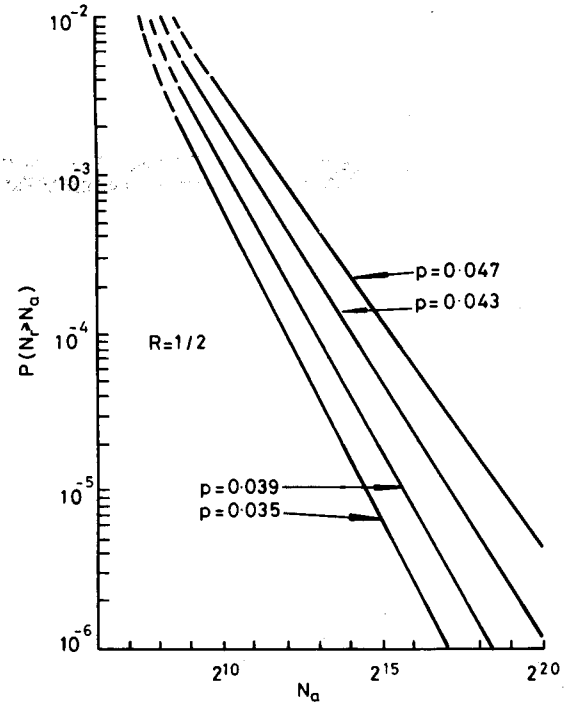

Fig. 1
Distribution of computational effort for a typical sequential decoder

Metric ration = 1/−9
Threshold spacing = 10
Variable channel-error probability = p

Paper 8244 E, first received 12th June and in revised form 29th September 1978

Dr. Goodman is, and Dr. Ng was formerly, with the Department of Electronic Engineering, University of Hull, Hull HU6 7RX, England. Dr. Ng is with the Aerospace Corporation, Los Angeles, California 90009, U.S.A.

0020–3270/79/010029 + 06 $01·50/0

pare this with the maximum required for sequential decoding, thereby showing a significant reduction in effort. Then, by dividing the decoding procedure to be adopted into several ranges of back-up distance, we introduce modifications of the algorithm in the area of suboptimum decoding. Future aspects of utilising the proposed decoding algorithm to achieve a maximum-likelihood decoding with long code are also discussed.

2 Basic minimum-distance decoding algorithm

In this Section, we will review briefly the basic minimum-distance decoding algorithm.[1,2] Consider the following notation:

ν = the received sequence, which may differ from the transmitted sequence due to channel errors

ω = the tentatively decoded path, a path in the code tree which is being compared with ν, and is the decoder's current estimate of the corresponding transmitted path.

$t = \omega \oplus \nu$ = the test-error sequence, which has ones in those positions where ω and ν differ

$|t|$ = the weight of t

t_b = the last b branches of t.

The code studied in this paper is a single-generator systematic rate 1/2 code with constraint length $K = 50$ segments. The generator sequence is $g = 3101010110010111000110001111110010110011010101010110$ in quaternary form; that is, [00] = 0, [01] = 1, [10] = 2, and [11] = 3.

We now summarise the basic procedures used in the algorithm:

(*a*) *Basic branch operation (b.b.o.).*
Whenever an ω is found which is known to be at minimum distance $|t|$ from ν, the decoder carries out the b.b.o. to select the next segment of ω, ω_1, which is the tentative version of the corresponding transmitted segment. For the code in this study, we impose a rule that the b.b.o. must choose an ω_1 that results in a $t_1 = 0$ or 1, and eliminate the possibility of $t_1 = 2$. Hence, we always have $|t_1| \leqslant 1$.

(*b*) *Maximum back-up distance*
When the b.b.o. results in a $t_1 = 0$ we can be sure that ω has minimum test-error weight, and the decoder can return to the b.b.o. after outputting the oldest segment of ω as the final decoded version of the corresponding transmitted segment. However, if $t_1 = 1$ it is possible that another path ω' may have smaller test-error weight $|t'| = |t| - 1$. In this case, we must determine whether or not a back-up search is needed, and, if so, how far to back up. The maximum back-up distance in branches b_t depends on the current value of $|t|$, and is tabulated in Table 1.

(*c*) *Required back-up distance*
If the value of b_t is $\leqslant b_m$, the range over which direct mapping operates, then no search is needed and ω' is found by a single mapping operation. If $b_t > b_m$, we can identify a set of nodes $b_t^* \leqslant b_t$ at which ω' might have diverged from ω and instigate the search procedure at each of these nodes to try to find ω'. The necessary condition for instigating a subsearch at back-up distance $b_t^* = b$ is $|t_b| \geqslant T^*(b)$, where $T^*(b)$ depends on the distance profile of the code and is tabulated in Table 2.

(*d*) *Permissible path decoding*
Assume that there is an ω with test-error weight $|t|$, and we are searching for an ω' of the same length as ω but belonging to the opposite-half truncated tree and having a smaller test-error weight $|t'|$. In this case, ω' and t' can be derived from $\omega' = \omega \oplus P$ and $t = t \oplus P$, where P is a truncated path in the lower-half initial code tree and is called a permissible path. Searching for ω' with the aid of a specially selected set of P will be used to modify the basic algorithm to reduce decoding effort, and is described later in the paper.

(*e*) *Direct mapping (d.m.)*
Consider a set of test-error patterns t and their corresponding minimum test-error patterns t', where $|t'| = |t \oplus P| < |t|$, and the minimum length of P is $\leqslant b_m$, the range over which direct mapping operates. In the decoder we store two sets of t and P into the memory. During the decoding process, whenever the tentatively decoded sequence ω has a test-error sequence t whose last b segments t_b exactly match a pattern stored in the memory, we directly map t to $t' = t \oplus P$, and ω' to $\omega \oplus P$. This guarantees that the new tentatively decoded sequence ω' has minimum test-error weight. Once a direct mapping takes place, no more searching is needed and the decoder returns to the b.b.o. If t is such that its tail sequence does not match any stored t_b, either t has minimum test-error weight, in which case the decoder returns to the b.b.o., or else the required t_b and P are not in the memory. Hence, only when $b_t^* \geqslant b_m + 1$ do we need to use the back-up search procedures. Otherwise, at most, one direct mapping is all we need to acquire the path having minimum test-error weight.

Table 1
MAXIMUM BACK-UP DISTANCE b_t FOR DIFFERENT VALUES OF $|t|$

$\|t\|$	$d(b_t)$	b_t
1	2	1
2	3	2
3	5	6
4	7	11
5	9	16
6	11	25
7	13	33
8	15	40
9	17	48
≧10	17	50

Table 2
DISTANCE PROFILE $d(b)$ AND THRESHOLD CONDITION $T^*(b)$ ON BACK-UP DISTANCE $b_t^* = b$

b	$d(b)$	$T^*(b)$	b	$d(b)$	$T^*(b)$
1	2	2	26	11	7
2	3	2	27	11	7
3	3	3	28	11	7
4	4	3	29	12	7
5	4	3	30	12	7
6	5	3	31	12	7
7	5	4	32	12	7
8	5	4	33	13	7
9	6	4	34	13	8
10	6	4	35	13	8
11	7	4	36	14	8
12	7	5	37	14	8
13	7	5	38	14	8
14	8	5	39	14	8
15	8	5	40	15	8
16	9	5	41	15	9
17	9	6	42	15	9
18	9	6	43	15	9
19	9	6	44	16	9
20	9	6	45	16	9
21	10	6	46	16	9
22	10	6	47	16	9
23	10	6	48	17	9
24	10	6	49	17	10
25	11	6	50	17	10

3 Upper bounds on maximum number of computations

In this Section, we upper bound the maximum number of computations required to advance one segment with the use of minimum-distance decoding, and compare this with sequential decoding.

The code used is as detailed in the preceding Section, and the search length L is assumed to be long enough (say, $L \geqslant K \geqslant 50$), so that it could be a valid comparison with the equivalent sequential decoder.

Considering that the decoding proceeds with basic branch operations (b.b.o.s) and direct mappings (d.m.s); the b.b.o. is taken to be the unit of computation. We therefore assume that one d.m. takes the same amount of time as one b.b.o. The underlying assumption is that it takes approximately the same time to compare two paths, regardless of length, in the range from one to L segments.

Whenever the b.b.o. results in $t_1 = 0$, which guarantees that the path ω being followed is at minimum distance from the received sequence ν, the decoder returns to the b.b.o. thus, the minimum computation for advancing one branch is one b.b.o. If $t_1 = 1$ and the decoder indicates that one d.m. has taken place to find ω', the decoder also returns to the b.b.o. In this case, it takes two computations, one b.b.o. and one d.m., to advance one branch. However, if $t_1 = 1$, and the decoder indicates that no direct mapping has taken place and that $b_t > b_m$, then a back-up search for ω' is needed. We develop an equation for the maximum number of computations N as follows.

First, let us assume that we have to search the complete $(b_t - b_m)$ unit at back-up distance b_t, by examining every path in the unit. (Note that this assumption is for simplicity in calculating the bound and is not the actual search procedure adopted in the algorithm). Because there are $(2^{(b+1)} - 2)$ branches in a b unit of the code, this involves a $(2^{(b_t+1)-b_m} - 2)$ branch search. However, the $(b_t - b_m)$

branches belonging to the present tentatively decoded sequence have already been searched, and so the required number of branch searches is $(2^{(b_t+1)-b_m}-2)-(b_t-b_m)$. Secondly, there are $2^{(b_t-b_m)}$ paths of length b_m stemming from the end of the (b_t-b_m) unit, and each of these is searched by direct mapping. Neglecting the present tentatively decoded path, this requires a search of $2^{(b_t-b_m)}-1$ b_m-units. Each b_m-unit search could require a maximum of b_m b.b.o.s and $|t_{b_m}|_{max}$ d.m.s. Hence, the maximum number of computations for this stage is $\{(2^{(b_t-b_m)}-1)\,(b_m+|t_{b_m}|_{max})\}$. Finally, we add one computation for the original b.b.o. that resulted in $t_1=1$. The maximum number of computations for a back-up search of $b_t > b_m$ is then

$$N \leqslant \{(2^{(b_t+1)-b_m}-2)-(b_t-b_m)\} + \{(2^{(b_t-b_m)}-1)(b_m+|t_{b_m}|_{max})\}+1 \quad (1)$$

Fig. 2 illustrates the above calculation for the case $(b_t-b_m)=3$. In this case, there are 11 branch searches in the (b_t-b_m) unit, and 7 b_m-unit searches stemming from the 7 branches at the end of the (b_t-b_m) unit.

Let us now evaluate N for the decoding algorithm. We assume $b_m=16$; that is, the direct mapping range is 16 segments. This choice is determined by the memory size allowable in the decoder. For example, in Reference 2 it is shown that when $b_m=10$, we only need to store 11 permissible paths and 30 tentative test-error sequences. Even if the memory requirement grows exponentially with increasing b_m, the memory size of a decoder with $b_m=16$ is still feasible and relatively cheap to implement. The actual value of b_m is therefore up to the individual hardware designer and does not affect the general nature of our calculations.

Assuming $b_m=16$ and that $b_t=25$ and $|t_{16}|_{max}=5$, evaluation of eqn. 1 shows that the maximum number of computations $N(b_t=25)$ is equal to 11 745. If $b_t=33$, $N(b_t=33)=3\,014\,617$. If $b_t=50$, $N(b_t=50)\simeq 4\times10^{11}$.

Consider now a sequential decoder utilising the same coding parameters as the minimum-distance decoding. When such a decoder enters a back-up search, the maximum back-up distance is b_t, and, therefore, a complete b_t-unit search is required. Thus, the maximum number of computations required for sequential decoding N_s is equal to the total number of branches in a b_t-unit, minus the number of branches of ω already searched, plus the original b.b.o. Hence $N_s=(2^{b_t+1}-2)-b_t+1$. For $b_t=25$, $N_s(b_t=25)\simeq 2{\cdot}6\times10^7$, and, if $b_t=33$, $N_s(b_t=33)\simeq 1{\cdot}7\times10^{10}$. If $b_t=50$, $N_s(b_t=50)\simeq 2{\cdot}3\times10^{15}$.

From the above calculations it can be seen that the ratio of N_s/N is 2213 for $b_t=25$ and tends to a limit of $2^{b_m}/\frac{1}{2}(b_m+|t_{b_m}|_{max})+1/\simeq 5700$ for large b_t. This represents a considerable improvement over sequential decoding.

4 Determination of maximum number of computations

In this Section, we tighten the bounds on the maximum number of computations for minimum-distance decoding by allowing for the actual search procedure utilised in a back-up search. We will show that this results in an even more marked decrease in decoding effort than that presented in the preceding Section compared to sequential decoding. To facilitate calculation of the bound we divide the analysis into four sections based on four back-up distances b_t and denote the new bound on maximum computation to be N^*. On the basis of $b_m=16$, the four regions are $b_t \leqslant 16$, 25, 33 and 50, corresponding to the four test-error weight conditions $|t| \leqslant 5$, $=6$, 7 and $\geqslant 8$.

4.1 Value of N^* for $b_t \leqslant 16$

This case has been previously analysed, showing that at most one b.b.o. and one d.m. are needed if $b_t \leqslant 16$. Hence, $N^*=2$.

4.2 Value of N^* for $b_t = 25$

Let us assume that $|t_1|=1$ and $|t|=|t_{25}|=6$. We therefore have $b_t=25$ and want to search for a ω'_{25} whose $|t'_{25}|=5$. We divide the analysis into three cases based on the three possible values of the test-error weight of the first segment of t_{25}; that is, $(|t_{25}|-|t_{24}|)$. Fig. 3 illustrates each case.

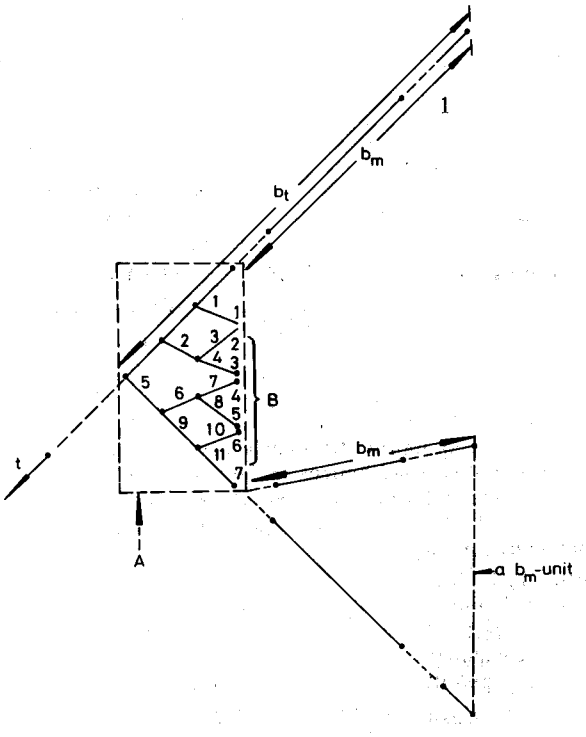

Fig. 2
Maximum number of decoding operations N for the minimum-distance decoding when $(b_t-b_m)=3$

$$N \leqslant [\,(2^{(b_t+1)-b_m}-2)-(b_t-b_m) + (2^{(b_t+b_m)}-1)(b_m+|t_{b_m}|_{max})+1\,]$$

The terms of the expression are obtained from the Figure as follows:
A = $2^{(b_t+1)-b_m}-2)-(b_t-b_m)$ = 11 = number of required branch searches in a (b_t-b_m) unit B = $(2^{(b_t-b_m)}-1)=7$ = number of branches or nodes at the end of a (b_t-b_m) unit that requires search. Each node could require b_m b.b.o.s and $|t_{b_m}|_{max}$ d.m.s. 1 = first b.b.o. that results in a $t_1=1$

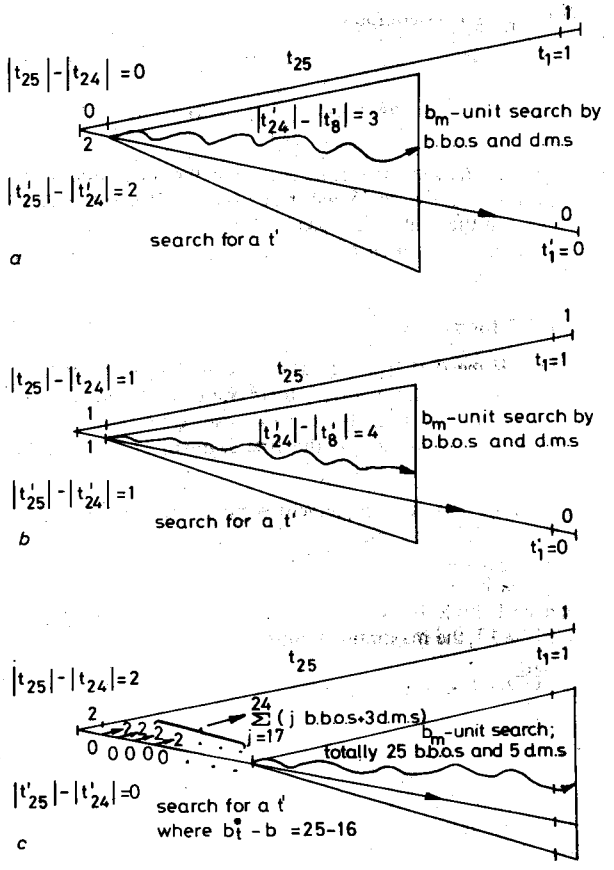

Fig. 3
Calculation of N^* for $b_t=25$ and $|t_{25}|=6$

a $|t_{25}|-|t_{24}|=0$
b $|t_{25}|-|t_{24}|=1$
c $|t_{25}|-|t_{24}|=2$

(*a*) *Fig. 3a*

If $|t_{25}| - |t_{24}| = 0$, its complement segment has weight $|t'_{25}| - |t'_{24}| = 2$. Therefore, if there is a t'_{25} stemming from $b_t^* = 25$ such that $|t'_{25}| = 5$, it will have weight $|t'_{24}| = |t'_{25}| - (|t'_{25}| - |t'_{24}|) = 5 - 2 = 3$ over the remaining 24 segments. This implies that the 24-unit can be searched by using direct mapping only, as follows.

From the distance property[1] of the code (Appendix 9), we can see that when a back-up search starts at $b_t^* > b_m$ and the t' is such that $|t'_i| - |t_{i-b_m}| \leqslant |t'| \leqslant [d(b_m) - 1]/2$, where $b_t^* \geqslant i > b_m$, the t' can be searched by using only b_t^* b.b.o.s and $|t'|$ d.m.s. In this case, $b_m = 16$ and $[d(b_m) - 1]/2 = 4$; therefore, $|t'_i| - |t'_{i-16}| \leqslant 3 < [d(16) - 1]/2 = 4$, and the 25 unit can be searched with a maximum of 25 b.b.o.s and $|t'_{24}| = 3$ d.m.s. Hence, when the range of required back-up distance b_t^* is such that every complement unit between 17 and 25 segments back must be searched; that is, $25 \geqslant b_t^* \geqslant 17$, the maximum number of computations is

$$N^* = \sum_{j=17}^{25} (j + 3) + 1 = 217$$

(*b*) *Fig. 3b*

If $|t_{25}| - |t_{24}| = 1$, its complement segment $|t'_{25}| - |t'_{24}| = 1$, and $|t_{24}| = 5$ implying $b_t^* = b_t = 25$ only. Also, $|t'_i| - |t'_{i-16}| \leqslant 5 - 1 = 4$ for $24 \leqslant i \leqslant 17$, which is equal to $[d(16) - 1]/2$, implying that the search at $b_t^* = 25$ can be carried out by means of 25 b.b.o.s and 4 d.m.s. Hence $N^* = 25 + 4 + 1 = 30$

(*c*) *Fig. 3c*

If $|t_{25}| - |t_{24}| = 2$, its complement segment $|t'_{25}| - |t'_{24}| = 0$, and $|t_{24}| = 4$ again, implying $b_t^* = b_t = 25$ only. In this case, $|t'_{25}| = 5$, which indicates that there is a possibility that $|t'_i| - |t'_{i-16}| = |t'_{25}| = 5 > [d(16) - 1]/2 = 4$, for some i within $25 \leqslant i \leqslant 17$, and so the search cannot be directly carried out with b.b.o.s and d.m.s only. The worst case situation is therefore one in which there are $(25 - 16) = 9$ consecutive zero test-error weight segments stemming from $b_t^* = 25$. In this case the b_m-unit at the end of the path is searched with direct mapping, and each of the 9 complement path segments having double test-error weight are searched in a manner similar to case (*a*). The zero test-error weight portion and the terminating b_m-unit can be searched with a maximum of 25 b.b.o.s and 5 d.m.s, and the paths stemming from the double error segments can be searched with $\sum_{j=17}^{24} (j + 3)$ computations. Hence, the maximum number of computations is

$$N^*(b_t = 25)_{max} = (25 + 5) + \sum_{j=17}^{24} (j + 3) + 1 = 219$$

From the above calculations, it can be seen that the maximum number of computations for $b_t = 25$ will not exceed 219. Not only is this significantly less than the value calculated for sequential decoding, but it is also 53 times less than the bound $N(b_t = 25)$ calculated in the preceding Section.

4.3 Value of N^* for $b_t = 33$

Let us assume that $|t_1| = 1$, $|t| = |t_{33}| = 7$, $b_t = 33$, and we are searching for a ω'_{33} whose $|t'_{33}| = 6$. If there is a t'_{33} satisfying this test-error weight condition, the analysis can again be split into three cases based on the weight of the first segment of t_{33}, that is, $|t_{33}| - |t_{32}|$:

(*a*) If $|t_{33}| - |t_{32}| = 0$, its complement segment has weight $|t'_{33}| - |t'_{32}| = 2$. Hence, $|t'_{32}| = |t'_{33}| - (|t'_{33}| - |t'_{32}|) = 4 = [d(b_m) - 1]/2 = |t_i| - |t_{i-16}|$, for $32 \geqslant i \geqslant 17$. This means that the search for t'_{33} at $b_t^* = 33$ can be carried out with a maximum of 33 b.b.o.s and 4 d.m.s. When each node between 17 and 33 needs to be searched, that is, $33 \geqslant b_t^* \geqslant 17$, the maximum number of computations is

$$N^* = \sum_{j=17}^{33} (j + 4) + 1 = 494$$

(*b*) If $|t_{33}| - |t_{32}| = 1$, its complement segment has weight $|t'_{33}| - |t'_{32}| = 1$, and $|t_{32}| = 6$. From Tables 1 and 2 it can be seen that for $|t_{33}| = 7$, and $|t_{32}| = 6$, we have $33 = b_t^* \leqslant 25$. Therefore, the maximum number of computations for this case is the sum of the computations for $b_t^* = 33$ and $b_t^* \leqslant 25$. If there is a t'_{33} stemming from $b_t^* = 33$ such that $|t'_{33}| = 6$, we would be searching for a t'_{32} with $|t'_{32}| = 5$ after accepting the first segment stemming from b_t^*. Hence, the worst-case situation is similar to that of Section 4.2 (*c*); that is, $(32 - 16) = 16$ consecutive zero test-error weight segments stemming from the first segment of t'_{33}, ending with a b_m-unit search. Also, the 16 opposite branches each having double test-error weight are searched according to case (*a*) in Section 4.2. Therefore, when the range of required back-up distance b_t^* is such that $33 = b_t^* \leqslant 25$, the maximum number of computations is

$$N^*(b_t = 33)_{max} = N^*(b_t = 25)_{max} + (33 + 5) + \sum_{j=17}^{32} (j + 3)$$
$$= 219 + 38 + 440$$
$$= 697$$

(*c*) If $|t_{33}| - |t_{32}| = 2$, its complement segment has weight $|t'_{33}| - |t'_{32}| = 0$ and $|t_{32}| = 5$. Therefore, $b_t^* = 33$ only. After accepting the first segment stemming from b_t^*, we consider two cases based on the test-error weight of the two branches stemming from the accepted node. These are detailed in Fig. 4.

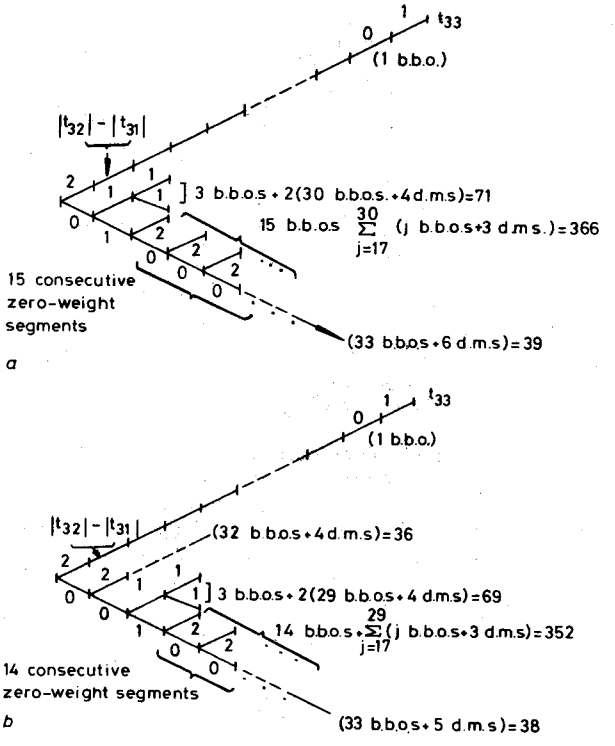

Fig. 4

Determination of the maximum number of computations N^* for searching a t'_{33} with $|t'_{33}| = 6$ when $|t_{33}| = 7$, $b_t = 33$ and $|t_{33}| - |t_{32}| = 2$

a Distribution of test-error weight when $|t'_{32}| - |t'_{31}| = 1$ $N^* = 1 + 71 + 366 + 39 = 477$

b Distribution of test-error weight when $|t'_{32}| - |t'_{31}| \neq 1$ $N^* = 1 + 36 + 69 + 352 + 38 = 496$

4.3.1 Details of Fig. 4*a*

If the test-error weight of both branches is 1, that is $|t'_{32}| - |t'_{31}| = 1$, there are four contributions to the total number of computations. These are as follows: First, there is the original b.b.o. that resulted in $|t_1| = 1$ and $|t_{33}| = 7$. Secondly, the maximum number of computations for the search stemming from the two segments with weight $|t'_{31}| - |t'_{30}| = 1$ is equal to 3 b.b.o.s + {2(30 b.b.o.s + 4 d.m.s)} = 71. Thirdly, the search stemming from the segment with weight $|t'_{31}| - |t'_{30}| = 0$ requires a maximum of (33 b.b.o.s + 6 d.m.s) = 39 computations. Fourthly, the searches stemming from those branches opposite to the consecutive zero test-error weight segments require a maximum of 15 b.b.o.s + $\sum_{j=17}^{30} (j$ b.b.o.s + 3 d.m.s) = 366 computations. The maximum number of computations in this case is therefore

$$N^* = (1 + 71 + 39 + 366) = 477$$

4.3.2 Details of Fig. 4*b*

If the test-error weight of the two branches is 2 and 0, that is, $|t'_{32}| - |t'_{31}| \neq 1$, there are five contributions to the total number of computations. These are as follows. First, there is the original b.b.o. which gave $|t_{33}| = 7$. Secondly, the search stemming from the segment with weight $|t'_{32}| - |t'_{31}| = 2$ requires a maximum

of (32 b.b.o.s + 4 d.m.s) = 36 computations. Thirdly, the search stemming from the segment with weight $|t'_{31}| - |t'_{30}| = 1$ and the two subsequent parallel segments each of weight $|t'_{30}| - |t'_{29}| = 1$ requires [3 b.b.o.s + {2(29 b.b.o.s + 4 d.m.s)}] = 69 computations. Fourthly, the search stemming from the path containing 14 consecutive zero weight segments requires a maximum of (33 b.b.o.s + 5 d.m.s) = 38 computations. Finally, the search stemming from those branches opposite to the consecutive zero test-error weight segments requires 14 b.b.o.s + $\sum_{j=17}^{29}$ (j b.b.o.s + 3 d.m.s) = 352. Therefore, the maximum number of computations for this case is

$$N^* = 1 + 36 + 69 + 38 + 352 = 496$$

From the calculations presented above, it can be seen that the maximum number of computations for $b_t = 33$ will not exceed 697. This is an extremely small amount when compared with sequential decoding, and is 4325 times less than the value of $N(b_t = 33)$ presented in Section 3.

4.4 Value of N^* for $b_t \geqslant 34$

By extending the similar technique utilised in the preceding Section and using the weight distribution of the initial code tree, we can calculate the maximum number of computations required at back-up distance greater than 33. However, let us at this point say that we wish to restrict the number of computations in a back-up search to be an absolute minimum value in order to have a very small buffer. We must therefore modify the algorithm to cope with searches at back-up distances of $b_t \geqslant 34$. This is dealt with in the following Section.

5 Searches at $b_t \geqslant 34$ using permissible path decoding

Consider that we have a ω with test-error weight $|t|$ and we are searching for a ω' with test-error weight $|t'| < |t|$. In this case, ω' and t' can be found from $\omega' = \omega \oplus P$, and $t' = t \oplus P$, where P is one of a set of truncated patterns from the lower-half initial code tree and is denoted a permissible path. Unfortunately, it is not possible to store all the possible P of length $\geqslant 34$ because of the large memory this would entail. This can be seen by examining one of the weight conditions on P which is $|P| < 2|t|$. In general, $|t|_{max}$ increases with increasing search length L and $|P|_{max}$ increases linearly with $|t|_{max}$ and, therefore, the number of permissible paths satisfying the weight condition $|P| < 2|t|_{max}$ could exponentially increase with $|t|_{max}$. Fortunately, we are obtaining results that show that the number of permissible paths can be reduced by simply limiting the maximum weight of P, and that the effect of this path reduction on coding gain is extremely small, even if the maximum weight of P is reduced to $|P|_{max} \leqslant d(L)$, where $d(L)$ is the minimum distance of the code over L segments.[3]

Let us therefore evaluate the approximate amount of storage needed to store a reduced set of P by estimating the weight structure of the code.

Minimum distance and weight spectrum as a function of constraint length has been studied by using a sequential decoder simulator to analyse the structure of different half-rate systematic codes.[4] From this study we may conclude that, among the good codes, there is very little difference in the weight distribution, and that the increase in distance with constraint length is consistent with a relationship of the form $d(K) = C + 0{\cdot}22K$, where C is a constant of about 2 to 4 and 0·22 is a factor equal to the asymptotic ratio of distance/constraint length for a half-rate code, based on the Gilbert bound. Table 3 shows a typical weight distribution for a half-rate code, where k indicates the length of the path in segments and the value of n_j indicates the number of paths of weight j. For example, it can be seen that there are 12 paths of length 33 which have weight 13. Also, the Table shows that $d(i)_{min} = d(i+h)_{min}$ for $i = 33, 37, 41, 45, 49$ and $0 \leqslant h \leqslant 3$. From the structure of convolutional codes, we can estimate the number of minimum-distance paths for values k between those given in the Table, by interpolation. Because 50% of code branch pairs stemming from a given node have weight 0 and 2 and the other 50% have weights equal to 1, we would expect that $n_j(k=i) \simeq 2n_j(k=i+1) \simeq 4n_j(k=i+2) \simeq 8n_j(k=i+3)$, where $j = d(i)_{min} = d(i+h)_{min}$. We can then estimate the number of minimum distance paths for $34 \leqslant k \leqslant 50$. For example, as $n_{13}(k=33) = 12$, $n_{13}(k=34) \simeq 6$ etc. Using this method, the total number of minimum-distance paths works out to be in the region of 124. We may now consider two ways of reducing the number of P stored for $34 \leqslant b_t^* \leqslant 50$.

Table 3
WEIGHT DISTRIBUTION OF THE LOWER-HALF INITIAL CODE TREE OF A TYPICAL HALF RATE CODE

k	n_{13}	n_{14}	n_{15}	n_{16}	n_{17}	n_{18}
33	12	142	848	4428	18066	65294
37		13	144	1019	5396	24156
41			12	170	1103	6629
45				16	172	1333

In the first case we apply the general rules for permissible path selection, that is, $|P|$ is odd and $P_2 = 01$. In addition, we impose the restriction $|P| \leqslant d(L=50) = 17$ that is, $|P| = 15$ and 17 only. From interpolation of the values in Table 3, and by knowing that roughly 1/16 of all paths of a given length end in 01, we can estimate that the total number of paths needed to be sorted is several thousand. Hence, a maximum of several thousand path-comparison operations could be performed in search for a ω' with $|t'| < |t|$, via $\omega' = \omega \oplus P$.

In the second case, we restrict the selection of permissible paths in such a way that (a) only minimum weight paths are stored and (b) if the weight is odd $P_2 = 01$, or if the weight is even $P_1 = 0$. Only a very few of the estimated 124 minimum-weight paths would satisfy the above two conditions, indicating that an exceedingly small memory is sufficient to store the permissible paths in this case. The decoder would then proceed as follows whenever a back-up search in the range $34 \leqslant b_t^* \leqslant 50$ is required:

(i) If $d(b_t^*)_{min}$ is odd, Fig. 4*a*, a search will be carried out at $b_t^* = b$ to find a t' with $|t'| = |t \oplus P| < |t|$, where the P are those stored permissible paths with length equal to b segments long. When such a t' is found, we will return to the b.b.o. Otherwise, go to (iii).

(ii) If $d(b_t^*)_{min} = d(b_t^* - 1)_{min}$ is even, Fig. 4*b*, a back-up search at $b_t^* = b$ will be carried out as follows. We first denote t_{-1} as the portion of t without the last segment t_1, which therefore has length $(b-1)$ segments. The same applies to t'_{-1} and t'. We then search for a t'_{-1} with $|t'_{-1}| = |t'_{-1} \oplus P| < |t_{-1}|$ where the P are those stored permissible paths with length $(b-1)$ segments. When such a t'_{-1} is found, we extend it with the b.b.o. to derive a t'_1. If the b.b.o. results in a $t'_1 = 0$, we accept t' as the minimum-weight path. Otherwise, go to (iii). For the case of $d(b_t^*)_{min}$ even, but $d(b_t^* - 1)_{min}$ is odd, ω is accepted as the minimum-weight path without any search.

(iii) We go to the next value of b_t^*, or if this is $(b_t^*)_{max}$, we accept ω as the best path and return to the b.b.o. In the latter case, the oldest segment of ω that is output as the corresponding segment of the transmitted sequence could be in error. If this happens, however, the decoder will eventually receover.

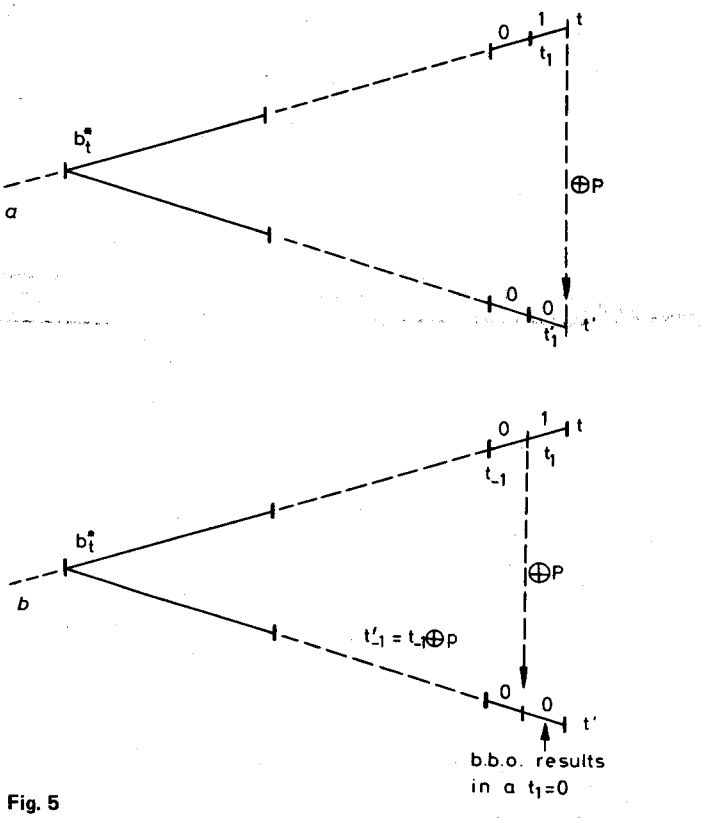

Fig. 5
Using permissible-path decoding to search for a t' with $|t'| < |t|$ at $34 \leqslant b_t^* \leqslant 50$

a $d(b_t^*)$ is odd
t' exists if there is a P having b_t^* segments long such that $|t'| = |t \oplus P| < |t|$
b $d(b_t^*) = d(b_t^* - 1)$ is even
t' exists if (i) there is a P having $(b_t^* - 1)$ segments long such that $|t'_{-1}| = |t_{-1} \oplus P|$ and (ii) b.b.o. results in a $t'_1 = 0$

6 Discussion

In this paper, we have analysed the computational and storage requirements of the proposed minimum-distance decoding algorithm. By adopting different search techniques at different stages in the search procedure we obtain an efficient trade-off between coding parameter selection and memory requirement. We can now discuss the various options available when implementing the algorithm by dividing the back-up distance into three regions: b_m, b_s and b_p, where $0 \leqslant b_m \leqslant b_s \leqslant b_p \leqslant L$.

First, we would utilise direct mapping for all back-up searches at distance $b_t^* \leqslant b_m$, because the maximum number of computations $N^*(b_t \leqslant b_m)_{max}$ is only equal to 2. This is the direct-mapping or b_m region. The bigger the b_m region, the smaller the buffer requirements, but the larger the decoder path storage requirement.

Secondly, the region of back-up distance that uses the minimum-distance search procedure is denoted the b_s region, where $b_s \geqslant b_m$. The bigger the value of b_s, the bigger the buffer requirement, especially if $b_s \geqslant 40$.

Thirdly, we denote the long back-up distance region that requires a larger buffer size, and that often causes buffer overflow in sequential decoding, as the b_p region. In this region, we use permissible path decoding with path reduction on the total number of paths, and this implies that the decoding is now suboptimum. Two different path-reduction techniques are used: one with $|P| \leqslant d(L)_{min}$ and another with $|P| = d(k)_{min}$. The value of k used could be either $L \geqslant k > b_s$ or $L \geqslant k > b_m$ depending on the trade-offs required. The $|P| = d(k)_{min}$ approach requires much smaller memory and much fewer computations than the $|P| \leqslant d(L)_{min}$ approach, but will result in a slight loss of coding gain in the lower signal/noise-ratio region.

As has been shown, the minimum distance decoding at present requires significantly less computational effort than sequential decoding, resulting in a much reduced probability of dismissal. In a previous paper,[5] we proposed a bidirectional search for convolutional codes which could achieve maximum-likelihood decoding with long codes, and the decoding algorithm discussed in this paper could easily be adopted in both the forward and backward decodings of the bidirectional search procedure. Further computational analysis is proposed for the bidirectional search and hybrid convolutional coding system with the aid of involved computer-simulation study.

7 Acknowledgments

This work was completed while Dr. Ng was with the University of Hull and before he joined the Aerospace Corporation.

8 References

1 NG, W-H.: 'An upper bound on the back-up depth for maximum likelihood decoding of convolutional codes', *IEEE Trans.*, 1976, **IT-22**, pp. 354–357
2 NG, W–H., and GOODMAN, R.M.F.: 'An efficient minimum-distance decoding algorithm for convolutional error-correcting codes', *Proc. IEE*, 1978, **125**, (2), pp. 97–103
3 NG, W-H., KIM, F., and TASHIRO, S.: 'Maximum likelihood decoding scheme for convolutional codes', *ITC Record*, Los Angeles, California, 1976
4 FORNEY, D. Jr.: 'High-speed sequential decoder study'. Contract DAA B07-68-C-0093, Codex Corp., 1968
5 NG, W-H.: 'Bidirectional search for convolutional codes', *Proc. IEE*, 1978, **125**, (6), pp. 495–500

9 Appendix

Fundamental distance property

Convolutional codes are group codes, and if ω and ω' are paths in opposite halves of any k-unit, then $x = \omega \oplus \omega'$ is a code path in the lower-half initial code tree. Therefore, the distance between half trees of any k-unit depends only on k and not on which k-unit is chosen, and is equal to the minimum weight path in the lower-half initial code tree.