

# Simple method for generating distance spectrum and multiplicities of convolutional codes

Li Ping

Indexing terms: Convolution codes, Error correction codes, Difference equations, Algorithms

**Abstract:** The author presents a simple algorithm for generating the distance spectrum and multiplicities of convolutional codes. It is different from the existing methods in that no topological searching or matrix inversion is involved. Instead, the method presented consists of the recursion of a set of difference equations with a properly selected operational sequence. The computational cost of the new algorithm can be easily predicted and is modest.

## 1 Introduction

The existing algorithms for generating the distance spectra and multiplicities of convolutional codes generally fall into two categories, based on trellis searching [1, 2] and matrix inversion, respectively [3, 4]. The computational cost of a searching algorithm depends on the code structure and is difficult to predict. It is a very skilful task to prevent excessive operations and to manage computer memory usage in implementing a searching algorithm [2]. The matrix approach involves the inversion of the state transition matrix that can be computationally costly. The recurrent technique recently introduced in [4] has greatly improved the efficiency of the matrix approach. However it still, at least in the direct form, requires a considerable amount of computer memory, especially when the state transition matrix is large.

In this paper we describe a very simple recursive algorithm formalised as a set of difference equations. We will show that the difference equations are solvable following a properly selected operational sequence. This is very concise for the purpose of programming. The computational cost of the new algorithm can be easily predicted and is less than the matrix method. In particular, the reduction of the computer memory usage is significant since no matrix manipulation is involved in the new method.

## 2 Introductory concepts and notations

For simplicity we consider only binary convolutional codes with at most one transition between any two

states. The results, however, can be easily extended to general linear trellis codes.

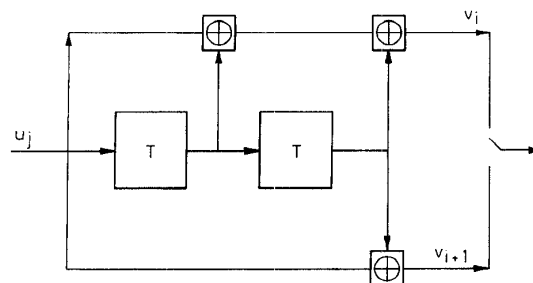


Fig. 1 1/2 convolutional encoder with memory order of 2  
⊕ inside a square represents modulo two addition

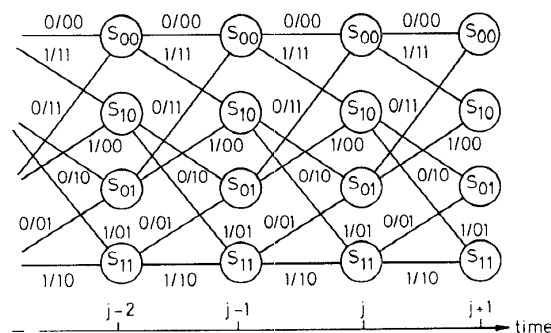


Fig. 2 Trellis representation of the code in Fig. 1  
Label for each branch is information sub-sequence/coded sub-sequence

Let the information sequence be  $\{u_j\}$ , convolutionally coded to  $\{v_j\}$ . In the following we refer to  $\{j\}$  as the time sequence. As a common treatment for linear codes [5], let the input be an all-zero sequence, i.e.  $u_j = 0$  for all  $j$ . Consequently  $v_i = 0$  for all  $i$  due to linearity. Adopt the trellis representation of the convolutional code, as shown in Figs. 1 and 2. Introduce the following notations:

$S_n$  = state (or node) where  $n$  is the binary expression for the state of the encoder's memory. In particular denote the all-zero state  $S_0 = S_{00\dots 0}$ .

$b_{mn}$  = transition (or branch) from  $S_m$  to  $S_n$ . For a rate  $b/a$  code a branch has an associated  $a$ -bit long coded sub-sequence from  $\{v_j\}$  and an associated  $b$ -bit long information sub-sequence from  $\{u_j\}$ .

$d_{mn}$  = the number of 1's in the coded sub-sequence associated with branch  $b_{mn}$ . For a rate  $b/a$  code  $0 \leq d_{mn} \leq a$ .

$e_{mn}$  = number of 1's in the information sub-sequence associated with branch  $b_{mn}$ .

© IEE, 1996

IEE Proceedings online no. 19960672

Paper first received 18th September 1995 and in revised form 8th May 1996

The author is with the Department of Electronic Engineering, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong

$p$  = path, which is a sequence of cascade branches. It will be represented by the sequence of the nodes incident to these branches, e.g.  $p = \{S_{n1}, S_{n2}, S_{n3}, \dots\}$ . In particular, we denote by  $p_0$  the correct path, i.e.  $p_0 = \{S_0, S_0, \dots, S_0\}$ .

SEP = simple erroneous path. It is a path that satisfies the following conditions:

- (a) its first node is  $S_0$
- (b) its second node is not  $S_0$
- (c) its last node may or may not be  $S_0$ , and
- (d) its remaining nodes are not  $S_0$ .

In short, a SEP is a path that deviates from  $p_0$  for the first time. As examples,  $\{S_0, S_2, S_1, S_0\}$  and  $\{S_0, S_2, S_1, S_2\}$  are both SEPs but  $\{S_0, S_2, S_1, S_0, S_2\}$  and  $\{S_0, S_0\}$  are not. A closed SEP is a SEP ending at  $S_0$ . It represents the merging of a SEP with  $p_0$  and could potentially cause a decoding error [5].

$w(p)$  = weight of a path  $p$ , defined by

$$w(p) = \sum_{\text{for all } b_{mn} \text{ belonging to } p} d_{mn} \quad (1)$$

In particular, the weight of a closed SEP is its Hamming distance from  $p_0$ , i.e. the total number of 1s in its associated coded sequence.

$a_k^{(n)}$  = total number of SEPs of weight  $k$  ending at node  $S_n$ . In particular,  $a_k^{(0)}$  is the total number of closed SEPs of weight  $k$ .

$$e(p) = \sum_{\text{for all } b_{mn} \text{ belonging to } p} e_{mn} \quad (2)$$

If the correct path is  $p_0$ , choosing a closed SEP  $p$  as the survivor in a Viterbi decoder would result in  $e(p)$  bits of errors, since each decoding output of 1 counts an error bit.

$c_k^{(n)}$  = total  $e(p)$  of all the SEPs of weight- $k$  ending at node  $S_n$ , i.e.

$$c_k^{(n)} = \sum_{\substack{p \text{ ending at } S^{(n)} \\ w(p)=k}} e(p) \quad (3)$$

In particular,  $c_k^{(0)}$  is the total number of 1s associated with all closed SEPs of weight  $k$ .

Notice that we have not specified the time instant above. We will assume that the trellis is time invariant so that these notations are independent of the time sequence  $\{j\}$ .

We will call  $\{a_k^{(0)}\}$  the *distance spectrum* and  $\{c_k^{(0)}\}$  the *multiplicities* of the code. The minimum  $k$  for which  $a_k^{(0)} \neq 0$  is called the *free distance* of the code. The distance spectrum and multiplicities determine the error correcting capability of a convolutional code [1]. For example, in an AGWN (additive Gaussian white noise) channel the bit error rate of a code is bounded by

$$P_b \leq \sum_k c_k^{(0)} \operatorname{erfc} \left( \sqrt{2k \frac{E_b}{N_0}} \right) \quad (4)$$

where  $E_b/N_0$  is the signal-to-noise ratio.

### 3 Algorithm

We emphasise again that  $a_k^{(n)}$  and  $c_k^{(n)}$  are constants with respect to the time sequence assuming a time invariant trellis. The main part of the algorithm is now given as a remark.

*Remark:* At any particular time instant  $j$

$$a_k^{(n)} = \sum_{m \neq 0, d_{mn} \leq k} a_{k-d_{mn}}^{(m)} + \delta_k^{(n)} \quad (5a)$$

$$c_k^{(n)} = \sum_{m \neq 0, d_{mn} \leq k} (c_{k-d_{mn}}^{(m)} + e_{mn} a_{k-d_{mn}}^{(m)}) + e_{0n} \delta_k^{(n)} \quad (5b)$$

with

$$\delta_k^{(n)} = \begin{cases} 1 & \text{if } d_{0n} = k \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

The proof of the remark is given in the Appendix. It appears that eqn. 5 can be solved by a recursion on  $k$ , but this is not entirely straightforward as  $d_{mn}$  can be zero. The problem can be overcome for a non-catastrophic code [6], which is always uniquely decodable. Call a state  $S_n$  a root if it has no input branch with  $d_{mn} = 0$ . As a special case, also call  $S_0$  a root. For a non-catastrophic code, every state  $S_n$  is either itself a root or backward linked by a unique path of weight zero to a unique root, referred as the root of  $S_n$ . Otherwise if a path of weight zero does not backward link to a root, it has to be infinitely long. This path cannot be distinguished from  $p_0$  in decoding and the code becomes catastrophic [6]. If there are more than one path of weight zero between any two states, these paths cannot be distinguished either. Excluding catastrophic codes, we can use an index  $t_n$  to indicate the weight between a node  $S_n$  and its root, i.e.

$$t_n = \text{number of branches along the weight-zero path from its root to } S_n \quad (7)$$

It is seen that for node  $S_n$ , eqn. 5 depends only on the solutions of  $\{S_m\}$  with lower  $k$  and with current  $k$  but lower  $t_m (< t_n)$ . Consequently it can be solved with double recursions. The outer recursion proceeds from  $k = 0$  to  $k = k_{max}$  ( $k_{max}$  is the maximum distance required). For each  $k$ , the inner recursion starts from nodes with  $t_n = 0$  and proceeds along zero-weighted paths to nodes with higher  $t_n$ . When  $k = 0$ , the process starts from all the roots, with  $a_0^{(n)} = \delta_0^{(n)}$  for eqn. 5a and  $c_0^{(n)} = e_{0n} \delta_0^{(n)}$  for eqn. 5b.

### 4 Computational cost

For a binary rate  $b/a$  convolutional code with memory order  $c$ ,  $2(a+1)$  vectors of dimension  $2^c$  are required to store  $a_i^{(n)}$  and  $c_i^{(n)}$  for  $i = k, k-1, \dots, k-a$ . This is because  $0 \leq d_{mn} \leq a$  in eqn. 5. We also need a  $2^b \times 2^c$  matrix for recording  $\{d_{mn}\}$  (there are  $2^b$  input branches to each node). We do not need to record  $\{e_{mn}\}$  since they can be directly determined from the state indices. For each recursion on  $k$ , eqn. 5 involves

$$(3 \times 2^b - 2) \times 2^c \text{ additions and } 2^b \times 2^c \text{ multiplications} \quad (8)$$

It can be shown that the above memory usage and operation numbers are less than those of the matrix method [4]. The reduction of the computer memory usage is significant as the matrix manipulation is completely avoided.

In particular, for a common  $1/a$  code the total memory usage is  $2 \times (a+2)$  vectors of dimension  $2^c$ . Eqn. 8 results in  $4 \times 2^c$  additions but  $2^c$  of them (for eqn. 5b) are not necessary when  $e_{mn} = 0$ . No multiplication is necessary since now  $e_{mn}$  is either 0 or 1. If we are interested in the parameters for up to  $k_{max}$  then the total amount of computation is  $3k_{max} 2^c$  additions. This is reasonable for  $c < 20$  with modest computing power.

Further savings of computational cost are possible with some extra programming effort. For example, the execution of eqn. 5 results in many zero values at the starting stage (see the example below) which can be skipped. Similar saving can be gained at the finishing stage by skipping calculations for unwanted values.

## 5 Example

As an example, we wish to find the free distance and the related multiplicity for the convolutional code in Figs. 1 and 2, with  $b = 1$ ,  $a = 2$  and  $c = 2$ . The roots are  $\{S_{00}, S_{01}, S_{11}\}$ .  $S_{10}$  is not a root since the branch between  $S_{01}$  and  $S_{10}$  is associated with a coded sequence of  $\{00\}$ . Thus,

$$t_{00} = t_{01} = t_{11} = 0 \quad t_{10} = 1 \quad (9)$$

also

$$\delta_k^{(n)} = 0 \text{ for all } n, k \text{ except } \delta_2^{(10)} = 1 \quad (10)$$

This is because besides  $S_{00}$  itself, only  $S_{10}$  is directly backward linked to  $S_{00}$ , with a coded weight of 2. Omitting zero terms, eqn. 5a and eqn. 5b can be expanded as

$$a_k^{(00)} = a_{k-2}^{(01)} \quad (11a)$$

$$c_k^{(01)} = c_{k-2}^{(01)} \quad (11b)$$

$$a_k^{(11)} = a_{k-1}^{(10)} + a_{k-1}^{(11)} \quad (11c)$$

$$c_k^{(10)} = c_{k-2}^{(00)} + c_{k-1}^{(01)} + \delta_k^{(10)} \quad (11d)$$

For each  $k$ , eqn. 11a-c have to be executed before eqn. 11d since  $S_{10}$  is not a root. For  $k = 0$  the initial values are all zero. Following eqn. 11a-d and noticing eqn. 10, we obtain the results in Table 1 for  $k$  up to 5. When  $k = 5$ ,  $a_k^{(00)} \neq 0$  for the first time. Therefore the free distance for this code is 5 and the multiplicity for all closed weight-5 SEP is  $c_k^{(00)} = 1$ .

Table 1: Results for the recursive eqn. 11

k	$a_k^{(00)}$	$a_k^{(01)}$	$a_k^{(11)}$	$a_k^{(10)}$	$c_k^{(00)}$	$c_k^{(01)}$	$c_k^{(11)}$	$c_k^{(10)}$
1	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	1
3	0	1	1	1	0	1	2	2
4	0	2	2	2	0	4	6	6
5	1	4	4	4	1	12	16	16

## 6 Conclusions

A simple algorithm has been presented for computing the distance spectra and the multiplicities of convolutional codes. The computational of the new algorithms is modest and is independent of the code structure, (besides the code rate and memory length). A main advantage of the new algorithm is its conciseness for programming, which is readily seen from eqn. 5. Further work is required to extend the results to nonlinear trellis codes [4].

## 7 Acknowledgments

The author wishes to thank J. Millott, G. Findlow, M. Faulkner and J. Santos for their help in this work. The permission of the Director of Telstra Research Laboratories to publish this paper is acknowledged.

## 8 References

- 1 BAHL, L.R., CULLUM, C.D., FRAZER, W.D., and JELINEK, F.: 'An efficient algorithm for computing the free distance', *IEEE Trans.*, 1972, **IT-18**, (6), pp. 437-439
- 2 CEDERVALL, M., and JOHANNESON, R.: 'A fast algorithm for computing distance spectrum of convolutional codes', *IEEE Trans.*, 1989, **IT-35**, (6), pp. 1146-1159
- 3 CONAN, J.: 'The weight spectra of some short low-rate convolutional codes', *IEEE Trans.*, 1984, **COM-32**, (9), pp. 1050-1053
- 4 TROFIMOV, A.N., and KUDRYASHOV, B.D.: 'Distance spectra and upper bounds on error probability for trellis codes', *IEEE Trans.*, 1995, **IT-41**, (2), pp. 561-572
- 5 VITERBI, A.J.: 'Convolutional codes and their performance in communication systems', *IEEE Trans.*, 1971, **COM-19**, (5), pp. 751-772
- 6 LIN, S., and COSTELLO D.J., Jr.: 'Error control coding: Fundamentals and applications' (Prentice-Hall, Englewood Cliffs, 1983)

## 9 Appendix

To prove eqn. 5a, consider a branch  $b_{mn}$  and a set of SEPs, (see Fig. 3)

$$\{p\} = \{p | w(p) = k, p \text{ ends with } b_{mn}\} \quad (12)$$

If  $m \neq 0$ , let  $\{p^*\}$  be obtained by deleting  $b_{mn}$  from every path in  $\{p\}$ . Obviously each element in  $\{p^*\}$  ends at  $S_m$  and  $w(p^*) = k - d_{mn}$ . The numbers of elements in  $\{p\}$  and  $\{p^*\}$  are the same and equal to  $a_k^{(m)}$ . Thus the summation term in eqn. 5a is the total number of weight- $k$  paths passing all  $S_m$  with  $m \neq 0$  and ending at  $S_n$ . Notice that  $m = 0$  is a special case since according to the definition  $b_{0n}$  must be the start of a new SEP. We include  $b_{0n}$  in counting  $a_k^{(m)}$  only when  $d_{0n} = k$ , in which case  $b_{0n}$  is itself a SEP. This contributes the delta term in eqn. 5a. Finally the condition  $d_{mn} \leq k$  ensures that the subscripts are not negative.

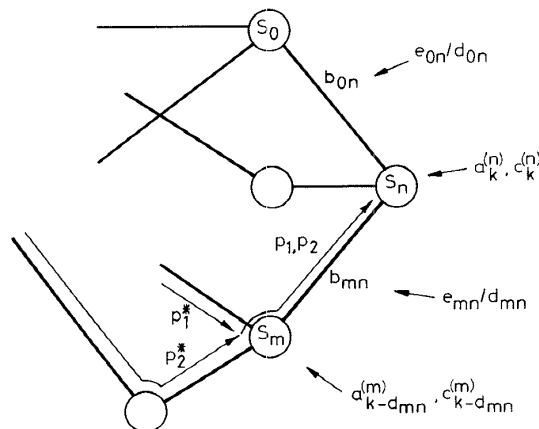


Fig. 3 Part of a trellis used in the proof of the remark Two paths,  $p_1$  and  $p_2$ , end with  $b_{mn}$

To prove eqn. 5b, notice that for  $m \neq 0$

$$\begin{aligned} \sum_p e(p) &= \sum_{p^*} (e(p^*) + e_{mn}) = c_{k-d_{mn}}^{(m)} + e_{mn} \sum_{p^*} 1 \\ &= c_{k-d_{mn}}^{(m)} + e_{mn} a_{k-d_{mn}}^{(m)} \end{aligned} \quad (13)$$

Summing both sides of eqn. 13 over all  $m \neq 0$  results in the summation term in eqn. 5b. For  $m = 0$ ,  $b_{0n}$  contributes the delta term in eqn. 5b when  $d_{0n} = k$ .