An efficient minimum-distance decoding algorithm for convolutional error-correcting codes

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

Indexing terms: Decoding, Error correction codes

Abstract

Minimum-distance decoding of convolutional codes has generally been considered impractical for other than relatively short constraint length codes, because of the exponential growth in complexity with increasing constraint length. The minimum-distance decoding algorithm proposed in the paper, however, uses a sequential decoding approach to avoid an exponential growth in complexity with increasing constraint length, and also utilises the distance and structural properties of convolutional codes to considerably reduce the amount of tree searching needed to find the minimum-distance path. In this way the algorithm achieves a complexity that does not grow exponentially with increasing constraint length, and is efficient for both long and short constraint length codes. The algorithm consists of two main processes. Firstly, a direct-mapping scheme, which automatically finds the minimum-distance path in a single mapping operation, is used to eliminate the need for all short back-up tree searches. Secondly, when a longer back-up search is required, an efficient tree-searching scheme is used to minimise the required search effort. The paper describes the complete algorithm and its theoretical basis, and examples of its operation are given.

List of symbols

b_m = maximum back-up distance in segments that can be performed by direct mapping
V = number of code digits in one branch of a single-generator-sequence convolutional code
K = constraint length of the code in segments
$\mathscr{G}$ = generator sequence
$g(2^i)$ = an arbitrary code segment in $\mathscr{G}$ where i is a positive integer $0 \leqslant i \leqslant K-1$
S = initial code tree
S_0 = upper-half initial code tree
S_1 = lower-half initial code tree
$d(k)$ = minimum distance between half-trees of any k-unit
v = received sequence
w = tentatively decoded sequence
t = test-error sequence
t_b = the sequence consisting of the last b segments of t
$|t|$ = the weight of t
$P_{(i)}$ = the ith permissible path stored in the memory, a code path selected from S_1 according to a set of criteria
b_t = maximum back-up distance in segments for a given $|t|$
b_0 = maximum back-up distance in segments after the back-up reduction operation
b_t^* = required back-up distance in segments

1 Introduction

It is well known that convolutional codes are capable of performing better than block codes in most error-control applications. For a particular application, the realisation of this superiority depends on the efficiency and practicability of the decoding algorithm used. In general, maximum-likelihood decoding (in the minimum-distance sense) of short constraint length codes can be achieved by using the Viterbi algorithm. However, to achieve low probabilities of sink bit error rate ($< 10^{-5}$) with minimum signal/noise ratio requirements, it is necessary to use codes with long constraint length, and this renders the usual Viterbi decoder impractical on the grounds of complexity. In this case non-maximum-likelihood sequential decoding[1] is usually used, because its complexity is insensitive to constraint length. This paper presents a minimum-distance decoding scheme whose complexity does not grow exponentially with constraint length, and which requires much less decoding effort than normal sequential decoding, because of the elimination of needless tree searching.

If a convolutional code is represented by its (semi-infinite) tree structure (Fig. 1), then encoding can be considered as the selection of a path through the tree, one branch at a time, in accordance with the message digits. The decoding operation then consists of determining the correct path through the tree, given that the received digit sequence, on which this determination is based, may contain errors.

A normal sequential decoder operates by computing the value of a suitable metric based on the distance between the received sequence and the (tentative) path being followed. If the metric exceeds some running threshold, it indicates that the decoder may be following the wrong path and that it is necessary to search for a better one. The decoder then backs up in a node-by-node manner, and searches for a path that has a better metric value. If a better path is found then decoding continues along this new path, subject to the threshold conditions being satisfied. If a better path cannot be found then either the threshold value is loosened or a buffer overflow may occur. Because the number of branches rises exponentially with depth in the tree, it can be seen that the maximum decoding effort of such a scheme could also rise exponentially with back-up distance. Several efficient decoding algorithms have been proposed,[2,3] but even so, the performance of a sequential decoder is directly related to the time available for searching the tree, that is, the probability of a buffer overflow. In addition, decoder operation is not maximum likelihood, because any path that is chosen is not guaranteed to be the path at minimum distance from the received sequence, but rather a path that satisfies the threshold conditions.

The algorithm presented in this paper is maximum likelihood in that at every node the path chosen is guaranteed to be the path at minimum distance from the received sequence. On the face of it, such a decoding scheme would appear to be impractical, because every path in the entire code tree would have to be tested at every forward node extension to guarantee minimum distance from the received sequence. However, the advantage to be gained from minimum-distance decoding is the capability of spotting incorrect decoding paths as early as possible. This has the effect of halving the number of branch-search operations for every one segment reduction in back-up distance. The algorithm presented differs from other convolutional decoding schemes in that it finds the minimum-distance path, and utilises the distance and structural properties of the particular convolutional code used, to eliminate the need for testing the whole tree and also to substantially reduce the required decoding effort in two main ways. Firstly, all short searches with a back-up distance of up to b_m nodes are eliminated by a direct mapping scheme which guarantees that the path chosen is at minimum distance from the received sequence. Thus a maximum of $2^{(b_m+1)}-2$ branch searches is replaced by a single mapping operation. The value of b_m depends on the storage available, and would typically be in the range 10–20 for a half-rate code. Secondly, when a back-up is required (because the path we are searching for diverges at more than b_m nodes back and cannot therefore be mapped to) we can not only derive a maximum back-up distance, but also determine the exact nodes at which the divergence might have occurred. As the number of these nodes is considerably less than the total number of nodes between b_m and the maximum back-up distance, the number of searches required (which increases exponentially with every node back) is very significantly reduced.

For reasons of brevity the discussion in this paper is limited to hard-decision decoding of binary half-rate single-generator convolutional codes. The approach used, however, can be extended to other codes and to soft-decision decoding.

This paper develops in the following way. Firstly we introduce the distance and structural properties of convolutional codes that are utilised in the algorithm, and describe the basic decoding strategy. Next, the concept of decoding with permissible paths is described, and then this is developed into the direct-mapping scheme for eliminating all short back-up searches. The technique for minimising the number of actual back-up searches is then outlined, and finally the algorithm is summarised and discussed.

Paper 8028 E, first received 10th March and in revised form 30th August 1977

Mr. Ng and Dr. Goodman are with the Department of Electronic Engineering, University of Hull, Hull, England

0020-3270/78/8028-0097 $1·50/0

2 Convolutional codes and their structural properties

In this section we introduce some of the distance and structure properties of single-generator convolutional codes that are utilised in the decoding algorithm.

A single-generator convolutional code is one in which each message digit is encoded individually into V code digits, where V is a positive integer, giving a maximum information rate of $1/V$. The V code digits for each message digit depend on both the present message digit and the $K-1$ previous message digits, where K is the contraint length of the code in segments. Such a code is generated by a K-segment generator sequence $\mathscr{G} = g(2^0)g(2^1)g(2^2)\ldots g(2^{K-1})$ and is a systematic code if the first digit of each code segment is the same as the corresponding message digit. The code can be represented by its tree structure, the branches of which can be extended indefinitely from any node (Fig. 1). Each branch has one segment of code digits associated with it, and the code digits of the two branches stemming from an arbitrary node are always ones-complements of each other. Fig. 2 shows the first five segments of the code tree for the rate one-half code used as an example in this paper, which has a fifty-segment generator sequence.

The encoding operation is one of selecting a path through the tree in accordance with the message digits. At each node the upper branch is taken if the message digit is a zero, and the lower branch is taken if it is a one.

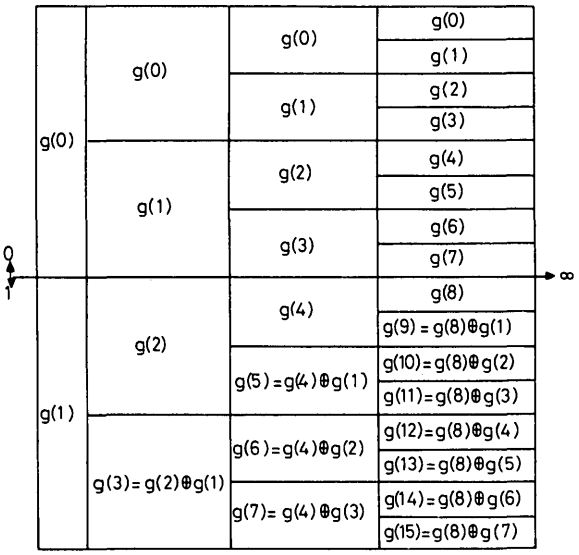

Fig. 1
The development of a single-generator initial code tree, where $\mathscr{G} = g(1)g(2)g(4)\ldots g(2^{K-1})$

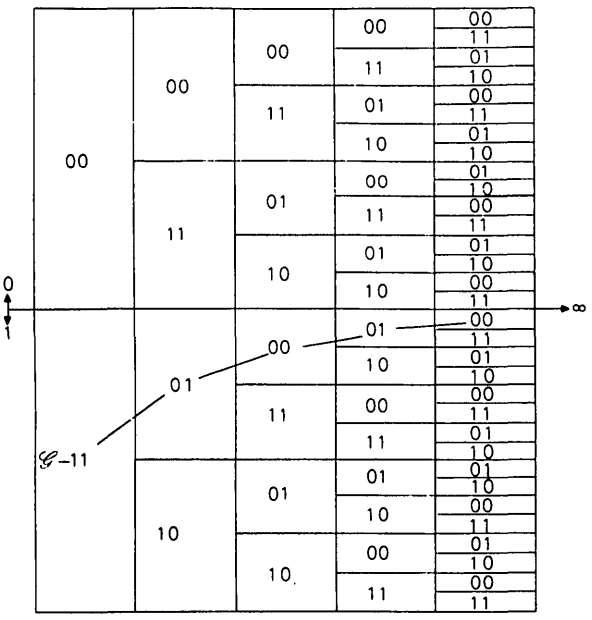

Fig. 2
The development of the initial code tree for the half-rate code with $\mathscr{G} = 11\ 01\ 00\ 01\ 00\ldots g(2^{K-1})$

Consider, for any node in the infinite tree, all the paths that extend k segments forward from that node. The resulting subtree is referred to as a truncated tree, or k-unit, and is divided into two half-trees depending on which branch was chosen at the first node. The initial code tree (S) is the k-unit stemming from the very first node, and is divided into the upper- and lower-half initial code trees (S_0 and S_1, respectively).

We may now summarise several useful properties of these codes.

(*a*) The code is a group code. That is, if w and w' are two equal-length code paths, belonging to the initial truncated tree S, it implies that there is a path x such that $x = w \oplus w'$ is within S.

(*b*) If w and w' are paths in *opposite* halves of *any* k-unit, then $x = w \oplus w'$ is a code path in the lower-half *initial* code tree S_1.

(*c*) The distance between the two half trees of *any* k-unit is defined as the minimum Hamming distance between pairs of paths, one from each half tree. Consider the initial code tree. Because of the group property, the minimum distance between the two halves of the initial code tree is equal to the minimum distance between the all-zero vector and all the paths in S_1, that is, the minimum distance equals the weight of the minimum-weight code path in S_1.

(*d*) Combining properties (*b*) and (*c*) above, we can state that the minimum distance between half trees of *any* k-unit is equal to the weight of the minimum-weight path in S_1. We can then define a distance function $d(.)$ such that $d(k)$ is the minimum distance between half trees of any k-unit, and depends only on k, and not on the k-unit chosen. The guaranteed error-correcting capability of any k-unit is then $T(k)$, where $T(k)$ is the largest integer such that $T(k) \leqslant [d(k)-1]/2$. Table 1 shows the distance function $d(.)$ for the half-rate code used in this paper.

(*e*) From properties (*b*) and (*d*) we can easily see that $|w \oplus w'| \geqslant d(k)$ and $|w'| \geqslant d(k) - |w|$, where $|w|$ denotes the weight of the sequence w.

Table 1
DISTANCE FUNCTION $d(.)$ FOR RATE ONE-HALF CODE

k	$\mathscr{G}$	$d(k)$	k	$\mathscr{G}$	$d(k)$
1	11	2	26	01	11
2	01	3	27	01	11
3	00	3	28	01	11
4	01	4	29	01	12
5	00	4	30	01	12
6	01	5	31	00	12
7	00	5	32	00	12
8	01	5	33	01	13
9	01	6	34	00	13
10	00	6	35	01	13
11	00	7	36	01	14
12	01	7	37	00	14
13	00	7	38	00	14
14	01	8	39	01	14
15	01	8	40	01	15
16	01	9	41	00	15
17	00	9	42	01	15
18	00	9	43	00	15
19	00	9	44	01	16
20	01	9	45	00	16
21	01	10	46	01	16
22	00	10	47	00	16
23	00	10	48	01	17
24	00	10	49	01	17
25	01	11	50	00	17

3 The basic decoding strategy

Consider the notation:

v the received sequence, which differs from the transmitted sequence due to errors

w the tentatively decoded sequence, a path in the code tree which is the decoder's tentative version of the transmitted sequence

$t = w \oplus v$ the test-error sequence, which has ones in the positions where w and v differ

t_b the sequence consisting of the last b branches of the sequence t.

Our basic decoding strategy is then as follows. We always seek a code path w which is at minimum distance $|t|$ from the received sequence v. In other words, a w is accepted to be the decoded sequence if and only if for all other paths w' in the corresponding truncated tree, w has minimum test-error weight. That is

$$|t| = |w \oplus v| \leqslant |w' \oplus v| = |t'|$$

We define the basic branch operation (b.b.o.) to be the decoding action of a single branch forward extension which selects the latest segment w_1 of w. Whenever a decoded path w is accepted as being the minimum distance path, the decoder shifts out the earliest segment of w, which is assumed to be a correct representation of the corresponding segment of the transmitted sequence, and shifts in the newly received segment v_1 of v. The b.b.o. then selects w_1 to be the segment closest in distance to v_1.

For the half-rate code, the b.b.o. results in a w_1 that always has a test-error weight $|t_1| = |w_1 \oplus v_1| \leqslant 1$. Thus $|t_1|$ is either 0 or 1. If we assume that the new segment w_1 results from the extension of a path that has minimum test-error weight, the following are implied. Firstly, if $|t_1| = 0$, the new path is guaranteed to have minimum test-error weight, and the decoder returns to the b.b.o. Alternatively, if $|t_1| = 1$, it is possible that there exists some other path w' with smaller test error weight $|t'| = |w' \oplus v| < |t|$, and if so $|t'_1| = 0$ and $|t'| = |t| - 1$. Proof of these assertions is given in Appendix 11.1.

Thus whenever the b.b.o. results in a $|t_1| = 1$ the decoder either automatically utilises the direct mapping scheme to eliminate the need to search for w', or else determines whether or not a back up search for w' is needed, and if so, how far to back up and how to conduct the search.

4 Permissible path decoding

Let us assume that the decoder needs to search the b-unit which spans the last b segments of the code tree, for a w' with smaller test-error weight. Following sequential decoding practice, this would require a step-by-step back-up, with the basic branch-by-branch encoding and examining method being used to calculate test-error weights. This is obviously a very lengthy process. We now introduce a systematic procedure for searching the b-unit, which requires considerably less effort than the method outlined above.

The procedure is based on property (b) of Section 2. This states that w' can be directly derived by the modulo-2 operation $w' = w \oplus x$, where x is a truncated path in the lower-half initial code tree. In addition

$$t' = w' \oplus v = w \oplus x \oplus v = t \oplus x$$

and so if w and w' are in opposite halves of a k-unit we can derive the test-error weight of w' by direct modulo-2 addition of t and the k-segment path x. This is still a cumbersome process, however, if all $2^k - 1$ truncated paths with length $k \leqslant b$ in the lower-half initial b-unit have to be used to search for w'. We now introduce several conditions which the x must satisfy because of the code structure. This serves to reduce the x required to search the b-unit to a very small number in most cases of interest. The reduced set of paths needed to search the b-unit are called permissible paths, and denoted by P.

The conditions are as follows:

(a) $|P|$ must be odd. Consider the following two cases:
- (i) If $|t|$ is odd, $|t'| = |t| - 1$ implies $|t'|$ is even. If $|P|$ is even, then $|t'| = |t \oplus P|$ is odd.
- (ii) If $|t|$ is even, $|t'| = |t| - 1$ implies $|t'|$ is odd. If $|P|$ is even, then $|t'| = |t \oplus P|$ is even.

In both cases $|t'| = |t| - 1$ is contradicted when $|P|$ is even, and therefore $|P|$ is odd.

(b) $|P_1| = 1$, as $|P_1| = |t_1 \oplus t'_1| = |t_1| = 1$.

(c) $|P| \leqslant 2|t| - 1$. Now $|t'| = |t \oplus P| \geqslant |P| - |t|$. If $|P| \geqslant 2|t|$ this implies $|t'| \geqslant |t|$, which is a contradiction to $|t'| = |t| - 1$.

We may further restrict the number of permissible paths by imposing a rule on the b.b.o. Because of the complement property, a rate one-half code will always have a $|t_1| \leqslant 1$. That is, the last segment of t is either 00, 01, or 10. For convenience, the quaternary digits 0, 1, 2, 3 are used to represent branches in this paper from now on. Therefore, t_1 is given by the quaternary digits 0, 1 and 2, respectively.

When $|t_1| = 1$ it does not matter (in terms of distance) whether the path giving $t_1 = 1$ or $t_1 = 2$ has been chosen. Let us then impose the condition that t_1 must either by 0 or 1, and eliminate the possibility of $t_1 = 2$. We may then further restrict the number of P, as the following conditions now also have to be satisfied.

(d) $P_1 = 1$, as $P_1 = t_1 \oplus t'_1 = t_1 = 1$.

(e) If P is longer than two segments, $P_2 = t_2 \oplus t'_2 = 01$.

This assertion requires further explanation. Under the modified b.b.o, the only possibilities for the last two segments of t are $t_2 = 00$, 11, or 01. We consider each possibility in turn:

(i) If $t_2 = 00$, the path w must still be at minimum distance from v since it is the b.b.o. extension from the decoded path having minimum test-error weight. Therefore no search is required.

(ii) If $t_2 = 11$, there exists the possibility of a w' with smaller test-error weight. An examination of the lower-half initial code tree (Fig. 3) shows that $P = 31$ satisfies the conditions for a two-segment permissible path, these conditions being $|P| = 3 =$ odd, $P_1 = 1$, and $|P| = 3 \leqslant 2|t_2| - 1$. Also there exists a $t'_2 = t_2 \oplus P = 11 \oplus 31 = 20$, such that $|t'_2| = |t_2| - 1$, and hence $|t'| = |t| - 1$. We thus impose the condition that for any tentatively decoded sequence w which has a $|t|$ with $t_2 = 11$, we will directly replace w_2 by $w'_2 = w_2 \oplus P$, and t_2 by $t'_2 = t_2 \oplus P$, where $P = 31$, and then return to the b.b.o.

(iii) if $t_2 = 01$, then $t'_2 = 00$ and $P_2 = t_2 \oplus t'_2 = 01$. Assume $|t'| = |t| - 1$ and $|t'_2| = 1$. Then as $|t'_2| = |t_2| = 1$ we have $[|t'| - |t'_2|] = [|t| - |t_2|] - 1$. Also as $|t'_1| = |t_1| - 1 = 0$ we have $|t'| - |t'_1| = |t| - |t_1|$. Thus t' should be the test-error sequence resulting from a b.b.o. extension of a path with minimum test-error weight, rather than t. Hence $|t'_2| = 0$ and therefore $P_2 = 01$ for all P longer than two branches.

Fig. 3 shows the first six segments of the lower-half initial code tree. Each segment is represented as a quaternary digit, and the number in the upper right-hand corner gives the weight of the code path up to that segment. A number in the lower right-hand corner indicates a permissible path, and gives the sequential order i of the permissible path $P_{(i)}$. It can be seen that there are only three permissible paths which satisfy the conditions on P. These are $P_{(1)} = 31$, $P_{(2)} = 32201$ and $P_{(3)} = 310101$. It is therefore possible to search the entire 6-unit without back-up, by making only three test-error weight comparisons based on $|t'| = |t \oplus P|$. In the next Section we eliminate the need for even this small number of comparisons.

5 Direct-mapping decoding

In this Section we introduce a direct-mapping scheme to eliminate all short back-up searches. In the Section it was shown previous that if the last two segments of t are $t_2 = 11$, we can always find a path with smaller test-error weight, $|t'| = |t| - 1 < |t|$, by directly changing w to $w' = w \oplus P_{(1)}$. The direct mapping scheme is an extension of this. In the scheme a set of test error patterns and corresponding permissible paths are stored, and utilised to directly change w to $w' = w \oplus P_{(i)}$.

To specify which test-error patterns do not have minimum weight, and should therefore be replaced by some t' during the decoding process, we need to build up a minimum test-error pattern tree. The tree is shown in Fig. 4 and starts with the b.b.o. from the very beginning. At each node in the tree the length of the test-error pattern increases by one segment. Also, we know that there are only two possibilities for t_1 at each b.b.o. extension, and so two branches stem from each node in the tree.

Starting from the first node, there are only two possible one-segment test-error sequences, 0 and 1. After the next b.b.o. extension there are four possible test-error sequences, 00, 01, 10 and 11. However, $t_2 = 11$ is not a minimum test-error pattern because there is a $t'_2 = t_2 \oplus P_{(1)} = 20$ with smaller weight. We therefore replace $t_2 = 11$ by $t'_2 = 20$ in the tree and assume that whenever a $t_2 = 11$ is encountered, the decoder directly maps t to $t' = t \oplus P_{(1)}$, and w is mapped to $w' = w \oplus P_{(1)}$. We continue building up the tree in a similar manner, such that each entry is guaranteed to be a minimum test-error pattern. In this way, we can build up a set of test-error patterns t_b and corresponding permissible paths $P_{(i)}$, for which $|t'| = |t \oplus P_{(i)}| = |t| - 1 < |t|$. Note that the test-error patterns in the upper half of the tree are the same as those in the lower half, preceded by one or more zeros. The search for the t_b can therefore be confined to the lower-half tree only.

Fig. 4 shows the first five segments of the minimum test-error pattern tree. The underlined sequences show where a t_b has been mapped to $t'_b = t_b \oplus P_{(i)}$, and the value of i is given in the lower right corner of that entry. The weight of each minimum test error pattern t is given in the upper right corner of each entry. Table 2 shows all the t_b for $b \leqslant 10$ segments, together with their corresponding $t'_b = t_b \oplus P_{(i)}$. The $P_{(i)}$ used are shown in Table 3.

A direct mapping decoder operating on this principle would therefore store the t_b and corresponding $P_{(i)}$ in memory. Decoding proceeds by using the b.b.o., and whenever the tentatively decoded sequence w has a t whose last b segments exactly match a pattern t_b stored in memory, we directly map t to $t' = t \oplus P_{(i)}$ and w to $w' = w \oplus P_{(i)}$. No searching for w' is therefore necessary. If t is such that its tail sequence does not match any stored t_b, then either t has minimum test-error weight, in which case the decoder can return to the b.b.o., or the required t_b and $P_{(i)}$ are ones which have not been stored. This latter case is dealt with in more detail later.

Table 2
TOTAL t_b AND t'_b FOR FIRST TEN SEGMENTS OF MINIMUM TEST-ERROR PATTERN TREE

b	t_b											i of $P_{(i)}$	$t'_b = t_b \oplus P_{(i)}$										
	11	10	9	8	7	6	5	4	3	2	1		12	11	10	9	8	7	6	5	4	3	2	1
2										1	1	1											2	0
5							1	0	2	0	1	2								2	2	0	0	0
5							2	0	2	0	1	2								1	2	0	0	0
5							1	0	1	0	1	3							3	0	0	0	0	0
6						1	0	0	1	0	1	3							2	1	0	0	0	0
6						2	0	0	1	0	1	3							1	1	0	0	0	0
8				1	0	0	2	0	1	0	1	4					2	2	1	0	0	0	0	0
8				2	0	0	2	0	1	0	1	4					1	2	1	0	0	0	0	0
8				1	2	0	0	0	1	0	1	4					2	0	1	2	0	0	0	0
8				2	2	0	0	0	1	0	1	4					1	0	1	2	0	0	0	0
8				1	0	2	0	0	2	0	1	5					2	2	0	0	2	0	0	0
8				2	0	2	0	0	2	0	1	5					1	2	0	0	2	0	0	0
8				1	2	0	0	0	2	0	1	5					2	0	2	0	2	0	0	0
8				2	2	0	0	0	2	0	1	5					1	0	2	0	2	0	0	0
9			1	0	0	1	2	0	0	0	1	6				2	1	0	0	1	0	0	0	0
9			1	0	1	0	0	2	0	0	1	7			3	0	0	0	0	2	0	0	0	0
9			1	0	0	2	0	1	0	0	1	8			3	0	0	2	0	0	0	0	0	0
9			1	0	2	0	0	1	0	0	1	8			3	0	0	0	2	0	0	0	0	0
10		1	0	0	1	0	0	2	0	0	1	7			2	1	0	0	0	2	0	0	0	0
10		2	0	0	1	0	0	2	0	0	1	7			1	1	0	0	0	2	0	0	0	0
10		1	1	0	0	0	0	2	0	0	1	7			2	0	0	1	0	2	0	0	0	0
10		1	0	0	0	2	0	1	0	0	1	8			2	1	0	2	0	0	0	0	0	0
10		2	0	0	0	2	0	1	0	0	1	8			1	1	0	2	0	0	0	0	0	0
10		1	1	0	0	0	0	1	0	0	1	8			2	0	0	2	2	0	0	0	0	0
10		2	0	0	2	0	0	1	0	0	1	8			1	1	0	0	2	0	0	0	0	0
10		3	0	0	0	0	0	2	0	0	1	7				1	0	1	0	2	0	0	0	0
10		3	0	0	0	0	0	1	0	0	1	8				1	0	2	2	0	0	0	0	0
10		2	0	1	0	1	0	0	2	0	1	9			1	2	0	1	0	2	0	0	0	0
10		2	3	0	0	0	0	0	1	0	1	10		3	3	0	0	0	0	0	0	0	0	0
10		2	3	0	0	0	0	0	2	0	1	11	3	2	0	0	0	0	0	0	2	0	0	0

Table 3
REQUIRED P_i AND EXPONENTIAL GROWTH OF THEIR APPLICATION

$P_{(i)}$	$P_{(i)}$												$\lvert P_{(i)}\rvert$	$A[P_{(i)}]$									
	b													b									
	12	11	10	9	8	7	6	5	4	3	2	1		10	9	8	7	6	5	4	3	2	1
$P_{(1)}$											3	1	3	83	31	19	11	3	2	2	–	1	–
$P_{(2)}$								3	2	2	0	1	5	8	4	2	1	–	2	–	–	–	–
$P_{(3)}$							3	1	0	1	0	1	5	9	5	4	1	2	1	–	–	–	–
$P_{(4)}$						3	2	1	2	0	1	0	1	7	3	–	4	—	–	–	–	–	–	–
$P_{(5)}$						3	2	2	0	2	2	0	1	7	2	–	4	–	–	–	–	–	–	–
$P_{(6)}$					3	1	0	1	3	0	0	0	1	7	–	1	–,	–	–	–	–	–	–	–
$P_{(7)}$				3	1	0	1	0	2	2	0	0	1	7	3	1	–	–	–	–	–	–	–	–
$P_{(8)}$				3	1	0	2	2	0	1	0	0	1	7	4	3	–	–	–	–	–	–	–	–
$P_{(9)}$				3	2	1	1	1	2	0	2	0	1	9	1	–	–	–	–	–	–	–	–	–
$P_{(10)}$			3	1	3	0	0	0	0	0	2	0	1	7	1	–	–	–	–	–	–	–	–	–
$P_{(11)}$		3	2	2	3	0	0	0	0	2	1	0	1	9	1	–	–	–	–	–	–	–	–	–

$A[P_{(i)}]$ is the number of applications of $P_{(i)}$ used in developing the bth segment of the minimum test-error tree

Table 4
COMPARISON OF GROWTH RATES BETWEEN $d(k)$ AND $\lvert t(k)_{max}\rvert$

k	$d(k)_{ave}$	$d(k)_{min}$	$\lvert t(k)_{max}\rvert$
1	2	2	1
2	3	3	1
3	4	3	2
4	5	4	2
5	6	4	3
6	7	5	3
7	8	5	3
8	9	5	4
9	10	6	4
10	11	6	4

Table 5
MAXIMUM BACK-UP DISTANCE b_t FOR DIFFERENT VALUES OF $\lvert t\rvert$

$\lvert t\rvert$	$d(b_t)$	b_t
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

Table 6
THRESHOLD CONDITIONS $T^*(b)$ ON BACK-UP DISTANCE $b_t^* = b$

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

An example of direct mapping decoding is shown in Fig. 5. The received sequence v has been obtained from an all-zero transmitted sequence, and contains four errors. The decoder starts by using the b.b.o., and whenever the tail of the test-error sequence matches one of the patterns in Table 2 a mapping operation is performed. The lines show the path taken by the decoder through the code tree. Each segment of w is given above the path, and the corresponding segment of t appears below the path. It can be seen that to correctly decode the 12-segment received sequence, it is necessary only to perform 12 b.b.o.s and four mapping operations. This is considerably less than the decoding effort required by other sequential decoding schemes to correct the same pattern of errors.

The range over which direct mapping can be operated in a practical decoder depends on the storage requirements of the t_b and $P_{(i)}$. This range, in segments, will be denoted b_m. For example, Table 2 shows that 30 t_b and 11 $P_{(i)}$ are needed to operate direct mapping over $b_m = 10$ segments.

An idea of the growth rate of the number of $P_{(i)}$ required can be obtained by considering condition (c) in Section 4, which states that $|P| < 2|t|$. As P is a path in the lower-half initial code tree, we can compare the weight of P to that of other paths in the lower-half initial code tree, as follows. Let $d(k)_{min}$ be the minimum distance of the code, that is, the weight of the minimum-weight path in the lower-half initial k-unit, and $d(k)_{ave}$ be the average weight of paths in the lower-half initial k-unit. If $|t(k)|_{max}$ is the maximum weight of k-segment test-error patterns, then in the worst case $|P| < 2|t(k)|_{max}$. Table 4 compares $|t(k)|_{max}$, $d(k)_{min}$, and $d(k)_{ave}$ for $k \leqslant 10$. From this Table it can be seen that the growth rate of $|t(k)|_{max}$ is not only less than the growth rate of $d(k)_{ave}$, but also less than that of $d(k)_{min}$. This indicates that the growth rate of $|P|$ with increasing k is very slow, and also that the number of required P increases slowly with k, because $|P|$ must be much less than $d(k)_{ave}$ and only just larger than $d(k)_{min}$.

The slow growth in the number of new P that must be stored as b_m is extended also limits the growth rate of the number of new t_b that must be stored. Thus, although the number of possible mappings increases exponentially with k, most of these are performed with permissible paths of length less than k. It is therefore the number of applications of existing t_b (and correspondingly $P_{(i)}$) that grows exponentially with k, rather than the number of new t_b. This is shown in Table 3. For example, in developing the minimum test-error pattern tree from 1 segment to 2 segments deep, $P_{(1)}$ is used once. However, when extending the tree from nine segments to 10, $P_{(1)}$ is used 83 times.

Note that direct mapping can be used by itself as a sub-optimum minimum-distance decoding procedure. In this case, if $t_1 = 1$ and the tail sequence of t does not match any pattern in store, we must consider the possibility that w' (which has $|t'| = |t| - 1$) diverges from w at greater than b_m nodes back. The earliest segment of w (which may or may not be in error) is then shifted out of the decoder, which reverts to the b.b.o. and direct mapping. A sub-optimum direct-mapping decoder of this type therefore does no searching at all, but will sometimes accept errors and then recover to the correct path in time.

The algorithm proposed in this paper, however, uses direct mapping to eliminate all short back-up searches, up to a maximum range of b_m nodes. If $t_1 = 1$ and no direct mapping is possible, then either w has minimum test-error weight or else w' diverges from w at greater than b_m nodes back. The next Section deals with the method for determining whether or not w has minimum test-error weight, and if not, how to determine the nodes at which it is possible for a w' with $|t'| = |t| - 1 < |t|$ to diverge from w.

6 Determination of the back-up distance

In this Section we examine the course of action to be taken if $|t_1| = 1$ and direct mapping is not possible. Some of the results utilised are based on our previous work,[4] and are therefore only summarised here.

The first question to be answered is whether or not a back-up search is necessary. That is, whether or not there is a possible w' with $|t'| = |t| - 1$ that diverges from w at greater than b_m nodes back. If the answer to this is no, then w is at minimum distance from the received sequence v, and the decoder returns to the b.b.o.

To answer this question we utilise an upper bound b_t on the back-up distance. The bound states that when w (with $|t_1| = 1$) is the b.b.o. extension of a path having minimum test-error weight, and if there exists a w' with $|t'| < |t|$, then w' diverges from w at most b_t nodes back, where b_t is the minimum value of i such that

$$d(i) = 2|t| - 1 \tag{1}$$

Thus if $b_t \leqslant b_m$ no search is necessary. The bound is proved in Appendix 11.3, and Table 5 shows b_t for various $|t|$. If $b_t > b_m$ then it is still possible that a search for w' will be needed. In this case the first thing that must be done is to obtain an improved (lower) value of b_t, which is denoted b_o. The process of reducing the maximum required back-up distance from b_t to b_o is referred to as the back-up reduction operation (b.r.o.), and is explained as follows.

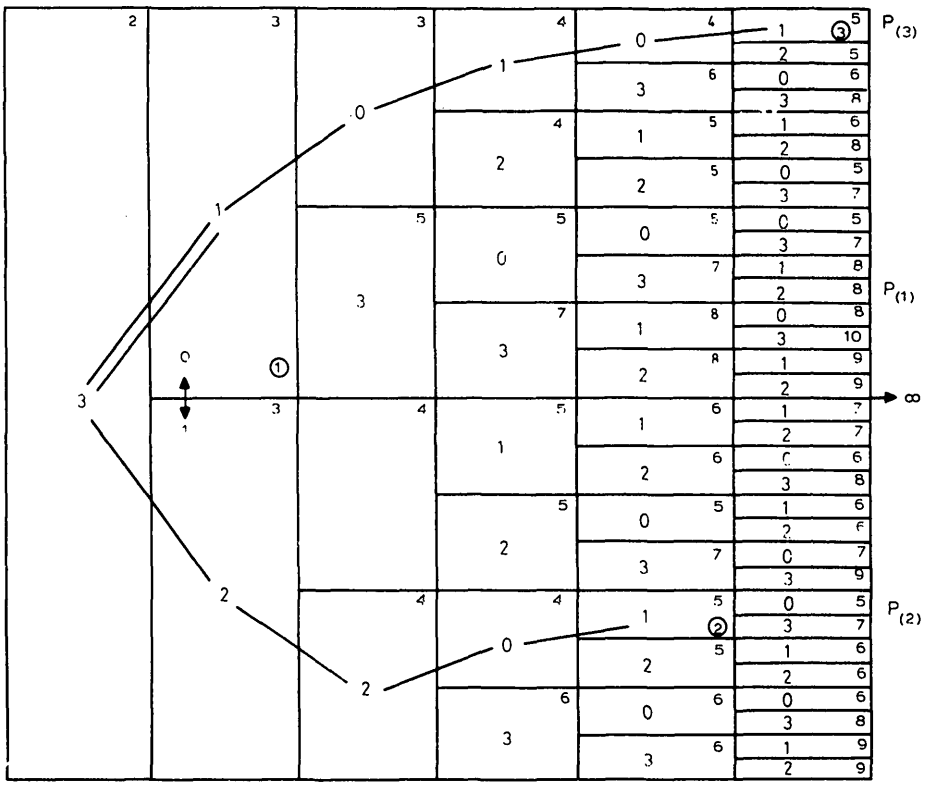

Fig. 3
Selection of permissible paths from the lower-half initial code tree

The maximum back-up distance b_t is based on the test-error weight over the entire sequence t. If some of the test-error weight is due to errors which have occurred earlier than b_t nodes back, that is $|t| > |t_{b_t}|$, then the value obtained for b_t by using eqn. 1 will be too high. Hence, as we are only interested in the test-error weight over the last b_t segments when searching for a w' with smaller test-error weight, we can replace $|t|$ by $|t_{b_t}|$ in eqn. 1, and determine a new $b_1 < b_t$, which corresponds to $|t_{b_t}|$ in the same way that b_t corresponds to $|t|$. Similarly, if $|t_{b_1}| < |t_{b_t}|$, we can again use eqn. 1 to determine a new $b_2 < b_1$, and so on. At some point the process stops with a minimum value b_o. If $b_o \leqslant b_m$ then we know that w has minimum test-error weight, and so no back-up search is needed.

An example of the b.r.o. is as follows. Suppose $|t| = 8$; from Table 5 this gives $b_t = 40$. If there is a test-error weight of 2 in front of the last 40 segments of $|t|$ then $|t_{40}| = 6$, and hence $b_1 = 25$. If $|t_{25}| = 4$, then $b_2 = 11$. If $|t_{11}| = 4$ then no further reduction is possible and we have $b_o = 11$.

We now consider the situation in which $b_o > b_m$. In this case we examine each node between $b_m + 1$ and b_o by means of a simple threshold value, to see whether or not it is possible that w' diverges from w at that node. The end result is a small set of nodes, whose back-up distances are denoted b_t^* at which w' may have diverged from w.

The b_t^*, are found as follows. Property (e) of Section 2 enables us to write $|t'| \geqslant d(k) - |t|$ (see Appendix 11.2). For a given node at back-up distance b we may then write $|t'_b| \geqslant d(b) - |t_b|$. Because we are searching for a $|t'_b| = |t_b| - 1$, the necessary condition for the existence of such a $|t'_b|$ is $|t_b| \geqslant [d(b) + \delta]/2$, where δ is 1 if $d(b)$ is odd or 2 if $d(b)$ is even. This gives us a lower found which $|t_b|$ must satisfy for it to be possible that w' diverges from w at b nodes back.

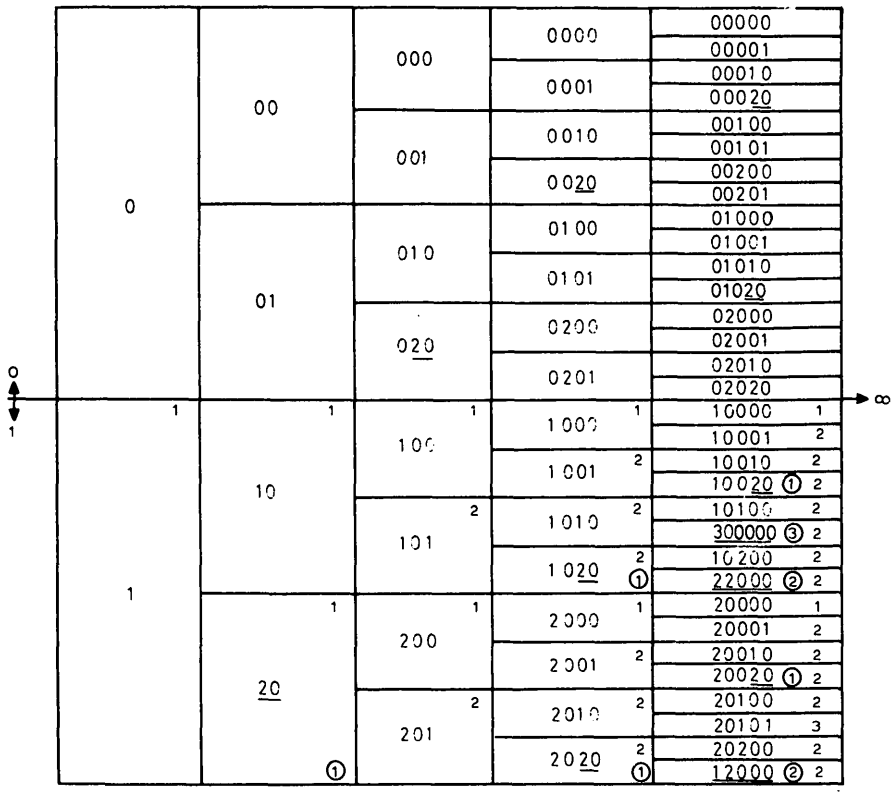

Fig. 4
The minimum test-error pattern tree up to 5 segments

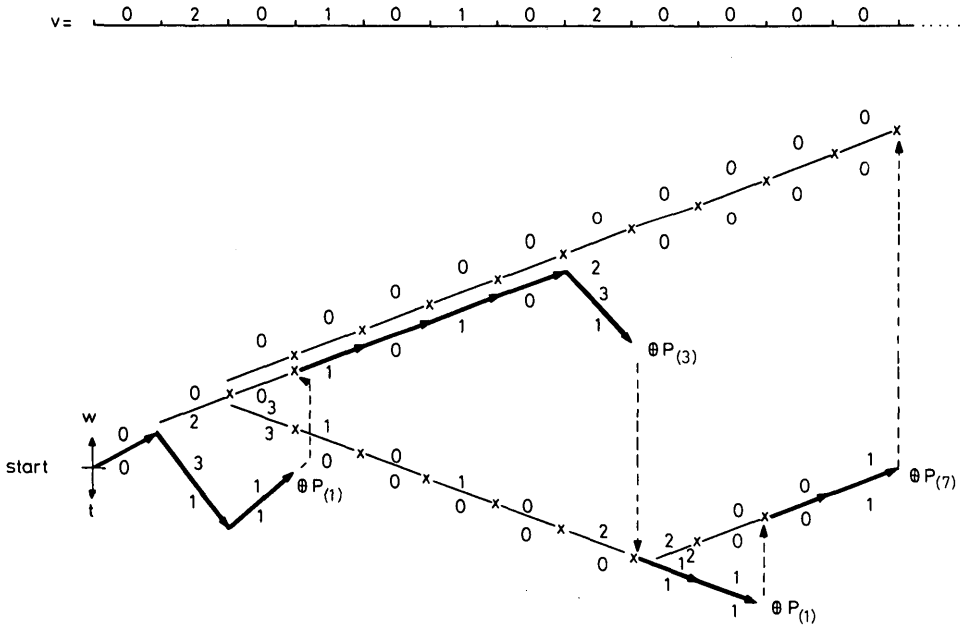

Fig. 5
Direct mapping search of a given received sequence
——→ obtained from b.b.o.
-----→ a mapping operation
x——x obtained by direct mapping with $P_{(i)}$

The above bound can be tightened slightly by noting that $|t_1| = 1$, and $|t'_1| = 0$. By using the lower bound on $|t_b|$ in conjunction with the upper bound on back-up distance (eqn. 1), we can establish a new threshold condition $T^*(b)$ which $|t_b|$ must satisfy in order for it to be possible that w' diverges from w at b nodes back. This threshold condition is as follows. It is only possible for w' to diverge from w at $b^*_t = b$ nodes back if $|t_b| \geq T^*(b) = [d(j) + 1]/2$, where j is the minimum value such that $d(j) \geq d(b)$ and $d(j)$ is odd. Table 6 shows values of b and $T^*(b)$.

An example of applying Table 6 is as follows. Suppose the b.r.o. gives $b_o = 11$, and $t_{11} = 10010000101$. This gives $|t_b| < T^*(b)$ for $b \leq 10$, and $|t_b| = T^*(b)$ for $b = 11$. This indicates that the only possible back-up distance is $b^*_t = 11$.

The method of specifying the b^*_t given in this Section considerably cuts down the amount of tree searching needed to find w'. In the next Section we outline an efficient method of searching for w' with the aid of direct mapping.

7 Utilising direct mapping in the tree search

Having established the values of b^*_t at which w' may have diverged from w, we instigate a search of the $b^*_t - 1$ segment truncated tree stemming from the complement branch of w, for each value of b^*_t, starting with the smallest value greater than b_m.

Each truncated tree is searched in the following manner. First of all the current test-error sequence t is put into storage for later use. At the node b^*_t we force the decoder to take the complement branch to w, and at the same time start a new test-error sequence t^*, which has $|t^*| = 0$ at the node b_t. The search of the truncated tree continues by using the b.b.o., direct mapping, and the back-up operation, as follows.

Assume that the decoder has reached a point c segments from the node b^*_t, and that the test-error weight $|t^*|$ has just become equal to $|t_{b^*_t}|$. If $c \leq b_m + 1$, and the direct-mapping decoder cannot perform a mapping, then the search of the truncated tree is abandoned because no path in it can have $|t'| = |t| - 1$. If $c > b_m + 1$ it is possible that the path w' diverges from the present path being following somewhere between $b_m + 1$ and $c - 1$ nodes back. In this case we can determine the possible nodes at which w' might have diverged by using the $T^*(b)$ threshold conditions on t^*. If the threshold conditions state that the smallest back-up distance is greater than or equal to c, then the search of the truncated tree is abandoned. Otherwise, a back-up is instigated, and we carry out the search using the b.b.o., direct mapping, and the back-up operation. If each of the possible nodes between $b_m + 1$ and $c - 1$ have been searched, and no path of length $c = b^*_t$ with test-error weight $|t^*_c| = |t_{b^*_t}| - 1$ can be found, there is no w' with $|t'| = |t| - 1$ in the truncated tree corresponding to the present value of b^*_t. In this case the back-up distance is increased to the next value of b^*_t, and the search procedure is repeated.

8 The final algorithm

The decoding algorithm can be summarised as follows:

(*a*) Decoding proceeds by means of the b.b.o. and direct mapping, which guarantees that the path being followed is at minimum distance from the received sequence. Whenever the b.b.o. extension results in a $|t_1| = 0$, the decoder returns to the b.b.o.

(*b*) If $|t_1| = 1$ and the direct mapping decoder indicates that no mapping has taken place, proceed to (*c*). If a mapping has taken place, return to the b.b.o.

(*c*) Determine the maximum back-up distance b_o by means of the upper bound and the b.r.o. If $b_o \leq b_m$ no search is needed and the decoder returns to the b.b.o. If $b_o > b_m$ proceed to (*d*).

(*d*) Determine the values of b^*_t for $b_m < b^*_t \leq b_o$.

(*e*) For each value of b^*_t utilise the direct mapping decoder to search for the path with minimum test-error weight, starting with the lowest value of b^*_t.

(*f*) If no path of length b^*_t with $|t^*_{b^*_t}| = |t_{b^*_t}| - 1$ can be found in the the truncated tree stemming from the complement branch of node b^*_t, repeat (*e*) for the next largest value of b^*_t.

(*g*) If the required path is found, replace $t_{b^*_t}$ with $t^*_{b^*_t}$ and return to the b.b.o. and direct mapping.

(*h*) If we run out of search time, then force the decoder to accept the earliest segment of w, and return to the b.b.o. and direct mapping. Thus an error may be accepted, but the decoder will recover to the correct path in time.

9 Conclusions

In this paper we have presented a new minimum-distance decoding algorithm for convolutional codes. Initial simulation tests have confirmed that the amount of decoding effort is considerably less than other convolutional decoding schemes. The advantages of the proposed algorithm are best seen in relation to sequential decoding. Firstly, from the performance point of view: since our algorithm is minimum-distance decoding, it is clear that for any received sequence v, the test-error weight obtained by the decoding algorithm will be always less than or equal to the test-error weight obtained from sequential decoding. Therefore, the probability of decoding error will be always less than or equal to that of sequential decoding. Secondly, from the decoding operations point of view: it is well known that the probability of buffer overflow ultimately determines the performance of a sequential decoder. By utilising direct mapping to eliminate all short back-up searches, by using minimum-distance decoding to catch possible decoding errors in the earliest possible segment, and by using the threshold conditions on back-up distance to eliminate unnecessary back-up searches, it can be seen that the proposed algorithm will require much less decoding effort than other sequential decoding schemes. Therefore when the size of buffer is fixed, the proposed algorithm will always give a lower probability of buffer overflow, and hence a better performance. Future work will be aimed towards analytically establishing the distribution of the number of computations for the algorithm and in obtaining fuller simulated performance results.

10 References

1 WOZENCRAFT, J.M., and REIFFEN, B.: 'Sequential decoding' (Wiley, 1961)

2 FANO, R.M.: 'A heuristic discussion on probabilistic decoding', *IEEE Trans.*, 1963, **IT-9**, pp. 64–67

3 JELINEK, F.: 'A fast sequential decoding algorithm using a stack', *IBM J. Res. & Dev.*, 1969, **13**, pp. 675–685

4 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

11 Appendix

11.1 Proof of $|t'| = |t| - 1$

Since w is an extension by b.b.o. of an accepted path which has minimum test-error weight, we have $|t| - |t_1| \leq |t'| - |t'_1|$. If $|t_1| = 1$ then $|t'| < |t|$ if and only if $|t| - |t_1| = |t'| - |t'_1|$, and $|t'_1| = 0$. Hence $|t'| = |t| - 1$.

11.2 Proof of $|t'| \geq d(k) - |t|$

From property (*e*) of Section 2, $|w \oplus w'| \geq d(k)$. As $t = w \oplus v$, then $|w \oplus v \oplus v \oplus w'| \geq d(k)$. That is, $|t \oplus t'| \geq d(k)$. Hence $|t| + |t'| \geq d(k)$, and $|t'| \geq d(k) - |t|$.

11.3 Proof of back-up distance bound

Assume there is a path w' with test-error weight $|t'| < |t|$ which diverges from w at a back-up distance of b'_t segments. Let $b'_t > b_t$. This implies $d(b'_t) \geq d(b_t)$, from the distance property of the code.

(*a*) If $d(b'_t) > d(b_t)$.

From Appendix 11.2 $|t'| \geq d(b'_t) - |t|$, that is, $|t'| \geq [d(b_t) + 1] - |t|$. Also, from Appendix 11.2 $|t'| \geq d(k) - |t|$, if $|t'| = |t| - 1$ then $d(b_t) \leq 2|t| - 1$. Hence $|t'| \geq [(2|t| - 1) + 1] - |t| \geq |t|$, which is a contradiction to $|t'| < |t|$.

(*b*) If $d(b'_t) = d(b_t)$.

Now $|t'| = [|t'| - |t'_1|] + |t'_1|$. Substituting for $(|t'| - |t'_1|)$ gives $|t'| \geq [d(b_t) - (|t| - |t_1|)] + |t'_1| \geq [(2|t| - 1) - |t| + 1] + 0 \geq |t|$, which is a contradiction to $|t'| < |t|$.

Thus $b'_t \not> b_t$ and b_t is the *minimum* integer which satisfies $d(b_t) \leq 2|t| - 1$.