DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

SECTION AUTOMATIQUE

Rapport Technique EP74-R-47

Classification: Library of Congress no ....

MULTIPLE-PATH STACK ALGORITHMS FOR DECODING CONVOLUTIONAL CODES

par

David Haccoun
Professeur adjoint

Octobre 1974

Ecole Polytechnique de Montréal

Campus de l'Université
de Montréal
Case postale 6079
Succursale 'A'
Montréal, Québec
H3C 3A7

15 NOV 1974

- DON -

MULTIPLE - PATH STACK ALGORITHMS FOR DECODING CONVOLUTIONAL CODES

by

David Haccoun, B.Sc.A. & Ing. Phys. (U of M)
S.M. (M.I.T.)

Department of Electrical Engineering

This work was submitted in April 1974 to the Faculty of Graduate Studies and Research of McGill University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

ABSTRACT

A new class of generalized stack algorithms for decoding convolutional codes is presented. It is based on the Zigangirov-Jelinek algorithm but instead of extending just the top node of the stack at all times, a number of the most likely paths are simultaneously extended. This number of paths may be constant or may be varied to match the current decoding effort with the prevalent noise conditions of the channel. Moreover the trellis structure of the convolutional code is exploited by recognizing the convergence of the paths. As a result the variability of the computation can be reduced up to a limit set by the ideal stack algorithm. Moreover the error probability is upper bounded by that of the ordinary sequential decoder. These algorithms close the gap between the one-path sequential decoding and the all-paths Viterbi decoding.

By allowing the correct path to be extended while not at the top of the stack, the apparent size of the correct path metric dip is reduced and small dips involve no search at all. Although the tail of the computational distribution is still Pareto, it is shown and verified from simulation with short constraint length codes ($K \leq 9$) of rate $1/2$, that compared to sequential decoding, the distribution of computations per decoded bit, the distribution of the metric dips and the maximum computational effort are all reduced at a cost of a modest increase in the average number of computations.

SOMMAIRE

Une classe nouvelle d'algorithmes généralisés utilisant une pile pour le décodage des codes de convolution est proposée. Ces algorithmes dérivent de l'algorithme de Zigangirov et Jelinek, mais au lieu de prolonger seulement le sommet de la pile, on prolonge simultanément un sous-ensemble des chemins les plus vraisemblables. Ce nombre de chemins peut demeurer fixe ou peut être varié pour adapter l'effort de décodage courant a l'état actuel du bruit dans la voie de transmission. De plus la structure en treillis des codes de convolution est mise en valeur par la reconnaissance et l'exploitation de toute convergence entre les chemins explorés. La variabilité du nombre de calculs peut ainsi être réduite jusqu'à une limite théorique correspondant à l'algorithme idéal. De plus la probabilité d'erreur est bornée (borne supérieure) par celle du décodeur séquentiel. Ces algorithmes font le lien entre le décodeur séquentiel et le décodeur de Viterbi.

Le chemin correct étant prolongé avant d'atteindre le sommet de la pile, la chute apparente de sa métrique cumulée est réduite. Bien que la queue de la distribution de l'effort de calcul suive la loi Pareto, on démontre et on observe par simulation avec des codes de taux 1/2 et de longueurs de contrainte courtes ($K \leq 9$),que comparées au décodeur séquentiel,les distributions du nombre de calculs par bit decodé, et de la chute apparente de la métrique sont réduits ainsi que le nombre maximum de calculs effectués. Ces améliorations sont obtenues pour un accroissement modeste du nombre moyen de calculs. De plus certaines erreurs du décodeur séquentiel sont corrigées.

ACKNOWLEDGEMENT

I wish to thank Professor Michael J. Ferguson for his guidance and stimulating discussions and criticisms during the course of this research.

I am grateful to l'Ecole Polytechnique de Montréal for its financial support, and to Hydro Québec for its scholarship.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

Communication is essentially the process of transmitting information from one point to another through a noisy channel. In a general communication system shown in Fig. (1.1), a message source generates messages that must be transmitted to a user over the channel. A specific signal is assigned to each of the possible M messages which the system can transmit, and the selection rule that assigns a transmitted signal to each possible message is called the code. Because of the channel noise the transmitted signal does not arrive at the receiver exactly as transmitted, and hence errors are made in conveying the source message to the user. The coding theorem of Shannon (1948) demonstrates the existence of codes that achieve reliable communication if and only if the information transmission rate R is less than some maximum rate C called the channel capacity.

Since Shannon's result appeared, a considerable amount of work has been concerned with the search of coding-decoding techniques that permit communication with low error probability, and which are simple to implement. The first code investigated were called block codes. In these codes, a sequence of K information symbols is encoded in a block of N symbols to be transmitted over the channel. Block codes have been studied extensively (Berlekamp 1968, Gallager 1968). Although Shannon proved that block codes exist for which the error probability decreases exponentially with the block length N, the long codes which are simply decoded yield a significantly greater error probability than predicted by the theory.

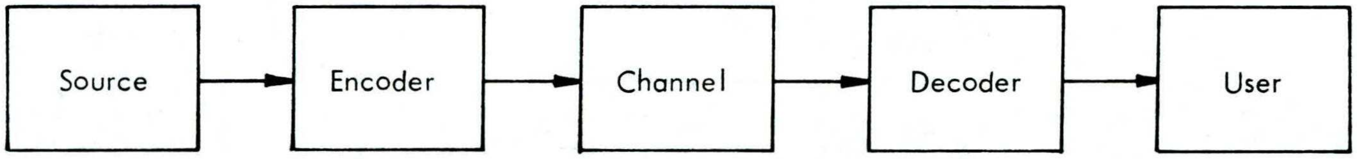


Figure (1.1) General communication system.

Convolutional codes are codes which are particularly suitable when the information symbols to be transmitted arrive serially rather than in large blocks. Randomly chosen convolutional codes are known which achieve error probabilities decreasing exponentially with the constraint length of the encoder. In general convolutional codes outperform block codes of the same order of decoder complexity.

In 1957 Wozencraft (1957) proposed a decoding procedure called sequential decoding whereby the tree-like structure of the convolutional code is used in a step by step search of the most likely transmitted message. As long as the rate does not exceed a quantity called R_{comp}, which is less than the channel capacity, the average number of computations necessary to decode one information digit is small and independent of the constraint length of the code.

There are two principal sequential decoding algorithms: the Fano algorithm (Fano 1963) and the Zigangirov-Jelinek (Z-J) stack algorithm introduced independently by Zigangirov (1966) and Jelinek (1969a). Regardless of the algorithm, a major problem with sequential decoding is the variability in the number of computations required per decoded digit. The cumulative distribution $P (c \geq N)$ of the number of computations performed per decoded digit is upper and lower bounded for the discrete memoryless channel by a Paretean distribution (Savage 1965, Jacobs and Berlekamp 1967).

The probability of error for sequential decoding decreases exponentially with the constraint length of the code whereas the coder complexity varies only linearly with the constraint length. Consequently, convolutional encoding and sequential decoding are among the most attractive means of achieving digital communication over memoryless channels at the low error probablility predicted by the theory.

In the tree representation of a convolutional code, a path is specified by the input message that entered the encoder. But in a convolutional encoder of finite constraint length K, the encoder outputs depend only upon the span of K', $K' < K$, past information input symbols. Hence the paths in the tree periodically remerge after their input sequences have been identical over K' input symbols, and the tree collapses in a trellis. The trellis model for convolutional codes was suggested by Forney (1967). Using this trellis model Viterbi (1967) proposed and analyzed a non-sequential decoding algorithm where all possibly distinct transmitted sequences are systematically examined in determining the most likely transmitted message. Viterbi decoding was shown to be optimum (Forney 1967, Omura 1969) but this optimality is obtained at a large decoding effort. In contrast with sequential decoding the number of computations performed per decoded digit is constant but increases exponentially with the constraint length.

Sequential decoding and Viterbi decoding are called "probabilistic" because the decoded message is obtained by probabilistic considerations rather than by a fixed set of algebraic operations. The error performance of sequential decoding is lower bounded by that of Viterbi decoding, but asymptotically as $K \to \infty$ both decoding techniques yield the same error performance.

Given the received sequence from the channel, sequential and Viterbi decoding appear to be at the extremes for determining the most likely transmitted message. The prime motivation for this research has been the desire to alleviate the variability of the computational effort of sequential decoding by closing the gap between the single-path sequential decoding and the all-paths Viterbi decoding.

Keeping the search properties of sequential decoding while using some of the concepts of Viterbi decoding, we develop a class of generalized stack decoding algorithms which unify these two seemingly different decoding techniques. The variability of the computational effort of sequential decoding is shown to be reduced without degrading the error performance.

The fundamental idea of the generalized stack algorithm is to avoid back-searching for the most likely transmitted sequence through the simultaneous extension of a subset of the most likely paths. Moreover some redundant and useless decoding effort is eliminated by using the trellis structure of the code. In the process, events which could lead to errors for sequential decoding are shown to be corrected by this technique. However, unless <u>all</u> distinct paths are constantly extended, a search mode of operation similar to that of sequential decoding appears to be asymptotically inescapable.

The generalized stack algorithm is further refined by exploiting the information about past decoding behaviours and current state of the channel noise in deciding on the actual number of paths that must be extended. Hence the decoding effort tends to be adapted to the particular requirements imposed by the channel noise. As a consequence both the computational behaviour and error performance can be further improved. The limit to this adaptive procedure is shown to be given by an optimum unrealizable ideal stack algorithm which lower bounds the computational performance of any stack decoding algorithm (including Viterbi and sequential decoding).

1.1 Outline of the Thesis

Chapter II is a review of the structure of convolutional codes. The essential concepts, error performance and computational behaviour of the sequential (Z-J) and Viterbi decoding algorithms are presented in Chapter III. The two decoding algorithms are compared, and methods to alleviate the computational variability of sequential decoding are introduced.

In Chapter IV we first show that the Z-J and Viterbi decoding algorithms are in fact only particular cases of a more general class of stack decoders. We then proceed to bridge the gap between these two algorithms by generalizing the Z-J decoding algorithm. A new M-path stack decoding algorithm is described, and its computational behaviour and error performance are analyzed. In this algorithm the useless computations of sequential decoding are eliminated by exploiting the reconvergence of the explored paths, and the variability of the computational effort is reduced by the simultaneous extension of the set of M, $M \geq 1$ most likely paths. Results of a simulation of the M-path algorithm with codes of constraint length $K \leq 7$, over a discrete memoryless gaussian channel are presented and compared to the Z-J algorithm .

The exploitation of some of the information imbedded in the stack, to help reduce further the variability of the computational effort without unduly increasing the average amount of computation is examined in Chapter V. Simulation results for variants of the M-path algorithm are presented in Section 5.1. An ideal adaptive stack decoding algorithm is proposed in Section 5.2 and shown to be the best algorithm in the class of generalized stack algorithms. A practical adaptive procedure is then described, modelled and its modes of operation analyzed by Markov chain techniques. The chapter ends with a discussion of simulation results for some implementations of the adaptive algorithm with short constraint length codes ($K \leq 9$).

Chapter VI contains final conclusions and suggestions for future research.

Appendix I contains a brief description of the generalized stack decoding algorithm as we have programmed it. Massey's Markov Chain model of the correct path metric differences is reviewed in Appendix II. This model is used to determine the average separation between breakout nodes of the correct path. In Appendix III a branching process model of the incorrect paths is used for a new approach in upper bounding the average decoding effort of sequential decoding. Appendix IV is an alternative method of determining the probability of entering a search mode of operation presented in Section 5.4.

1.2 Claim of Contributions of this Research

Some specific contributions of this research are summarized below.

A new class of generalized stack decoding algorithms is proposed. In this class, depending on whether the path extension decision rule is fixed or variable we distinguish two important subclasses of algorithms.

(a) The M-path algorithm which unifies the Z-J and Viterbi decoding algorithms.

(b) The adaptive algorithm where a variable number of paths are extended depending on how much the maximum stack metric has dipped. In this subclass the ideal algorithm is shown to be the best possible stack decoding algorithm.

In these algorithms we developped:

(a) A practical procedure to recognize and exploit the convergences of the paths.

(b) A simple method of extracting the exact top node of the stack.

We showed and verified through simulation with short constraint length codes ($K \leq 9$) that:

(a) Compared to sequential decoding, the variability of the computational effort is reduced. However, the tail of the computational distribution is still Pareto.

(b) The error probability is upper bounded by that of the ordinary sequential decoder and lower bounded by that of the Viterbi decoder.

Using Massey's Markov chain model of the metric differences on the correct path we established the expressions for

(a) The average separation between breakout nodes on the correct path.

(b) The effective reduction of this separation afforded by the generalized stack decoder.

(c) The probability to enter a search mode of operation.

Based on a branching process model of the incorrect paths, a new approach to bounding the average number of computations for the ordinary sequential decoding is presented.

CHAPTER II

STRUCTURE OF CONVOLUTIONAL CODES

2.1 Convolutional Encoder

A binary convolutional code of rate $1/V$ bits/symbol may be generated by a linear finite-state machine consisting of a K-stage shift register, V modulo-2 adders connected to some of the shift register stages, and a commutator that scans the output of the modulo-2 adders. The machine is called a convolutional encoder and is sketched in Fig. (2.1a).

Let us assume that the information to be encoded is a sequence

$$\underline{U} = (u_1, u_2, u_3, \dots u_L) \tag{2.1}$$

of binary letters, 0 or 1. Assuming the shift register to be initially filled with an all-zero sequence (or any other known sequence), the first binary digit u_1 is fed into the first stage of the shift register. The V modulo-2 adders are then sampled in sequence by the commutator, producing V output coded symbols

$$x_1^{(1)}, x_1^{(2)}, \dots x_1^{(V)} \tag{2.2}$$

where the output of an adder is "1" if and only if the total number of 1's in the shift register stages connected to the adder is odd. Otherwise the adder output is "0". After the Vth output symbol $x_1^{(V)}$ is obtained, the second input u_2 is shifted into the first stage of the shift register causing the contents of all other stages to move one step to the right. The right-most digit leaving the encoder is lost. The V modulo-2 adders are again sampled, yielding V new output symbols $x_2^{(1)}, x_2^{(2)}, \ldots x_2^{(V)}$. This procedure continues until the last input digit u_L enters the shift register. In order to return the shift register to its initial zero state, with each shift of u_L a zero symbol is fed-in until u_L leaves the shift register. This terminating sequence of $(K-1)$ zeros is called the <u>tail</u> of the message.

The L-bit message sequence $\underline{U}$ of Eq. (2.1) produces the output sequence or <u>codeword</u>

$$\underline{X} = (x_1^{(1)}, x_1^{(2)}, \ldots x_1^{(V)}, x_2^{(1)}, x_2^{(2)}, \ldots x_2^{(V)}, \ldots x_{K+L-1}^{(1)}, \ldots x_{K+L-1}^{(V)}) \quad (2.3)$$

of length $(K+L-1)V$ binary digits. The rate of the code is

$$R = \frac{L}{(L+K-1)V} \text{ bits/symbol} \quad (2.4)$$

and for $L >> K$

$$R \approx \frac{1}{V} \text{ bits/symbol.} \quad (2.5)$$

Let the row vector

$$\underline{G}_j = (g_{1j}, g_{2j}, \ldots g_{Vj}) , j = 1, 2, \ldots K \quad (2.6)$$

specify the connection between the stage j of the shift register and the V modulo-2 adders. The component g_{ij} is a 1 if the ith modulo-2 adder is connected to stage j, otherwise it is a 0. For example, the rate 1/3

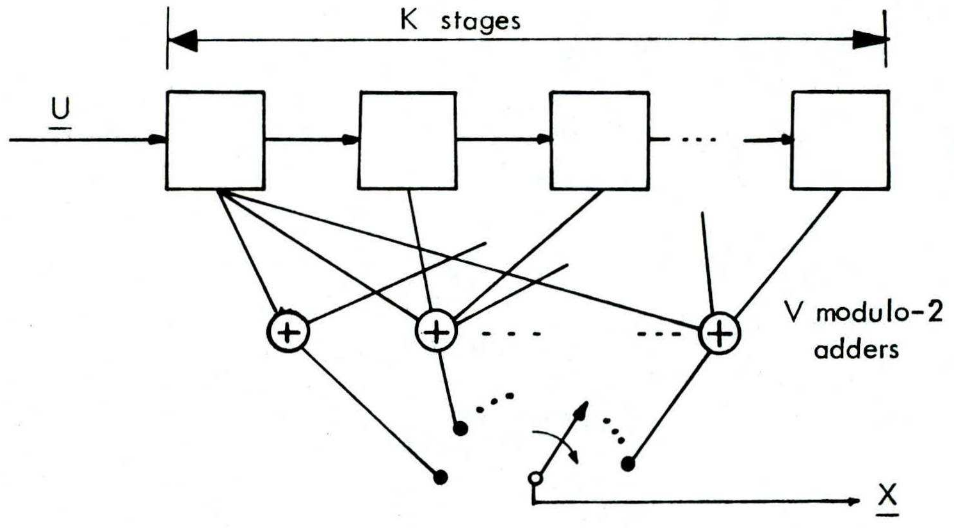


Figure (2.1a) Convolutional encoder.

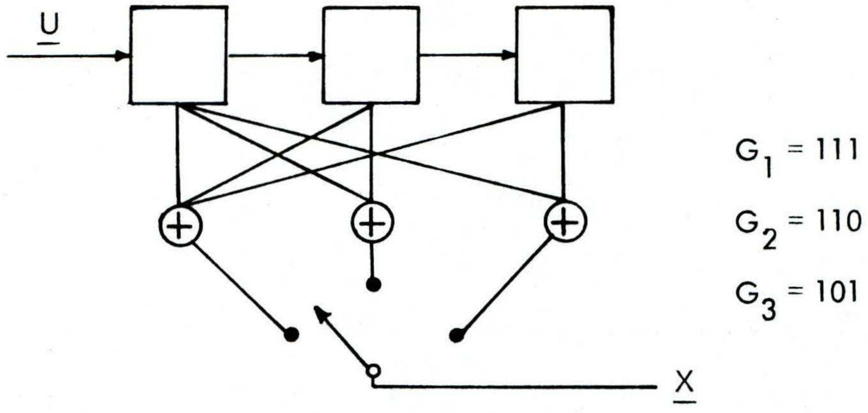


Figure (2.1b) Convolutional encoder with K = 3, V = 3, d = 1.

convolutional encoder encoder shown in Fig. (2.1b) has the following connection vectors

$$\underline{G}_1 = (1, 1, 1)$$
$$\underline{G}_2 = (1, 1, 0) \qquad (2.7)$$
$$\underline{G}_3 = (1, 0, 1)$$

From the operation of the encoder, the V output digits $x_i^{(1)}, x_i^{(2)}, \ldots x_i^{(V)}$ produced when the input binary digit u_i is first shifted in the shift register are

$$u_i \underline{G}_1 \oplus u_{i-1} \underline{G}_2 \oplus \ldots \oplus u_{i-K+1} \underline{G}_K \qquad (2.8)$$

where the sign $\oplus$ represents the modulo-2 addition. Relation (2.8) is the convolution of the vectors $\underline{G}$'s and the K-bit long input sequence $u_i, u_{i-1}, \ldots u_{i-K+1}$. The term "convolutional" is taken from the form of the relation (2.8) and clearly the vectors $\underline{G}$'s specify the code. We observe that the V output symbols corresponding to a particular input digit depend upon that digit and the (K-1) preceding digits that entered the encoder. The constraint length of the code is defined as the number of shifts over which a single information symbol can influence the encoder output. For the simple binary convolutional code, the constraint length is equal to K, the length of the shift register.

Considering the input binary sequence as an L-dimensional row vector $\underline{U}$, the (L+K-1) V - component output $\underline{X}$ can be written, using modulo-2 arithmetic, as

$$\underline{X} = \underline{U}\,[G] \qquad (2.9)$$

where

$$
\begin{array}{cc}
\longleftarrow (L+K-1)\,V \text{ columns} \longrightarrow & \\
[G] = \left[\begin{array}{cccccccc}
\underline{G}_1 & \underline{G}_2 & \underline{G}_3 & \cdots & \underline{G}_K & & & \\
 & \underline{G}_1 & \underline{G}_2 & \underline{G}_3 & \cdots & \underline{G}_K & & \\
 & & \underline{G}_1 & \underline{G}_2 & \underline{G}_3 & \cdots & \underline{G}_K & \\
 & & & \ddots & & & \ddots & \\
 & & & \underline{G}_1 & \underline{G}_2 & \underline{G}_3 & \cdots & \underline{G}_K
\end{array}\right] & \updownarrow L \text{ rows}
\end{array}
\qquad (2.10)
$$

The matrix $[G]$ is called the generator matrix of the code, and the KV-component vector

$$\underline{G}^* = (\underline{G}_1, \underline{G}_2, \underline{G}_3, \ldots \underline{G}_K) \qquad (2.11)$$

is called the generator of the code. The first row of the matrix $[G]$ is the generator $\underline{G}^*$ followed by $(L-1)V$ zeros, and each succeeding row is the previous row shifted V places to the right, with all elements to the left of $\underline{G}_1$ equal to zero. The number of rows is the length L of the input sequence, and the number of columns is $(K+L-1)V$. Every codeword $\underline{X}$ can be expressed as a linear combination of the rows of $[G]$ and the code may thus be seen as the set of all 2^L linear combinations of the rows of $[G]$. For the example of Fig. (2.1b), the generator of the code is

$$\underline{G}^* = (111 \quad 110 \quad 101)$$

and for the 4 bit long input sequence

$$\underline{U} = (1011)$$

the output is

$$\underline{X} = (111,\ 110,\ 010,\ 001,\ 011,\ 101)$$

One can generalize this binary convolutional encoder by allowing the K-stage shift register to receive groups of d q-ary information symbols of an input sequence whose components are elements of a finite Galois field GF (q). Then the input to the encoder is the sequence

$$\underline{U} = (\underline{u}_1, \underline{u}_2, \underline{u}_3, \dots \underline{u}_L) \tag{2.12}$$

where

$$\begin{aligned} \underline{u}_i &= (u_i^{(1)}, u_i^{(2)}, \dots u_i^{(d)}), \\ u_i^{(j)} &\in GF (q),\quad j = 1, 2, \dots d,\quad i = 1, 2, \dots L \end{aligned} \tag{2.13}$$

The V modulo-2 adders, $V > d$ and their connections are replaced by V inner product computers, each of which compute the inner product in GF (q) of the shift register contents and some specified vector. In other words, the elements of the generator matrix $[G]$ belong to GF (q), and the operations of Eq. (2.9) are performed by arithmetic operations in GF (q). The length K is chosen to be a multiple of the integer d,

$$K = kd \tag{2.14}$$

and the constraint length of the code becomes the number $k = K/d$ of <u>groups of d shifts</u> over which a single information symbol can influence the encoder output. The rate of the code is

$$R = d \, \frac{\ln q}{V} \qquad \text{nats/symbol} \tag{2.15}$$

Following the encoding of the L groups of d information symbols, a tail of $(k-1)$ groups of d zeros is shifted in to clear the shift register.

In practice it may be simpler to consider the d-tuple input sequences entering the encoder in parallel. The K-stage shift register is replaced by a stack of d k-stage shift registers, each with its own kV-component generator vector. The generator of the whole code is then a $d \times kv$ matrix.

$$[G^*] = \begin{bmatrix} \underline{G}_1^* \\ \underline{G}_2^* \\ \vdots \\ \underline{G}_d^* \end{bmatrix} \tag{2.16}$$

where $\underline{G}_i^*$, $i = 1, 2, \ldots d$ is the generator corresponding to the ith shift register. The generator matrix $[G]$ of the whole code has the same form as that of the simple binary code of Eq. (2.10), but with each row replaced by the matrix $[G^*]$ of Eq. (2.16). For example the rate 2/3 code with generator vectors

$$\underline{G}_1^* = (\, 111 \quad 011 \quad 001 \quad 100 \,)$$

$$\underline{G}_2^* = (\, 101 \quad 001 \quad 111 \quad 011 \,)$$

is shown in Fig. (2.2). The input sequence starting with 11001001 ..., will be encoded as 010 010 001 001 ... .

2.2 The Structure of Convolutional Codes

From the above description of the operation of a convolutional encoder, we observe that

(1) The code symbols depend on past values of the inputs.

(2) The past dependency does not extend to the infinite past but is limited to the length of the shift register.

(3) The code is linear since the output symbols are linear combinations of the past inputs.

We now show that the first two observations lead to two different representations of the convolutional codes.

2.2.1 Tree Structure

The fact that at any time, an input may take q^d values hence leading to q^d output sequences, suggests representing the output of a convolutional encoder by a tree with q^{dn} possible sequences corresponding to n inputs. The

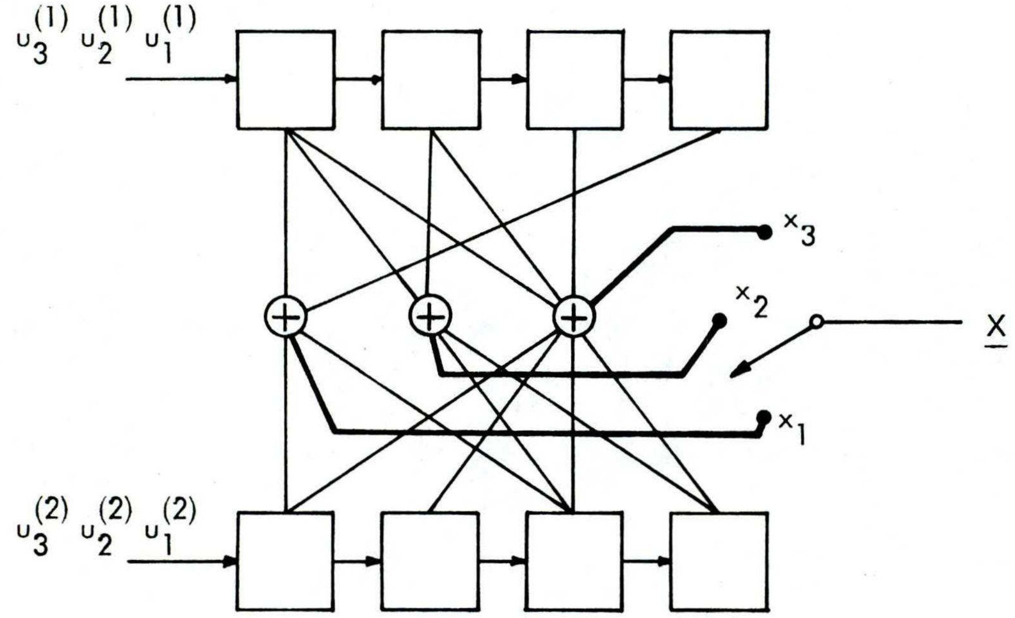

$u_3^{(1)}\ u_2^{(1)}\ u_1^{(1)}$
x_3
x_2
$\underline{X}$
x_1
$u_3^{(2)}\ u_2^{(2)}\ u_1^{(2)}$

Figure (2.2) Rate 2/3 convolutional encoder with $K = 8$, $V = 3$, $d = 2$.

root node of the tree has no predecessor while every other node has one predecessor. There are q^d branches stemming out of each node, with the exception of the nodes corresponding to the tail of the message which have only a single branch extension. All branches carry V coded symbols. Fig. (2.3) depicts the tree representation for the outputs of the encoder of Fig. (2.1b).

A path in the tree is traced from left to right according to the input sequence that specifies it. For the binary tree of Fig. (2.3), a 0 input means taking the upper branch leaving a node, and a 1 means taking the lower branch. Hence a message sequence $\underline{u}_1, \underline{u}_2, \underline{u}_3, \ldots$ traces a path through the tree, and the corresponding coded symbols of the branches of the path are the channel inputs that are transmitted. In Fig. (2.3), the input message starting with the sequence 01100 determines the path indicated by the thick line having the encoded sequence 000 111 001 011 101. The tree of Fig. (2.3) is said to have five levels, one for each branching along a possible path. The number of levels in a tree can be extended indefinitely, and naturally there is a one-to-one correspondence between the set of all q^{dL} information sequences of length dL and the set of paths through the L levels of the tree. A code that can be represented by a tree is called a tree code. In general, tree codes are codes where coded symbols depend on past input sequences and where paths may be represented as branch choices on a tree.

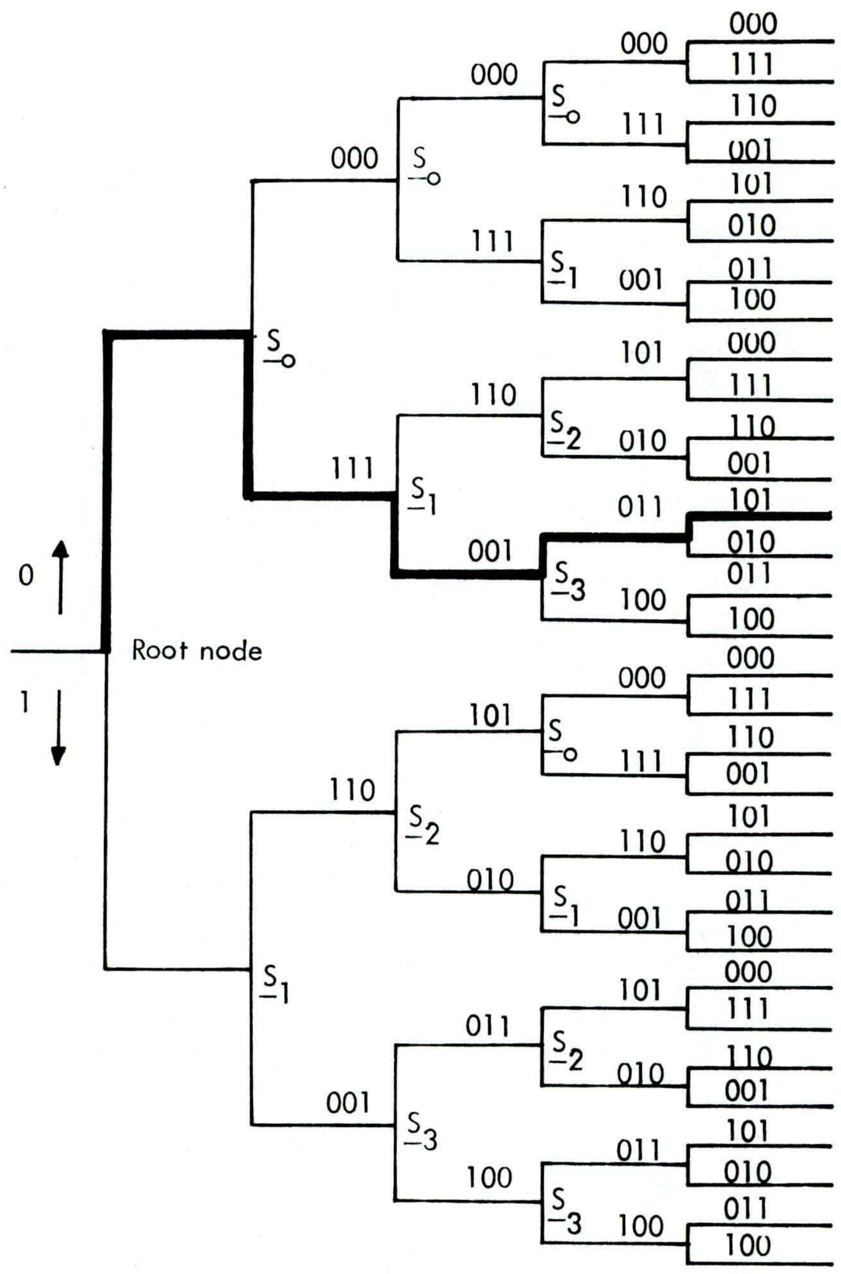


Figure (2.3)   Tree representation of encoder of Fig. (2.1b).

2.2.2 State Diagram

In the causal transformation of the input sequence $\underline{U}$ into the coded sequence $\underline{X}$, at any given time the V output digits are completely determined by the most recent k groups (including the present one) of d information symbols that entered the encoder, and are residing in the shift register. For every new shift of input symbols, the output will depend upon the previous $(k-1)$ shifts of input data, and clearly, if the first nd components of 2 or more input sequences are equal, the corresponding output sequences will be identical in their first nV components.

A convolutional encoder was defined as a finite state machine, and since a finite state machine changes from one state to another according to the input it receives, the state of a convolutional encoder is then simply the $(k-1)$ input data symbols preceding the current input. That is, the state is the content of the first $(k-1)d$ shift register stages, and hence the encoder state together with the new d input symbols uniquely specify the V output symbols. We can label the states by the sequence

$$\underline{S} = (s_1, s_2, \ldots s_{K-d}) \tag{2.17}$$

where $s_i \in GF(q)$, $i = 1, 2, \ldots K-d$, and where s_i entered the encoder before s_{i+1}.

The total number of states is then

$$q^{(k-1)d} = q^{K-d}$$

and from one state the machine may move to q^d other states. Naturally, some input symbols may cause the machine to stay in the same state.

This description suggests representing the encoder by a state diagram with q^{K-d} states or nodes and q^d branches leaving and entering each state. These branches represent the transitions of the encoder from one state to another, and therefore they carry the V coded symbols delivered by the encoder in the transition. For example, the state diagram representation of the encoder of Fig. (2.1a) is shown in Fig. (2.4). It has 4 states and each branch is labelled by the corresponding input (above) and output V-tuple (below).

The state diagram is a very compact representation of the encoder. Starting from the initial state $\underline{S}_o = (0, 0, \ldots 0)$, the coded output corresponding to the input sequence $\underline{U} = (u_1, u_2, u_3, \ldots)$ is readily obtained by writing the branch symbols corresponding to the transitions due to $u_1, u_2, u_3, \ldots$. For example, from Fig. (2.4), the output to the message 110100 is 111, 001, 011, 010, 110, 101. A potentially infinite tree has been reduced to a mere 4-state diagram!

Regarding the state diagram as a signal flow graph, particular codes can be analyzed via the transfer function of their diagram (Viterbi 1971). However, the great difficulty in finding the transfer function for long constraint length codes limits the use of this technique to very short constraint length codes.

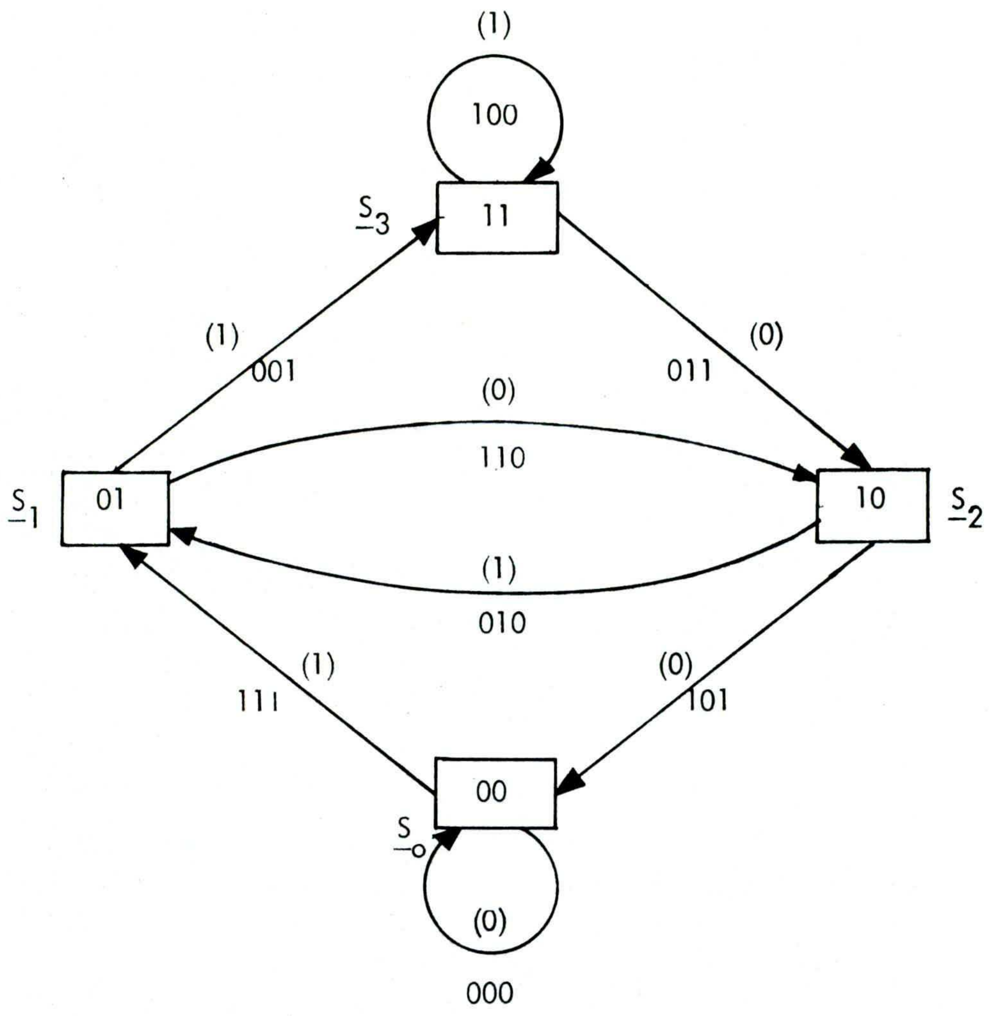


Figure (2.4) State diagram for encoder of Fig. (2.1b).

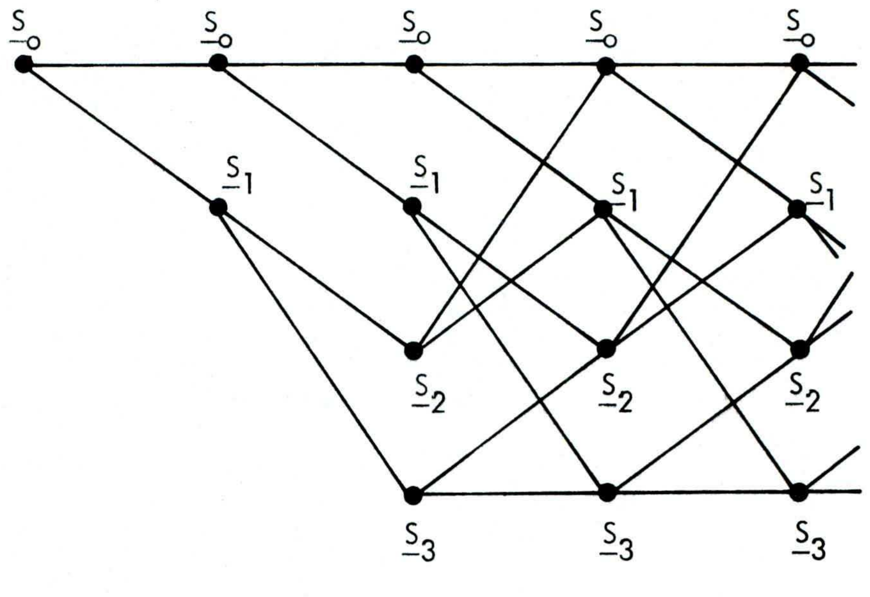


Figure (2.5) Trellis representation of encoder of Fig. (2.1b).

2.2.3 Trellis Structure

Using the definition of a state, to each node of the tree there corresponds an encoder state. For an input sequence of dL symbols, there are q^{dL} terminal nodes in the L-level tree, but only q^{K-d} different states. Therefore, for $L \geq (K/d)$, the number of nodes in the tree exceeds the number of states of the encoder. This means that several nodes must correspond to the same encoder state, and hence the coded tree paths are not distinct over their entire length. From our discussion on the state diagram we know that after the input symbols have been identical for $(K-d)$ consecutive symbols, the encoded symbols are the same: the paths remerge. However, this fact is obscured in the tree representation of the code since the tree keeps on branching off with q^d branches from each node, as if the states were all different. This behaviour is equivalent to considering an infinite number of states or a dependence over the infinite past. Realizing that the number of states is finite, we see that the tree contains much redundant information which can be eliminated by merging together, at any same level, all nodes corresponding to the same encoder state. The redrawing of the tree with remerging paths is called a trellis.

Taking the example of Fig. (2.3), let the four states by $\underline{S}_0 = (0,0)$ $\underline{S}_1 = (0,1)$; $\underline{S}_2 = (1,0)$; $\underline{S}_3 = (1,1)$. After the third level, the data sequence 100 a b c ..., and 000 a b c ... generate the same code symbols, and hence both nodes in the figure labelled $\underline{S}_0$ can be joined together. The same reasoning applies for the nodes labelled $\underline{S}_1$, $\underline{S}_2$ and $\underline{S}_3$, and the tree collapses in the trellis of

Fig. (2.5). More insight about the remerging of the tree paths in general is given by the following properties (Viterbi 1967).

Property 1:

If two sequences of information symbols are identical except for n consecutive groups of d information symbols, the corresponding codewords will be distinct over $(n+k-1)$ consecutive branches.

Proof:

The paths in the tree code corresponding to the 2 codewords will be identical as long as the input sequences are the same (prior to the nd different symbols), and will diverge, i.e. will be different, for the n consecutive branches where the input symbols are different. Thereafter, although the input symbols are the same for the two data sequences, the code symbols will be different as long as the encoder states are different. Hence, $(k-1)$ additional shifts are necessary before obtaining identical outputs, causing the paths to diverge over a total of $(n+k-1)$ branches.

Property 2:

If the information symbols of two paths agree at some point in $(k-1)$ branches, the subtrees extending from these two paths thereafter must be identical.

The trellis representation of a convolutional code is more instructive than the tree representation for it explicitly uses the fact that the encoder is a finite state machine. Each state of the trellis has q^d possible successors, q^d possible predecessors, and from level to level the structure is repetitive.

This reduction of the tree structure of the code into a trellis is credited to Forney, (1967) and the trellis structure is at the heart of an optimum decoding technique for convolutional codes due to Viterbi (1967). This decoding technique will be presented in section 3.4.

CHAPTER III

DECODING OF CONVOLUTIONAL CODES

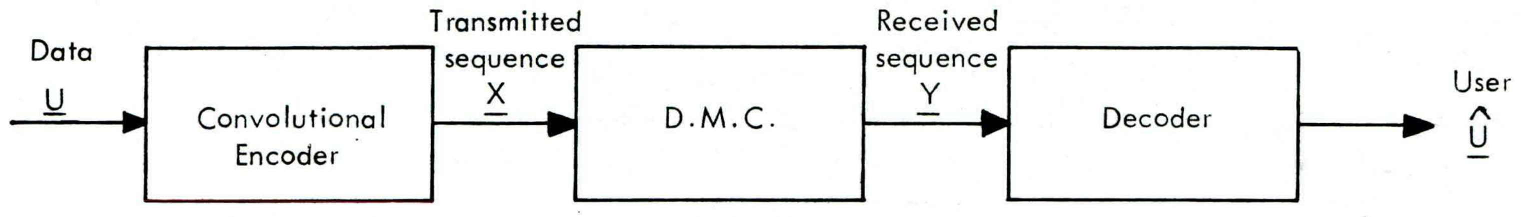
3.1 Introduction

In a terminated convolutional code, the message input sequence $\underline{U}$ is encoded as a sequence $\underline{X}$ and is represented by a particular path through the tree or trellis of the code. This codeword $\underline{X}$ (also called the transmitted sequence) is sent through a noisy memoryless channel to a decoder and a user.

A discrete memoryless channel (DMC) is an idealized model of a noisy channel with digital input and quantized or digital outputs. The input to a DMC is a sequence of letters from a Q-symbol alphabet, and the output is a sequence of letters from a J-symbol alphabet. During each channel use, a letter of the input sequence is transmitted, and the corresponding output letter is received. A received letter j, $j = 1, 2, \dots J$ is assumed to be statistically dependent only on the corresponding input letter i, $i = 1, 2, \dots Q$, and is determined by a fixed conditional probability assignment $P(j/i)$. Successive input-output transitions are random and statistically independent.

The decoder observes the output sequence of the DMC corresponding to the transmitted sequence, and determines which of the possible q^L q-ary data sequences of length L entered the encoder. Fig. (3.1) shows a communication system employing a convolutional code and a DMC.

The encoder simply maps the integers 1 to $M = q^L$ onto the codewords $\underline{X}^{(1)}$ to $\underline{X}^{(M)}$. As message m enters the encoder, $\underline{X}^{(m)}$ is



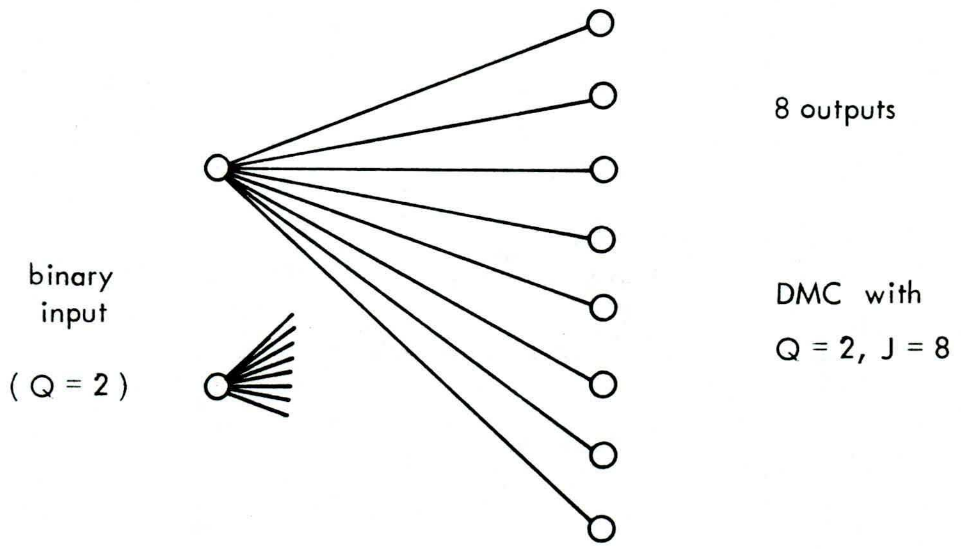


Figure (3.1)   Communication system and Discrete Memoryless channel.

transmitted, and on the basis of the corresponding received sequence $\underline{Y}$, the decoder produces an integer m'. Decoding errors occur if $m \neq m'$.

It is well known (Gallager, 1968) that for completely general channels, the decoder, which, given $\underline{Y}$ chooses m' for which

$$P(\underline{Y} \mid \underline{X}^{(m')}) \geq P(\underline{Y} \mid \underline{X}^{(m)}), \text{ all } m \neq m' \tag{3.1}$$

minimizes the error probability of the sequence, if all input data sequences are equally likely. This decoder which is optimum is called a "maximum likelihood sequence decoder" and the conditional probabilities $P(\underline{Y} \mid \underline{X}^{(\cdot)})$ are called the likelihood functions.

Specializing the discussion to convolutional codes, the following notation will be used. There are in general q^d branches leaving each node and V transmitted symbols per branch. A node on the tree level (or tree depth) ℓ is uniquely specified by the data sequence $\underline{U}_\ell = (\underline{u}_1, \underline{u}_2, \ldots \underline{u}_\ell)$ which locates the encoder at that node. Each subsequence $\underline{u}_i = (u_1^{(i)}, u_2^{(i)}, \ldots u_d^{(i)})$, $d < V$, $u_i^{(j)} \in GF(q)$ is the set of d symbols that entered the encoder at time i. If $\underline{U}_i$, $1 \leq i < \ell$ is the initial sequence of length i of $\underline{U}_\ell$, then we may write

$$\underline{U}_\ell = (\underline{U}_i, \underline{u}_{i+1}, \underline{u}_{i+2}, \ldots, \underline{u}_\ell) \tag{3.2}$$

The V symbols on the last branch reaching node $\underline{U}_\ell$ are denoted by

$$\underline{x}_\ell^{(\underline{U})} = (x_{\ell 1}^{(\underline{U})}, x_{\ell 2}^{(\underline{U})}, \ldots x_{\ell V}^{(\underline{U})}) \tag{3.3}$$

Likewise let

$$\underline{y}_j = (y_{j1}, y_{j2}, \dots y_{jv}) \tag{3.4}$$

represent the V received symbols when $\underline{x}_j^{(\underline{U})}$ is transmitted. Then from Eq. (3.1) the likelihood function for the jth branch of that path is

$$P (\underline{y}_j | \underline{x}_j^{(\underline{U})}) = \prod_{i=1}^{V} P (y_{ji} | x_{ji}^{(\underline{U})}) \tag{3.5}$$

since the channel is assumed memoryless.

Generally it is more convenient to use the logarithm of the likelihood function since the logarithm is a monotonely increasing function, and therefore does not alter the final result. Defining the log-likelihood function $\gamma_j^{(\underline{U})}$ for the jth branch on the path specified by the input sequence $\underline{U}$ as

$$\gamma_j^{(\underline{U})} \triangleq \log P (\underline{y}_j | \underline{x}_j^{(\underline{U})}) = \sum_{i=1}^{V} \log P (y_{ji} | x_{ji}^{(\underline{U})}) \tag{3.6}$$

the log-likelihood function $\Gamma_N^{(\underline{U})}$ for the first N branches of that path is

$$\Gamma_N^{(\underline{U})} = \sum_{j=1}^{N} \gamma_j^{(\underline{U})} \qquad N = 1, 2, 3, \dots \tag{3.7}$$

(In the sequel whenever there is no ambiguity as to which path is being considered, the superscript $\underline{U}$ will be removed).

The log-likelihood function (or simply the likelihood) is used as a <u>metric</u>, and for memoryless channels the metric is additive over the received symbols. Because a metric can be computed for each path in the tree or trellis, maximum likelihood decoding of convolutional codes may be simply regarded as the finding of the path with the largest accumulated metric, given the received sequence. However, depending on whether the tree or trellis structure of the code is used, the determination of this most likely path will lead to either an impractical or practical decoding technique.

3.2 Maximum Likelihood Decoding of Convolutional Codes

By definition, maximum likelihood decoding implies comparing the received sequence with all possible transmitted sequences before making a decision. Hence, in general for memoryless channels, the optimal decoding of an L-bit long binary sequence requires comparing the 2^L accumulated metrics of the 2^L different codewords that could have been transmitted, and picking the best one. Because of this exponential increase of the decoding effort with the length L of the sequence, maximum likelihood decoding is usually difficult to implement and therefore rarely used in practice. However, its importance lies in its use as a standard with which other practical suboptimum decoding techniques may be compared. Moreover, it is used to determine the performance of codes, since a measure of performance for any code is the probability of error with the optimum decoder.

For the decoding of convolutional codes, either the tree or trellis structure of the code may be used. In the tree representation of the code, the fact that the paths remerge is entirely ignored, and hence the decoding of an L-bit sequence requires the exhaustive comparison of 2^L accumulated metrics. Consequently over the tree structure of the code, optimal decoding is not practical, and given the received sequence, the estimation of the most likely data sequence would call for a suboptimum but more practical decoding technique.

Consider now the trellis structure of the code where the redundancy of the tree was eliminated through merging. In principle this redundancy could be exploited by a clever decoder which would consider only those paths that could ever maximize the likelihood function over the whole set of paths. In the decoding process, if at some point it is realized that a path cannot possibly yield the largest metric, then that path would be ignored by the decoder. The decoded path would then be chosen from among the set of remaining or "surviving" paths that reached the last level L. Such a decoder would still be optimum in the sense that the decoded path would be the same as the decoded path of a "brute force" maximum likelihood decoder, but the early rejection of unlikely paths would reduce the decoding effort. The objective is naturally to find a procedure that breaks the exponential message sequence-length dependency of the decoding effort and yet yields maximum likelihood decisions. We now show that the trellis structure of the code allows such a decoding procedure.

Consider the set of all tree paths lying at level (or depth) ℓ, $1 \leq \ell \leq L$, whose end node correspond to the same encoder state $\underline{S}_i^{(\ell)}$, $i = 1, 2, \ldots q^{K-d}$, where the superscript refers to the level of the tree. These paths are distinct in the tree but converge into a single node in the trellis. In this set, the path having the largest accumulated metric is called the survivor at state $\underline{S}_i^{(\ell)}$, and all other paths constitute the non-survivor set at $\underline{S}_i^{(\ell)}$. It is readily shown that an optimum decoder can discard the non-survivor set of paths without altering the optimum decision.

Since the subtrees issued from a set of converging paths are identical, the metric differences existing between the survivor and all non-surviving paths at $\underline{S}_i^{(\ell)}$ will be maintained thereafter up to the last level L, for all identical paths of these subtrees (see Fig. (3.2)). Therefore, among the paths belonging to these subtrees, the path yielding the largest accumulated metric at level L was the survivor at $\underline{S}_i^{(\ell)}$. A maximum likelihood path cannot belong to any non-survivor set which can thus be discarded.

It is clear that a decoder that would compare the metrics of all paths converging into a node, keeping only the survivor at that node will yield a maximum likelihood decision if the operation is repeated for all distinct states at each level. The natural structure to use is of course the trellis. There are q^{K-d} states per level, hence q^{K-d} survivors must be determined for each level, yielding a decoding effort that would vary as q^{K-d} for a code of constraint length K/d. Therefore, the exponential growth of the decoding effort is in the constraint length of the code, not in the length of the input sequence as in maximum likelihood decoding of tree codes. Whether or not this effort is tolerable will depend on the particular application, but it may already be concluded that maximum likelihood decoding of convolutional codes using their trellis structure will be limited in practice to codes of short constraint lengths.

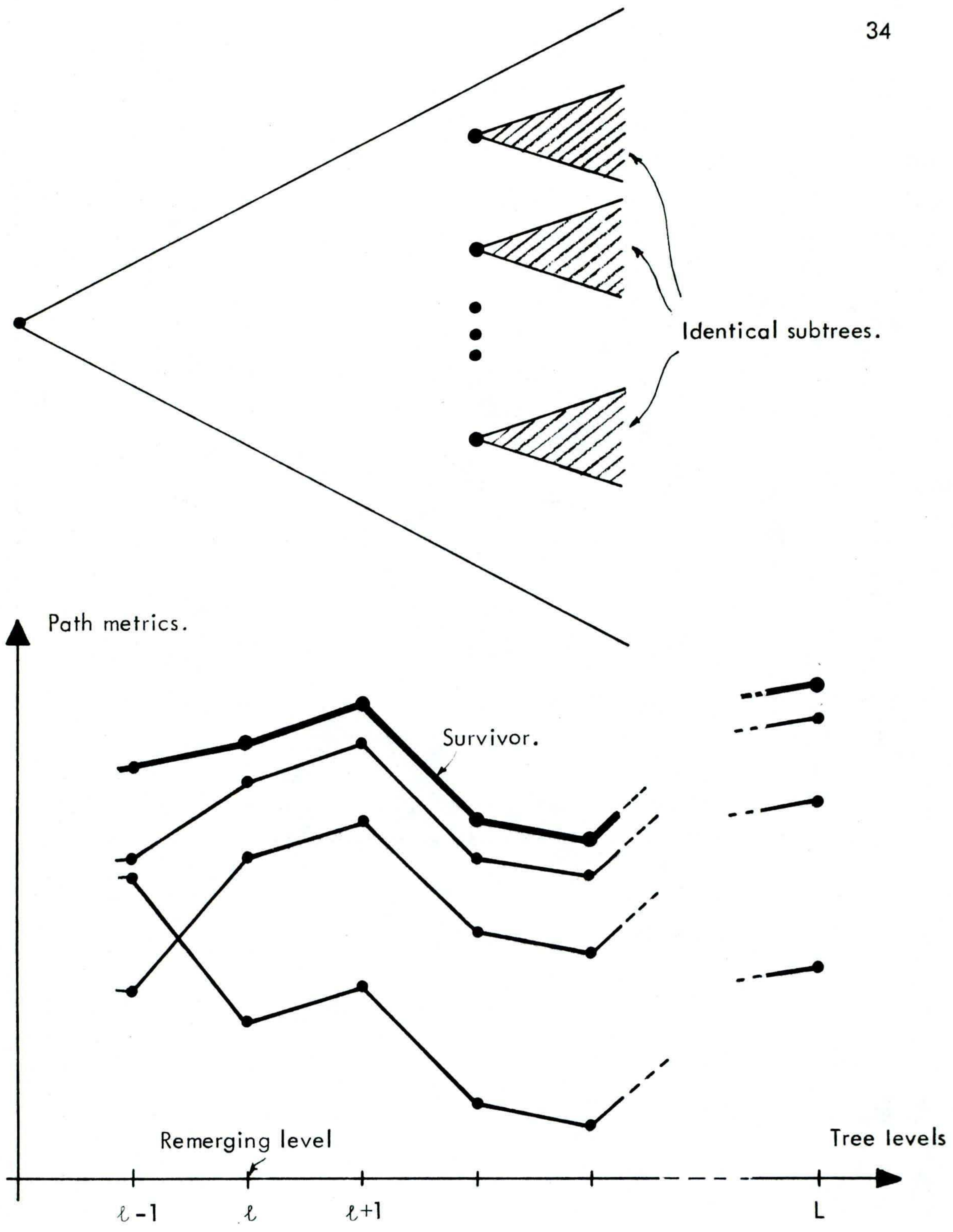


Figure (3.2) Path metrics of remerging paths.

A very practical optimum decoding technique is the Viterbi decoding algorithm invented by Viterbi (1967). This decoding procedure uses the trellis structure of the code, and is very efficient in the way the periodic remerging of the paths is systematically exploited to determine the survivors. The optimality of the Viterbi decoder was demonstrated by Forney (1967) and Omura (1969). Forney showed that for any convolutional code, the output of the decoder is a sequence of estimated information digits which is maximum likelihood conditioned upon the received sequence, and Omura showed that the Viterbi algorithm was in fact a forward dynamic programming technique.

3.3 Suboptimum Decoding

Since the number of states increases exponentially with the constraint length, maximum likelihood decoding becomes very rapidly impractical. However, given the received sequence, the search for the most likely path can be attempted sequentially, one branch at a time, making tentative decisions (or hypotheses) on successive branches of explored paths, in such a way that the path being followed is the most likely among the subset of paths currently examined. Whenever a subsequent choice indicates an earlier incorrect decision, this decision is properly modified, and hence at each decoding step, the current most likely path is chosen among different paths lying at different levels in the tree (or trellis). Clearly the explored path is only locally most likely.

This step by step procedure is facilitated by the tree (or trellis) structure of the code, and by following only the current most likely path, unlikely sequences are eliminated until a single sequence of length L remains; this sequence is accepted as the decoded path.

The elimination of an unlikely path based on the observation of its first m branches, $m < L$, is of course equivalent to the elimination of <u>every</u> path beginning with these m branches, i.e., a whole subtree of length $(L-m)$. The earlier the rejection, the larger (exponentially) the number of paths discarded, reducing the decoding effort but also making the procedure clearly suboptimum. This idea of extending only the current most likely path together with the concomitant path rejection is credited to Wozencraft (1957) who exploited it in a particular decoding procedure for tree codes called "<u>sequential decoding</u>".

One could improve on this suboptimal search procedure by extending not just the most likely path, but the set of the M-most likely paths. By increasing the set of paths considered in the determination of the most likely sequence, such a procedure "reduces" the suboptimality of sequential decoding at a cost of a larger decoding effort. Moreover, instead of the tree, the trellis structure of the code could be used by allowing the decoder to recognize and exploit the convergence of paths just like an optimum decoder.

Sequential decoding and Viterbi decoding are two powerful and practical decoding techniques. They are termed "<u>probabilistic</u>" because the decoded sequence is obtained by probabilistic considerations rather than by a fixed sequence of algebraic operations.

These two techniques are presented in the next section, whereas the new decoding technique that extends the M-most likely paths will be presented in the next chapter.

There exists also a number of non-probabilistic (or algebraic) suboptimum decoding techniques for convolutional codes, for example Massey's threshold decoding (Massey 1963), suitable for codes having certain algebraic (orthogonal) properties. In general these decoding techniques lead to inferior performance compared to Viterbi or sequential decoding, and will not be discussed in this thesis.

3.4 Probabilistic Decoding Algorithms

This section is a brief introduction to the two main probabilistic decoding techniques introduced in the preceding sections: the tree search of sequential decoding and the trellis search of Viterbi decoding.

For both decoding techniques, the decoder is assumed to have a replica of the encoder and hence is able to duplicate all possible transmitted sequences that constitute the code. Successive sequences of V channel output symbols called "received branches" are received from the channel, and from these the corresponding branch metrics can be computed. Assuming a DMC, the metric along the paths is cumulative and therefore, for both techniques, the objective of the decoder is to find the path that yields the largest accumulated metric, given the received sequence. Viterbi (or optimum) decoding represents an exhaustive search that takes advantage of the code topology, whereas sequential decoding uses an intuitive trial-and-error search method to reduce the average decoding effort.

3.4.1 Tree Search: Sequential Decoding

The central idea of sequential decoding is the decoding of the received message one symbol at a time rather than decoding all information symbols simultaneously as in maximum likelihood decoding. Intuitively, one could suspect that in many instances, the correct path has a larger metric than the incorrect paths diverging from it, and hence could be accurately estimated by considering only the succeeding few branches. Starting from the origin of the tree, the path selected to be searched one step further (i.e., extended one level deeper in its q^d branches) is the path that has the largest accumulated metric among those already examined. Consequently, by extending only the path that appears the most promising, most of the computations necessary for an optimum decoding can be avoided. This idea is common to various algorithms known as sequential decoding algorithms, the specific method of searching and selecting the path to be extended depending on the particular algorithm. Among the subset of explored paths, the path that reaches the last level of the tree (including the tail) with the highest metric is accepted as the decoded path.

Jacobs and Berlekamp (1967) have given two conditions which a decoding algorithm must satisfy in order to be a sequential decoding algorithm.

(1) The branches of the explored part of the tree are examined sequentially, and the decision on which path to extend is based only on those paths already examined. Each new path is thus an extension of a previously examined one.

(2) The decoder performs at least one computation for each of every examined path.

Algorithms which do not have these two properties are not considered to be sequential decoding algorithms.

The interesting feature of such a decoding procedure is that the rejection of a branch is in fact the rejection of a whole subtree whose root node is the end node of the rejected branch.

As the decoding proceeds, the number of paths in the subset of explored paths grows, and occasionally the decoder goes back in the tree and extend early and possibly incorrect paths. A fundamental idea of sequential is to bias the metrics (on which the search decisions are based) in such a way that the backing-up and extension of unlikely paths is reduced to a minimum. For a memoryless channel, the branch metric $\gamma_{i}^{(U)}$ of the branch $\underline{x}_{i}^{(U)} = (x_{i1}^{(U)}, x_{i2}^{(U)}, \dots x_{iV}^{(U)})$ of a path $\underline{U}$ is generally taken as

$$\gamma_{i}^{(U)} = \sum_{j=1}^{V} [\log \frac{P(y_{ji} | x_{ji}^{(U)})}{f(y_{ji})} - B] \tag{3.8}$$

where $\underline{y}_i = (y_{i1}, y_{i2}, \dots y_{iV})$ is the sequence of V received symbols corresponding to the transmitted branch $\underline{x}_{i}^{(U)}$, and where $f(j)$ is the nominal probability of output j for a DMC with an input probability assignment $w(i)$, $i = 1, 2, \dots Q$ and transition probabilities $P(j/i)$, $j = 1, 2, \dots J$. That is

$$f(j) = \sum_{i=1}^{Q} w(i)\, P(j/i) \tag{3.9}$$

the bias term B is chosen in such a way that, on the average, the branch metrics are positive along the correct path and negative along the incorrect paths. As the metric assigned to each node on a path is the sum of the branch metrics along the path leading to that node, the accumulated metric for node $\underline{U}_N$ is

$$\Gamma_{N}^{(U)} = \sum_{i=1}^{N} \gamma_{i}^{(U)} \qquad N = 1, 2, 3, \dots \tag{3.10}$$

The metric of Eq. (3.8) is then a "tilting" of a modified form of the log-likelihood function of Eq. (3.6). This biasing or "tilting" of the likelihood function is common to all sequential decoding algorithms, and the value of the bias which minimizes the average computational effort per decoded digit is equal to the rate of the code, (Fano 1963, Bucher 1970). Clearly then, the consequence of biasing the likelihood function is that along the correct path the accumulated metric tends to increase, while along any incorrect path it tends to decrease. Therefore, although the objective of any sequential decoder is to find the path that yields the largest accumulated metric, the strategy is to search and follow the path of increasing metric value.

There are two main sequential decoding algorithms: the Fano algorithm (Fano 1963) and a search algorithm introduced independently by Jelinek (1969a) and Zigangirov (1966). Although seemingly different, both algorithms search the tree from its root node out, trying to "match" sequentially the beginning segment $\underline{Y}_i$, $i = 1, 2, \ldots L$ of the received sequence $\underline{Y}_L$ to the corresponding initial segments of the various paths of the tree. The Fano algorithm introduced by Fano in 1963 is a modification of the original algorithm presented by Wozencraft in 1957. A detailed description of the algorithm may be found in either Wozencraft and Jacobs (1965) or Gallager (1968), and will not be given here. The Fano algorithm is quite complex, the simplicity of the decoding principles being somewhat lost in the details of the rules that govern the motion of the decoder. It is however, quite popular, and has been used to derive most of the theoretical results on sequential decoding.

(a) <u>The Zigangirov-Jelinek (Z-J) Algorithm</u>

The Z-J algorithm is a very simple algorithm where the essential concepts of sequential decoding are readily apparent. The decoder consists of a list or <u>stack</u> of the already searched paths ordered in decreasing order of their metric values. The "top" of the stack is the path of largest accumulated metric among the paths in the stack, and will be searched further (extended one level further in its q^d branches) since it is the one which is the most likely to be the correct path. As the stack is reordered after each extension, a path whose metric is ever-increasing will continue to be searched further. Should the metric decrease and drop from the top position, that path will be properly stored in the stack and the new top node will be extended.

Denoting each explored path by the node of its extremity, the stack can equivalently be considered as a list of nodes ordered according to their metric values. The objective of the decoder is the finding of the top node, the extension of its successors, and the proper reordering of the stack. After initially loading the stack with the origin node whose metric is taken as zero, the decoding algorithm consists of the following rules:

(1) Compute the metrics of all successors of the top node and enter them in their proper place in the stack.

(2) Remove from the stack the node whose successors were just inserted.

(3) If the new top node is the final node, stop. Otherwise go to (1).

When the algorithm gets out of the loop, the top node is the end node of the decoded path. The whole path is then easily recovered from the information stored in the stack.

A decoding example for a binary tree is illustrated in Fig. (3.3). The paths are associated with their terminal nodes which are numbered in the figure. The numbers written on top of the branches represent the corresponding branch metrics for some received sequence $\underline{Y}_4$ of length 4 branches. Ordering the nodes from left to right, the contents of the stack during decoding is then:

	Step No.	Stack contents : node (metric)
Initialization	1	0 (0)
	2	2 (-1), 1 (-7)
	3	5 (-5), 6 (-5), 1 (-7)
	4	6 (-5), 11 (-6), 1 (-7), 12 (-12)
	5	11 (-6), 1 (-7), 13 (-9), 14 (-9), 12 (-12)
	6	1 (-7), 13 (-9), 14 (-9), 23 (-10), 24 (-10), 12 (-12)
	7	4 (-8), 13 (-9), 14 (-9), 23(-10), 24(-10), 12(-12), 3(-14)
	8	10 (-8), 13 (-9), 14 (-9), 23 (-10), 24 (-10), 9(-10), 12 (-12), 3 (-14)
	9	21 (-6), 13 (-9), 14 (-9), 23 (-10), 24 (-10), 9(-10), 12 (-12), 3 (-14), 22 (-18).

The decoded path is thus specified by the data sequence terminating at node 21, that is (0, 1, 1, 0).

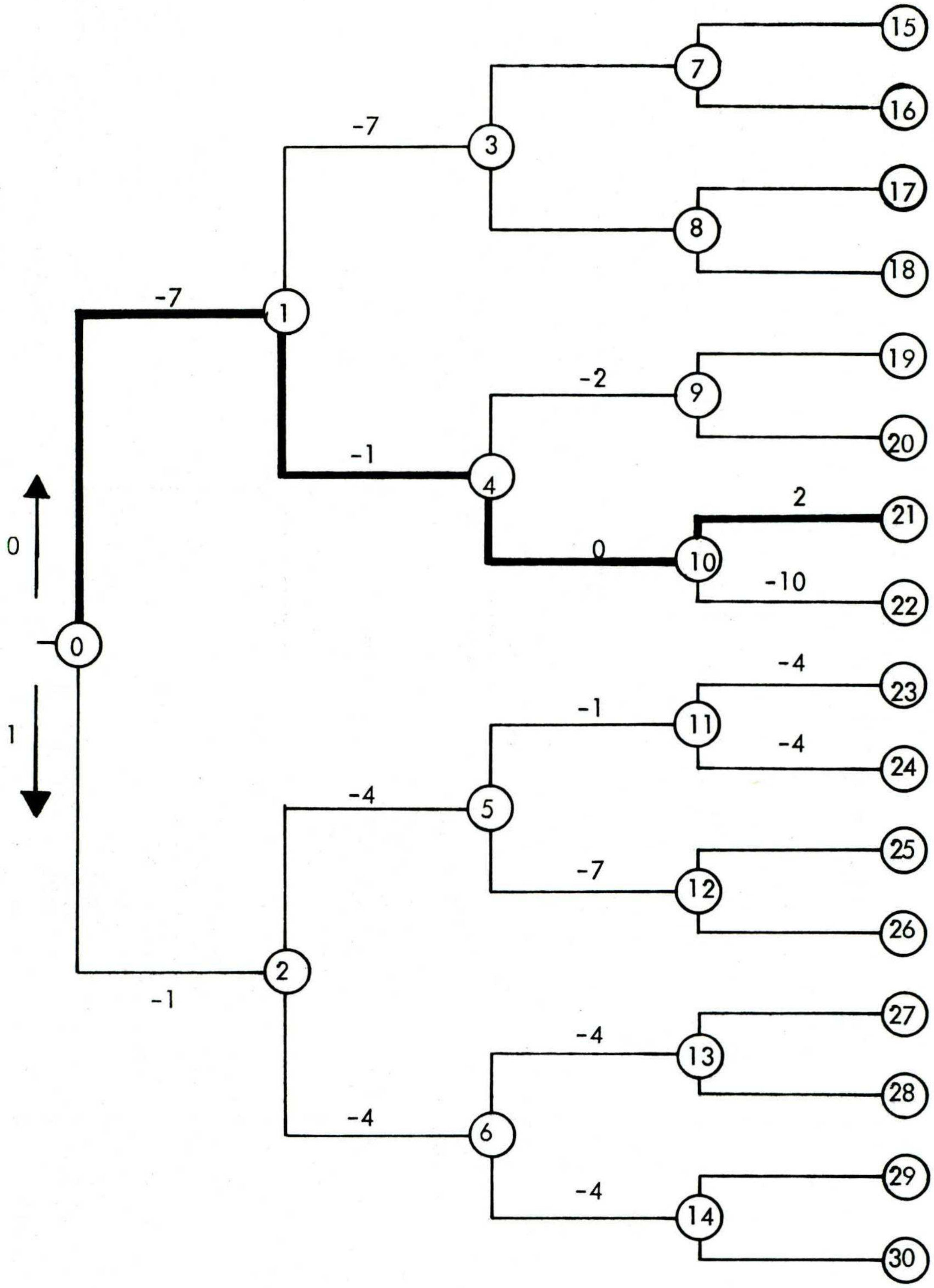


Figure (3.3) Branch metrics on a tree. The heavy line corresponds to the decoded path.

With the help of this example, the Z-J algorithm may be regarded as the most natural way to extend that explored path having the largest biased metric. By always extending the most likely path among those already explored (that is, stored in the stack), the algorithm maximizes the probability that the next step will be taken along the correct path in the encoded tree.

The mechanics of the search are quite simple in principle. After the top node is eliminated, the q^d new successors are inserted in the stack at the place assigned by their total metrics. The stack grows by (q^d-1) entries at each decoding step, so that after j steps, it contains $(1+j(q^d-1))$ paths of various lengths terminating in different nodes in the tree. Defining a computation as the execution of step 1 of the algorithm, the total number of computations to decode a binary tree is equal to $(W-1)$ where W is the size of the stack when decoding stops.

A graph of the metric values of the paths called the "received value tree" may be constructed as in Fig. (3.4) for the example of Fig. (3.3). At the 4th level obviously the decoded path has the highest value. However, this is not the case for any other intermediate node level and by the rules of the algorithm, all paths whose metric is higher than the correct path metric must be eventually extended. A typical example of the plot of the correct path metric is shown in Fig. (3.5). A span of bad noise causes the metric to drop, and therefore some incorrect paths may occupy the top of the stack and be extended. Clearly, before the correct path reaches the top of the stack and is extended anew (hence has

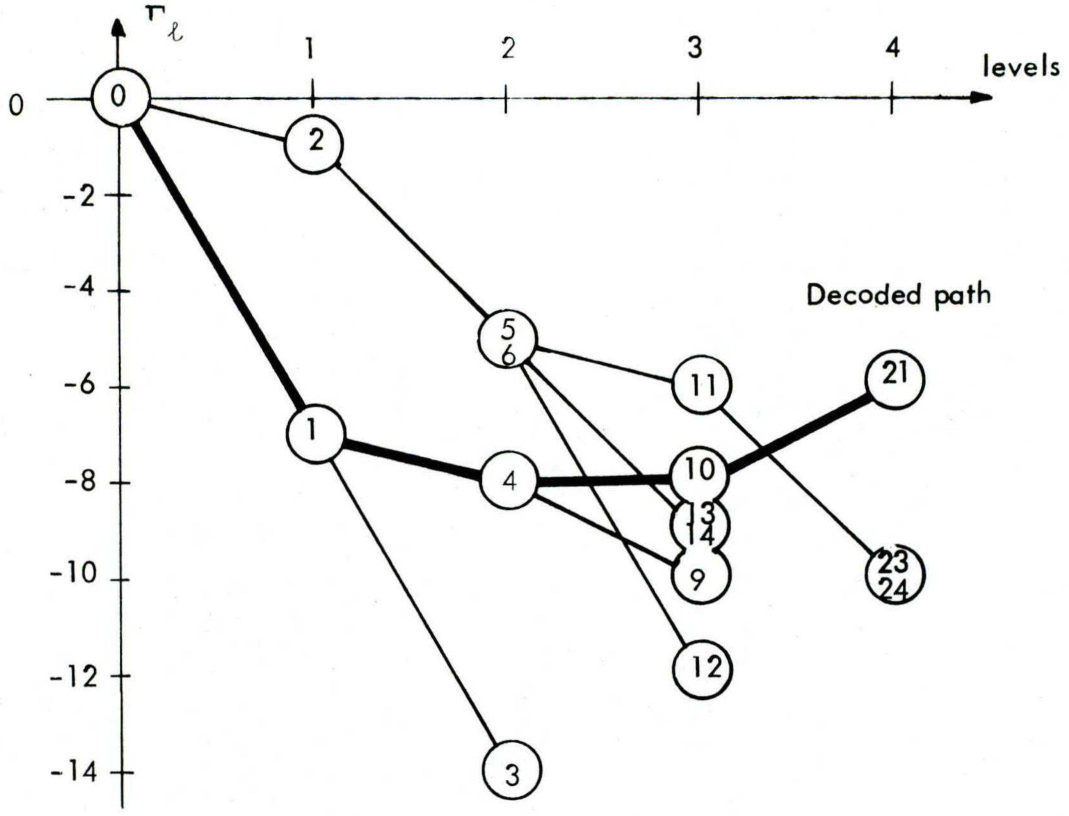


Figure (3.4) Metric values of the explored paths.

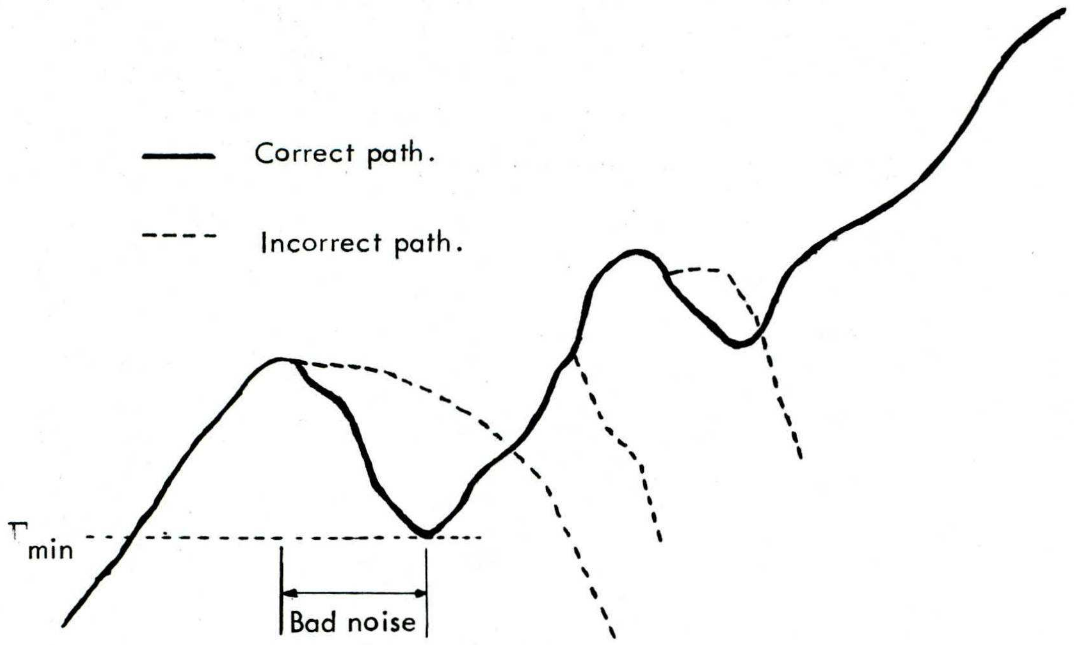


Figure (3.5) Typical correct and incorrect paths metrics.

passed the region of bad noise), all incorrect paths with a metric higher than the smallest value of the correct path metric must have been extended. This observation is fundamental to understanding the computational behaviour of sequential decoding. It is the occasional drop in the correct path metric and the concomitant extension of incorrect paths that is responsible for the random nature of the decoding effort.

(b) Quantized Z-J Algorithm

The simplicity of the Z-J algorithm is paid for by requiring a large memory for storing all the information about the explored paths. However, the real problem is keeping the stack exactly ordered. An exact ordering of the nodes is so time consuming that the algorithm becomes practically worthless. To overcome this difficulty, Jelinek (1969a) proposed a quantized version of the algorithm in which the nodes are placed at random into substacks (also called bins) according to their metric values. In each substack are stored all those nodes whose metric value lies within a certain range. That is, a node m of metric Γ_m is inserted at random into substack Q if

$$QH \leq \Gamma_m < (Q+1)\,H \tag{3.16}$$

where H is the substack spacing in metric value.

In this quantized version, the search for the top node reduces to the search for the highest non empty substack clearly a much simpler task.

The node which is to be extended is then taken at random from this top substack, usually in the last-in first-out mode. This quantized version of the Z-J algorithm is sometimes called the Jelinek algorithm and the rules become:

(1) Compute the metrics of the successors of any node from the highest non-empty substack and enter them in their proper substack.

(2) Remove from the stack the node whose successors were just inserted.

(3) If any node in the highest non-empty substack is the final node, stop. Otherwise go to (1).

Under this quantized form, the Z-J algorithm becomes practical and competitive with the older Fano algorithm. In a comparative study of the two algorithms, Geist (1970) has shown that although quite different, both algorithms essentially follow the same rules of path extension. Considering the decoding time as a measure of the decoding effort, Geist observed by simulation that for cases of low noise, the Fano decoder performs better than the stack decoder, but the advantage goes rapidly to the stack decoder as the noise increases. Hence the stack decoder would be superior than the Fano decoder for periods of high channel noise and vice-versa for periods of calm channel noise. The choice of the particular decoder will depend on a trade-off between storage, speed and software sophistication.

(c) The Computational Problem of Sequential Decoding

Regardless of the algorithm used, sequential decoding involves a random motion into the tree, where the exact moves of the decoder are determined by the received sequence and the particular algorithm. Since any decoder move implies a computation, the number of computations to decode a given block of information symbols is a random variable. Consequently, the analysis of sequential decoding is not only concerned with the error probability but also with the distribution of the computational effort. This variability of the computation is one of the draw-backs of sequential decoding.

A combination of asymptotic results by Jacobs and Berlekamp (1967), Savage (1965), Falconer (1967), and Jelinek (1969b) shows that the number of computations required to decode one information symbol for any sequential decoding algorithm has a Pareto distribution (under the assumption of an infinite constraint length code). Thus ignoring the convergences of paths,

$$P(C \geq N) \approx k_1 N^{-\alpha} \tag{3.12}$$

for large N. k_1 is a constant. The exponent α is called the Pareto exponent and is given by the parametric equation

$$R = \frac{E_o(\alpha)}{\alpha} \tag{3.13}$$

where R is the information rate of the code and $E_o(\alpha)$ is the Gallager function (Gallager 1968) defined as

$$E_o(\alpha) = \max_{\underline{\omega}} \left[-\log \sum_{j=1}^{J} \left[\sum_{i=1}^{Q} \omega(i)\, P(j/i)^{\frac{1}{1+\alpha}} \right]^{1+\alpha} \right] \quad (3.14)$$

for a discrete memoryless channel with Q inputs having a distribution $\underline{\omega} = \{\omega(i), i = 1, 2, \ldots Q\}$, J outputs, and transition probabilities $\underline{P}(j/i)$. $E_o(\alpha)$ is a concave monotonic non-decreasing function of α and is determined by the channel statistics. This function has the properties that

$$E_o(0) = 0$$

$$E_o(1) \triangleq R_{comp} \quad (3.15)$$

$$\lim_{\alpha \to 0} \frac{d(E_o(\alpha))}{d\alpha} = \mathbf{C}$$

Where $\mathbf{C}$ is the channel capacity, and the rate $R_{comp} = E_o(1)$ is called the computational cut-off rate of sequential decoding. For a discrete memoryless channel $R_{comp} < \mathbf{C}$, and for most of the channels, $\mathbf{C} > R_{comp} \geq \mathbf{C}/2$.

Examination of the moments of the Pareto distribution shows that if the exponent α is less than 2, the variance is unbounded, and for $\alpha \leq 1$, the mean does not even exist: the average number of computations to decode one bit becomes unbounded. The rate for which $\alpha = 1$ is recognized as R_{comp}, and is the limiting rate for sequential decoding. Clearly R_{comp} is an important parameter for sequential decoding systems.

The theoretical values of the exponent α and R/R_{comp} may be obtained by a graphical construction on the graph of $E_o(\alpha)$ as shown in Fig. (3.6). Observing that $E_o(\alpha)$ is monotone non-decreasing, then

$$E_o(\alpha) \geq R_{comp}, \quad \alpha \geq 1 \tag{3.16}$$

and from Eq. (3.13) we obtain

$$\alpha \geq \frac{R_{comp}}{R} \quad \text{for } R \leq R_{comp} \tag{3.17}$$

Similarly for $\alpha \leq 1$

$$\alpha \leq \frac{R_{comp}}{R} \quad \text{for } R \geq R_{comp} \tag{3.18}$$

Observe that a larger Pareto-exponent will make the tail of the distribution of the computation $P(C \geq N)$ decrease more rapidly. This is a desirable situation which for a given DMC can be obtained only by decreasing the information rate of the code.

Many experimental investigations (Jordan & Bluestein 1963, Niessen 1965) confirm the Paretean nature of the distribution of the computation for the coherent and incoherent discrete memoryless channel (Haccoun 1966, Heller 1967), and the fading dispersive channel (Wright 1967). An intuitive simple argument may explain this behaviour. When the noise causes the correct path metric to dip, the decoder goes into a back search and enters a subset of incorrect paths. Since the number of paths in the incorrect subset grows exponentially

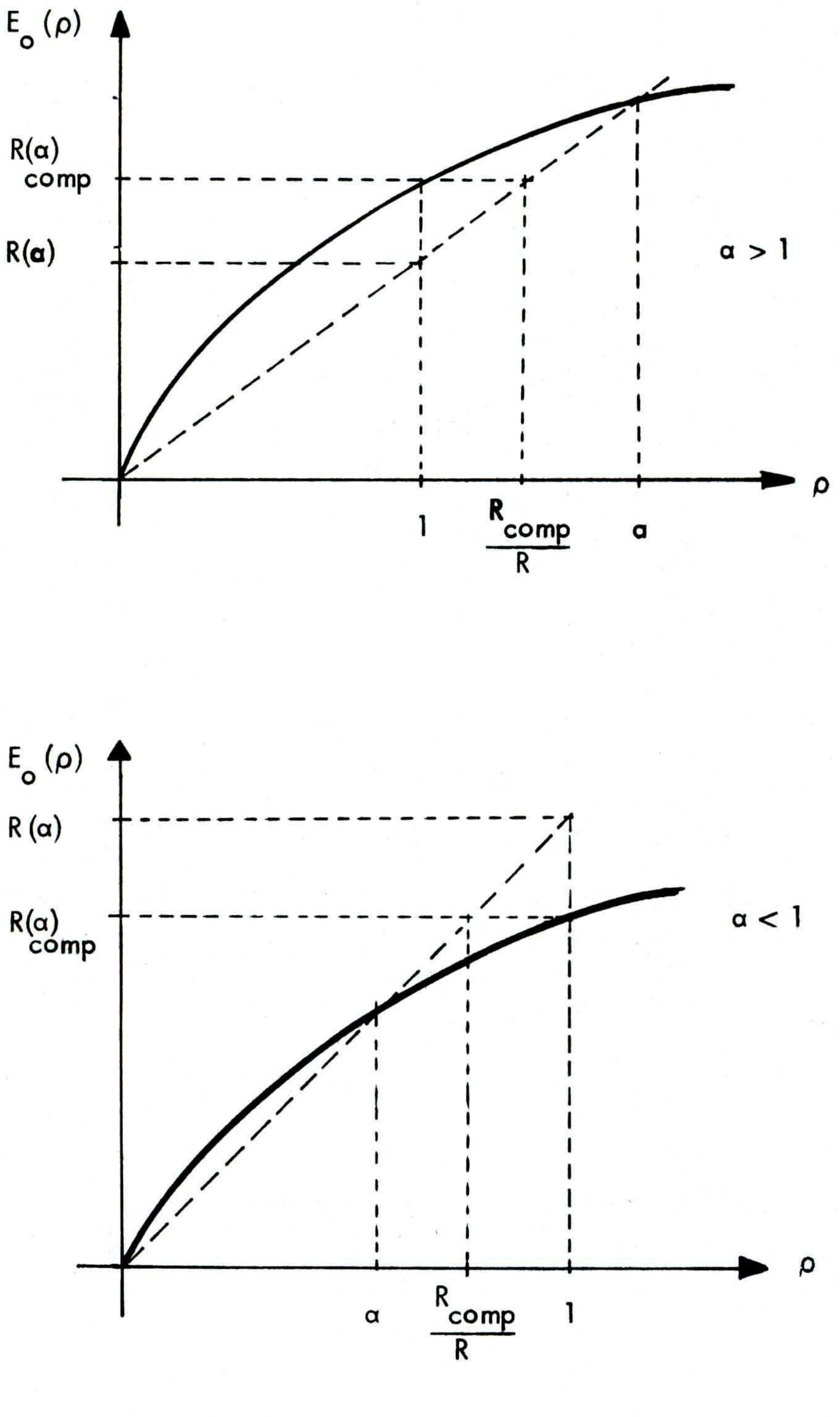


Figure (3.6) Graphical construction for determining the Pateto exponent.

with the penetration into the subset, the number of required computations will also grow exponentially with the length of the correct path metric dip. However, on the discrete memoryless channel, the interval of high noise that caused the correct path metric to dip, occurs with a probability that is exponentially decreasing with its length. The combined effect of these two exponential behaviours results in a distribution which decreases at most algebraically.

Another unpleasant consequence of the variability of the computation is that incoming data may not be always immediately processed as it arrives, and buffer storage must be provided. Regardless of the size of the buffer, there is a non-zero probability that it may fill-up, leading to an overflow and complete communication breakdown. This overflow problem is the most serious problem with sequential decoding on the discrete memoryless channel. The probability of overflow has been bounded by Savage (1965) as

$$P\ (\text{overflow}) \approx F\ (S,\ B)^{-\alpha} \tag{3.19}$$

where S is the decoder speed factor in computations per information digit interarrival time, B is the size of the buffer and α is the Pareto exponent. The probability of overflow is rather difficult to combat because of its relative insensitivity to buffer size and decoder speed.

To overcome this major difficulty, data is sent in blocks, usually in the order of 500 to 1000 branches, and known sequences (the tail of the message) clear the shift register of the convolutional encoder after each block.

In case of an overflow, the entire block is lost, but the buffer is cleared and decoding can be resumed in the following block.

The sequential decoder is said to have made an error if it makes an incorrect decision on an information bit and never returns to correct it. The error probability in sequential decoding decreases exponentially with the constraint length of the code and hence can be made arbitrarily small. Sequential decoding being a suboptimal procedure, a true lower bound on its error probability is obviously the lower bound on the error probability for the optimal decoding of Viterbi (1967). Asymptotically the bound is

$$P(E) > k_2 \, e^{-K E_o(\alpha)/R}, \quad R < \frac{E_o(\alpha)}{\alpha} \tag{3.20}$$

where k_2 is a constant, K the constraint length of the code, R the rate and $E_o(\alpha)$ the Gallager function of Eq. (3.14).

Long constraint lengths present no problem in actual sequential decoders because the coder complexity varies only linearly with the constraint length and the probability of overflow is insensitive to it. Therefore, the real problems with sequential decoding are the variability of the computational effort and the buffer overflow.

These difficulties have somewhat been alleviated by a hybrid scheme due to Falconer (1967), where instead of using one sequential decoder, a mixture of several sequential decoders working in parallel and an algebraic block decoder are used. The system works in such a way that periodically the algebraic decoder helps those sequential decoders involved in long searches by putting them in the right path. Compared to straight sequential decoding, the computational variability of Falconer's scheme is reduced, but the distribution of the computation remains asymptotically of a Pareto-type. The increased complexity of the hybrid scheme has so far prevented its implementation in practice.

A new method to reduce the computational variability is to allow the decoder to extend several paths simultaneously, and by using the trellis of the code exploit the convergence of the paths and eliminate unnecessary computations and storage. This method which will be presented in the next chapter, involves only a slight modification of the Z-J algorithm, and in addition makes sequential decoding attractive for short constraint length codes.

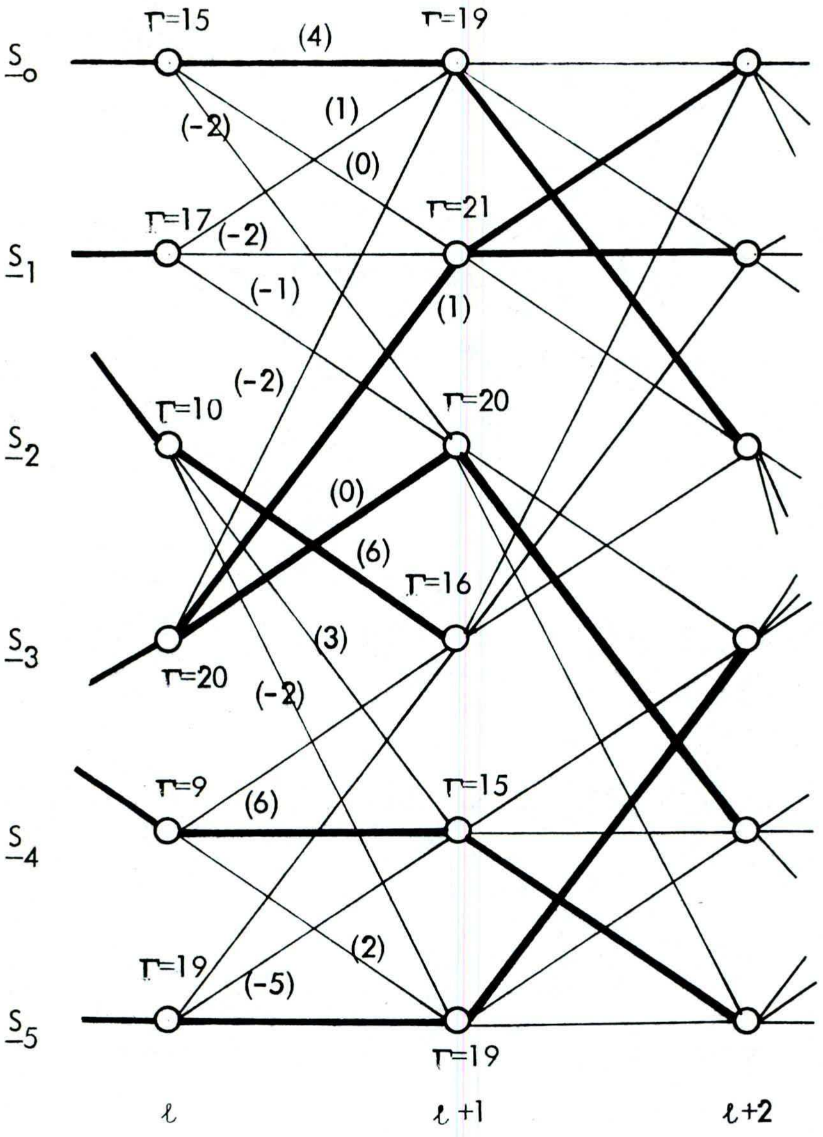
3.4.2 Trellis Search: Viterbi Decoding

The Viterbi decoding algorithm (Viterbi 1967) is a simple decoding procedure, which delivers a sequence of estimated information digits, maximum-likelihood conditioned upon the received sequence. In contrast with sequential decoding, the Viterbi decoded sequence corresponds to the path having the largest accumulated metric of <u>all</u> possible distinct paths.

We recall from section 3.2 that of all the paths remerging at a given node, only the survivor at that node needs to be retained by the decoder, and consequently the trellis is the structure to use for decoding. At any trellis depth (or level) ℓ further than one constraint length, the survivor for each of the q^{K-d} distinct states must be determined, which implies that <u>all</u> distinct paths of length ℓ branches must have been examined. Since none of the possibly valid paths are discarded, there is no need to bias the likelihood function, and hence the metric will be the log-likelihood function of Eq. (3.6).

In the trellis structure, a node is entirely specified by its state and its level. That is, $\underline{S}_i^{(\ell)}$ specifies a node at level ℓ, $\ell = 1, 2, \ldots$, having state $\underline{S}_i$, $i = 1, 2, \ldots q^{K-d}$. Given the survivor for each node at some level ℓ, the survivor at any node $\underline{S}_i^{(\ell+1)}$ is easily obtained by considering only those q^d 1-branch extensions (emerging from the q^d survivors at level ℓ) that merge onto node $\underline{S}_i^{(\ell+1)}$. Consequently, the survivor at node $\underline{S}_i^{(\ell+1)}$ is the extension of that survivor which yields the largest accumulated metric at node $\underline{S}_i^{(\ell+1)}$. For example in Fig. (3.7), the survivor at $\underline{S}_2^{(\ell+1)}$ is the extension of the survivor at node $\underline{S}_3^{(\ell)}$ and not of the survivors at either $\underline{S}_1^{(\ell)}$ or $\underline{S}_o^{(\ell)}$.

In case none of the extensions of a survivor yield a survivor at the next level, then clearly that survivor cannot become part of the decoded path and hence may be discarded. This situation is illustrated in Fig. (3.7), where none of the successors of the survivors at nodes $\underline{S}_1^{(\ell)}$ and $\underline{S}_3^{(\ell+1)}$ were chosen, and hence these survivors are eliminated. Summarizing we see that a surviving path at any level is always a surviving path at all of its preceding levels, but it is not necessarily a part of a surviving path at any following level. This observation is at the heart of the Viterbi decoding algorithm which can be regarded as a set of rules for obtaining the survivor at each state of every trellis level.



The heavy lines represent the surviving paths.

Figure (3.7) Surviving and non-surviving paths in the trellis.

(a) The Viterbi Decoding Algorithm

The Viterbi decoding algorithm operates as follows where we may assume $d = 1$ without loss of generality. Given the first $(K-1)$ branches of the received sequence, $\underline{Y}_{K-1}$ $(= \underline{y}_1, \underline{y}_2, \dots \underline{y}_{K-1})$, start by examining, from the origin out, all q^{K-1} paths of length $(K-1)$ branches and compute their total metrics $\Gamma_{K-1}^{(i)}$, $i = 1, 2, \dots q^{K-1}$. Since no merging takes place for the first $(K-1)$ consecutive branches of the tree, those q^{K-1} paths are distinct and identical for both tree and trellis. Upon reception of the Kth branch $\underline{y}_K$, each path extends its q single-branch to level K, where q branches converge into each node. The total metrics $\Gamma_K^{(i)}$, $i = 1, 2, \dots q^K$ of the q paths converging into each node $\underline{S}_i^{(K)}$, $i = 1, 2, \dots q^{K-1}$ are compared, and only the path with the largest metric (the survivor at node $\underline{S}_i^{(K)}$) is retained. The other $(q-1)$ single-branch extensions that converged on $\underline{S}_i^{(K)}$ are discarded. All q^{K-1} survivors are determined in this way. Now finding the survivors at level $(K+1)$ involves only the extension of each survivor into its q successors, computing for each the branch metrics $\gamma_{K+1}^{(i)}$, $i = 1, 2, \dots q^K$, (given the received branch $\underline{y}_{K+1}$), and comparing q-wise the total metrics Γ_{K+1}'s of the converging paths.

The mechanics of the decoding are now apparent: at each step the q^{K-1} surviving paths are extended and the comparison is made among the paths which were generated by input sequences not previously discarded. Out of each comparison a single path is chosen, hence at each step the extensions increase the number of paths by a factor of q while the comparisons reduce that number by a factor of q, resulting in a constant number of survivors. From our earlier discussion we know that the paths eliminated in the reduction from q^K new paths to q^{K-1} new survivors do not affect the optimality of the final decision.

A difficulty may arise in the case of ties: the survivor at a node is not unique, that is, two or more paths yield the same highest accumulated metric. Keeping all the contenders would not in any way help solve the ambiguity later, since thereafter further received symbols would yield identical metrics. Therefore, in the case of ties the sole survivor is picked at random.

To help in making a final decision as to which survivor should be chosen as the decoded path, the trellis is terminated by a tail of (K-1) known information symbols. In the tail, branching ceases, for only the branch corresponding to the known transmitted symbols is extended from each state. Therefore, the number of survivors is reduced by a factor of q by the comparison at each step. Consequently, after the (K-1) tail branches are received and decoded, there is only a single path left in the entire trellis: this path is accepted as the decoded path, and given the received sequence, it corresponds to the most likely transmitted sequence. No other path has a larger accumulated metric.

Compared to sequential decoding, the operations of the Viterbi decoder are far less complex, and the motion of the decoder is always forward with no backing-up. A decoding step involves only the determination of the branch metrics, the total accumulated metric and the q-wise comparison and proper selection. These operations are identical from level to level, and since they must be performed at every state, the complexity of the decoder is proportional to the number of states, hence grows exponentially with the constraint length. To decode a block of L information symbols the total number of Viterbi operations is $L q^{K-1}$, and for $L >> K$ it is far smaller than the total number $\frac{q}{q-1}(q^{L}-1)$ of operations that would be required to perform maximum likelihood decoding on a tree. However, as a practical decoding technique, clearly the exponential growth of the number of states with K will limit Viterbi decoding to convolutional codes of short constraint lengths ($K \leq 7$).

The probability of error for the optimal decoding of a convolutional code of rate R over a memoryless channel has been bounded by Viterbi (1967) as

$$P(E) < \frac{L(q\;1)}{1-q^{-\epsilon/R}} \; e^{-N E(R)} \tag{3.21}$$

where L is the number of information symbols that are encoded, $N = KV$ is the symbol constraint length of the code and where

$$E(R) = \begin{cases} R_{comp} = E_o(1) & 0 < R = R_{comp} - \epsilon < R_o \\ E_o(\rho) & R_o - \epsilon \leq R = \dfrac{E_o(\rho) - \epsilon}{\rho} < C \\ & 0 < \rho < 1 \end{cases} \tag{3.22}$$

$E_o(\rho)$ is the Gallager function given by Eq. (3.14), $\epsilon = (E_o(\rho) - \rho R) > 0$ and C is the channel capacity.

The bound shows that the error probability is exponentially decreasing with the constraint length of the code and can be made arbitrarily small by increasing N.

Using computer search techniques, good short constraint length codes of rates 1/2 and 1/3 have been obtained by Odenwalder (1970) for $K \leq 9$, and excellent decoding performance with these codes was reported (Heller & Jacobs, 1971). More recently Larsen (1973) extended the list of good codes with constraint lengths up to $K = 14$ and rates 1/2, 1/3 and 1/4.

(b) Practical Simplifications of the Algorithm

The great advantage of the Viterbi decoder is that the number of decoding operations performed per received branch is constant. These operations are always of the same nature and do not require a sophisticated logic. The main disadvantage is the huge memory necessary to store all the paths. Moreover, forcing the encoder into a known final state by using a tail of a known sequence of symbols is equivalent to waiting until all L information symbols have been decoded _before_ starting the delivery of the _first_ symbol to the user. In a real system this delay may be operationally undesirable.

It is observed that all the surviving paths do not stay distinct over their entire length but have a tendency to stem from a single node several constraint lengths earlier (Heller & Jacobs, 1971). For example in Fig. (3.7) the survivors at nodes $\underline{S}_o^{(\ell+2)}$, $\underline{S}_1^{(\ell+2)}$ and $\underline{S}_4^{(\ell+2)}$ merge together at node $\underline{S}_3^{(\ell)}$. Supposing that all the survivors at the present decoding depth n merge together at some node $\underline{S}_i^{(n-M)}$ lying M levels back, $M > K$, then regardless of which path is finally chosen as the decoded path, the surviving path leading to node $\underline{S}_i^{(n-M)}$ must be a part of it. Hence, it is not necessary to wait until the unique decoded path has been obtained before starting to deliver the decoded symbols. A final decision may be made on these branches prior to the point of common convergence, and the decoder can safely deliver the information (with a small delay corresponding to M) as it progresses in the trellis. With this procedure, clearly the decoder can eliminate the tail altogether.

Getting rid of the tail in this way leads to a very attractive situation with respect to the memory requirement. Clearly instead of storing the entire paths history for the total length L, the decoder needs to store only the history of the paths up to and including the point of common convergence. If the point of common convergence lies M levels back (M is called the memory of the decoder), then the total amount W of path storage required is

$$W = M\, q^{K-1} \tag{3.23}$$

A practical decoder has a given amount of storage available, so that in fact M becomes fixed. A practical refinement of the above method of symbol decision consists in deciding on the oldest symbol of the most likely of the q^{K-1} retained paths. It has been shown theoretically by Forney (1967) that with this procedure, asymptotically the expected value of the error probability on that oldest symbol is not increased if the memory M is large enough. It has been found through simulation (Odenwalder 1970, Heller & Jacobs 1971) that a memory of 4 or 5 constraint lengths is sufficient to insure a negligible degradation from the error probability of the optimum decoder. Should an error occur by this method it does not lead to catastrophy, for although the Viterbi decoder was assumed to start operating from a known starting state, it has been found through simulation (Heller & Jacobs 1971) that it may in fact start decoding from any state. The first 3 or 4 constraint lengths of data symbols may be unreliable but normal operations will prevail thereafter. Therefore, for a practical decoder the tail can be eliminated and the memory reduced without altering the performance of a (theoretical) optimum decoder.

Finally a quantized integer metric may be used instead of the exact log-likelihood function of Eq. (3.6) without much difference in error performance: the Viterbi decoder is relatively insensitive to metric quantization.

3.4.3 Comparisons and Limitations of Sequential and Viterbi Decoding

To summarize, the two decoding techniques presented above, attempt to find the shortest path through a graph. The Viterbi algorithm was shown to be an exhaustive search growing naturally out of the code topology whereas sequential decoding appears heuristically to be a natural method of reducing the average number of computations (but not the maximum) by trial and error search in the tree. Among the several sequential algorithms, the stack algorithm of Zigangirov and Jelinek is the simplest and most natural way to utilize the key concepts of path rejection of sequential decoding.

As discussed previously, the computational cutoff rate R_{comp} limits the rate at which sequential decoders can be used but the same Pareto distribution of the number of computations to decode one branch prevails regardless of the rate (Falconer 1967). Storage requirements and decoding speed must be traded in such a way that a decrease in buffer storage, must be compensated by an increase of the decoding speed over the bit rate, thus limiting the maximum bit rate capability. Below R_{comp}, if the number of computations that the decoder can perform per unit time is greater than the mean value of the corresponding Pareto distribution of computations, then on the average the decoder can keep-up with the data, although buffering is necessary. Regardless of the buffer storage provided and even for rates below R_{comp}, occasional long searches do occur resulting in a possible buffer overflow and the consequent erasing of long sequences. For sequential decoders using long constraint lengths the probability of undetected error is indeed very small, and usually the main contribution of errors comes from the buffer overflow.

For the Viterbi decoder such a situation obviously does not occur due to the fixed nature of the computational effort. Storage history and decoding speed requirements may be definitely set for a particular application.

Since both the number of operations per decoded branch and the total storage are proportional to the number of states, the complexity of the Viterbi decoder is proportional to the number of states, and therefore it increases exponentially with the constraint length of the code. Although the error probability decreases exponentially with the constraint length, clearly any improvement on the error performance by increasing K, is expensive in terms of computational effort and storage. Therefore, Viterbi decoders are limited to small ($K \leq 7$) constraint length codes, and consequently where very moderate error probability is sufficient. However, the very simple nature of a Viterbi operation and its identical repetition from level to level permits a parallel implementation of a decoder and hence decoding at very high data rates in the tens of Megabits/sec range.

Decoding delays are inescapable for both sequential and Viterbi decoders. For sequential decoders the delay is a direct consequence of the variability of the computational effort, and for Viterbi decoders it corresponds to the path memory storage of each state. Sequential decoders tend to have buffers longer than the path memory storage of Viterbi decoders. However, in a real-time situation where a fixed time delay is required for putting out data, the total number of computations required in one delay grows only linearly with the constraint length K for sequential decoders whereas it grows exponentially with K for Viterbi decoders.

In general, performance and complexity combine in such a way that sequential decoding is chosen over Viterbi decoding when high performance is required ($P(E) \leq 10^{-7}$), provided the information rate is sufficiently low that operation below R_{comp} is assured. Moreover, since sequential decoders will be extremely sensitive to bursty noise patterns, they will usually be restricted to well behaved memoryless channels such as the space channel.

Viterbi decoders may be preferred when the performance demanded is more modest but the data rate very high. The crossover point occurs for bit error probability in the 10^{-3} to 10^{-5} range and constraint length between 6 and 8.

Most of the drawbacks of sequential decoding are the consequences of the variability of the computational effort and improvements should be directed at reducing this variability without undue increase in decoder complexity. The shortcomings of sequential decoding could be alleviated by using some of the concepts of Viterbi decoding: exploitation of the remergers by using the trellis structure of the code, and improvement on the suboptimality of the decoding by the simultaneous exploration of the M, $M > 1$ currently most likely paths.

CHAPTER IV

MULTIPLE PATH STACK DECODING ALGORITHM

4.1 Introduction: Generalized Stack Algorithm

The two algorithms presented in the preceding chapter appear to be the extremes for determining the most likely transmitted sequence of a convolutional code, given the received sequence. The Viterbi algorithm examines all distinct paths at every level while a sequential decoding algorithm follows only that path that appears to be the most likely. The step by step procedure of sequential decoding tends to reduce to a minimum the average value of the number of computations to decode one information digit. Unfortunately, the procedure makes this computational effort variable with a Pareto distribution. By comparison, a Viterbi decoder carries along the one correct path and all incorrect paths of the trellis. As a consequence the computational effort becomes rather large but remains constant for the entire decoding.

We now turn our attention to improving the computational behaviour of sequential decoding by reducing the variability of the computational effort without degrading the error performance. This variability will be reduced when the distribution function $P(C \geq \ell)$ is itself reduced, mostly when ℓ becomes relatively large. The fundamental idea is to keep the search properties of sequential decoding and the notion of a stack while using some of the features of Viterbi decoding.

Denoting each explored path by the node of its extremity, we recall that a stack can be considered as a list of nodes ordered (in decreasing order) according to their metric values. Allowing for the extension of several paths from the stack we now generalize the concepts of the stack algorithm of Zigangirov and Jelinek.

Let a path extension cycle consist of computing the metrics of the successors of the extended paths, entering them in their proper place in the stack, and removing from the stack the nodes just extended. In a path extension cycle, the extension of the Jth node of the stack, $J \geq 1$, implies the simultaneous extension of all nodes stored higher in the stack. In other words if J paths are extended, they are the J most likely explored paths. The actual number of extended paths in a path extension cycle is assumed determined by a decision rule which is specified in the algorithm.

Following the path extension cycle a purging cycle eliminates from the stack some unwanted nodes and reorders the stack. The purging cycle is directed by some purging rule also specified by the user. The decision and purging rules need not be kept fixed over the entire decoding, but may be varied according to any information pertinent to the decoding as it progresses (past stack contents, metric behaviours, channel measurements, etc.).

The set of operations comprising a path extension cycle and a purging cycle is defined as a decoding cycle. An algorithm that uses this decoding cycle at every step is called a generalized stack algorithm.

It is now easy to show that both the Z-J algorithm and Viterbi algorithm may be viewed as two particular cases of this generalized stack algorithm.

In the Z-J algorithm the decoding cycle is degenerate since it contains no purging cycle. The decision rule is the simplest possible as it directs the extension of only the top node of the stack for each path extension cycle. (The quantized form of the Z-J algorithm introduces only a practical detail in the path extension cycle).

Now suppose the Viterbi decoder uses a stack to store and order all explored paths. For a binary code the decision rule is simply to extend the

$$M_v = 2^{K-1} \tag{4.1}$$

highest metric paths (the survivors at any depth) in the stack at each path extension cycle, whereas the purging rule is recognized as the elimination of all non-surviving paths. All redundancy is eliminated from the stack whose size remains constant. The M_v-path extension together with the purging rule clearly guarantees the optimality of the decoding and the constancy of the computational effort. Therefore, the Viterbi decoding algorithm is also a particular case of the generalized stack algorithm.

In conclusion, we see that depending on the particular decision and purging rules, the generalized stack algorithms allow a class of decoding algorithms among which the Zigangirov-Jelinek and Viterbi are only two members. In keeping with our objective of reducing the variability of the computational effort of sequential decoding, we may modify the Z-J algorithm in the following way:

(1) Use a purging cycle to exploit remergers and eliminate all redundancy from the stack.

(2) Expand on the decision rule and allow the extension of the M, $M \geq 1$ most likely paths.

4.2 Exploitation of the Remergers

By using the tree structure of the convolutional code, a sequential decoder totally ignores the fact that paths corresponding to the same encoder state and same tree level remerge. The redundant information imbedded in the tree is not used by the decoder. Consequently when the decoder backs-up in its search for a better path, it may follow a path that appears to be new, but because of remergers it may have been already explored. By adding to the Z-J algorithm a purging cycle whose purging rule exploits the reconvergence of the explored paths, some duplicated computations may thus be avoided. Before introducing the procedure used in the purging cycle we first examine the computational behaviour of the sequential decoder when explored paths remerge.

Let 2 paths $\underline{U} = (\underline{U}_\ell, \underline{u}_{\ell+1}, \underline{u}_{\ell+2}, \ldots)$ and $\underline{U}' = (\underline{U}_\ell, \underline{u}'_{\ell+1}, \underline{u}'_{\ell+2}, \ldots)$ emerge from some common node $\underline{U}_\ell$ at level ℓ, and let $\Gamma_{\ell+1}^{(\underline{U})} > \Gamma_{\ell+1}^{(\underline{U}')}$. Then a well known property of sequential decoding is that, given node $\underline{U}_\ell$ is reached, $\underline{U}'$ may be extended beyond level $(\ell+1)$ only if

$$\Gamma_{\ell+1}^{(\underline{U}')} \geq \operatorname*{Min}_{i} \; \Gamma_i^{(\underline{U})} , \quad i \geq \ell+1 \tag{4.2}$$

Conversely, if

$$\Gamma_{\ell+1}^{(\underline{U}')} < \operatorname*{Min}_{i} \Gamma_{i}^{(\underline{U})} , \quad i \geq \ell + 1 \tag{4.3}$$

then $\underline{U}'$ will not be extended by the decoder beyond $(\ell + 1)$.

In the absence of errors, the finally decoded path (i.e., the maximum likelihood path) is the correct path. The above property then shows that a necessary condition for extending an incorrect path issued from the correct path, is that its metric value must be larger than the minimum value, say Γ_{min} of the correct path metric beyond the point of common convergence. Hence, an incorrect path will be extended as long as it has a metric larger than Γ_{min}.

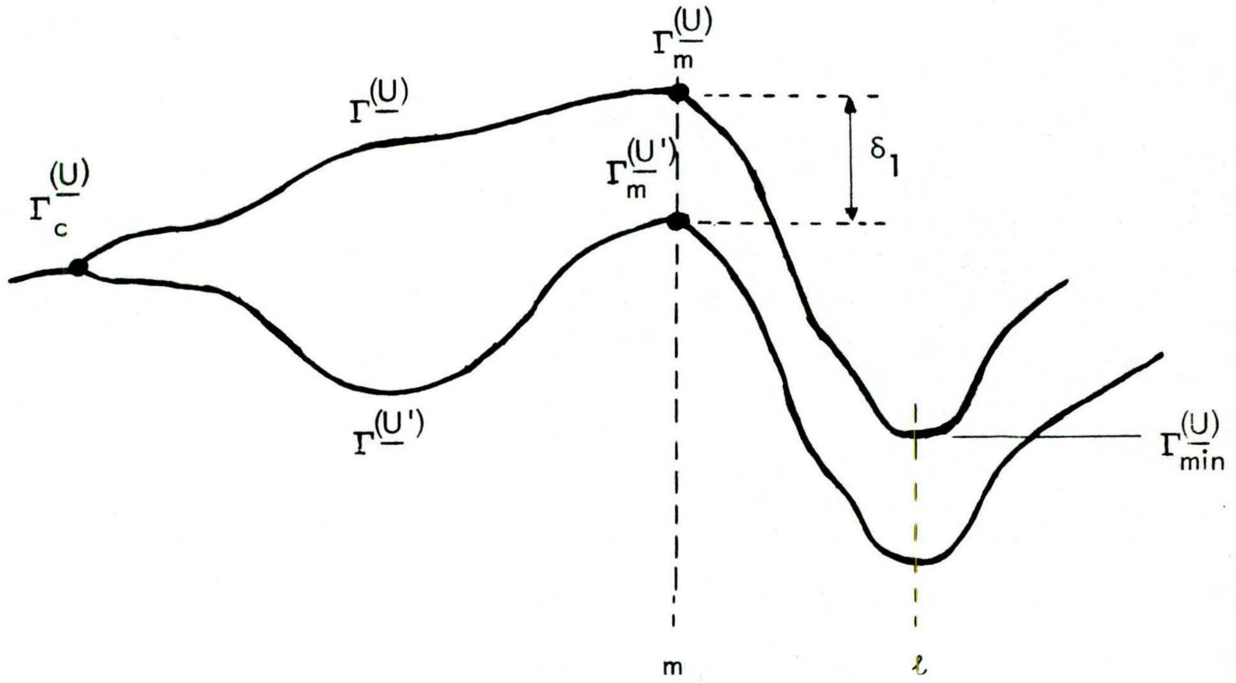
Consider the situation depicted in Fig. (4.1a) where 2 paths $\underline{U}$ and $\underline{U}'$ issued from node $\underline{U}_c$, remerge at level m with metrics $\Gamma_m^{(\underline{U})} > \Gamma_m^{(\underline{U}')}$. Let the subsequent minimum of the metrics of these paths occur at level ℓ, $\ell > m$, that is

$$\operatorname*{Min}_{i} \Gamma_{i}^{(\underline{U})} = \Gamma_{\ell}^{(\underline{U})} = \Gamma_{min}^{(\underline{U})} , \quad \ell > m \tag{4.3}$$

and let

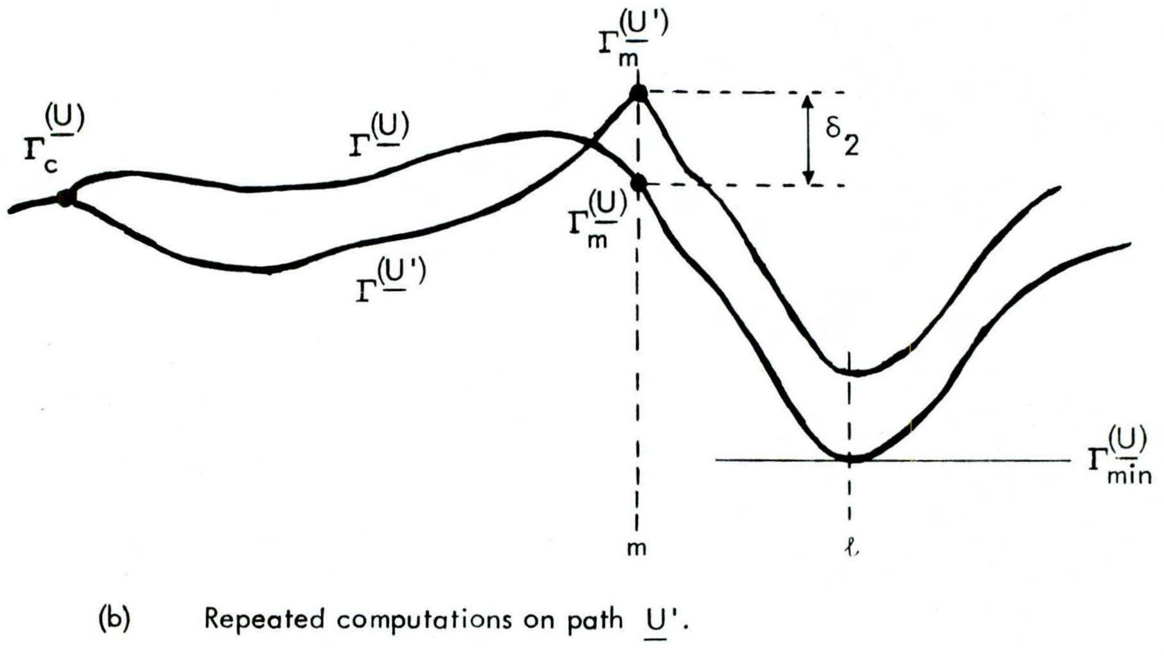
$$\Gamma_{i}^{(\underline{U}')} > \Gamma_{min}^{(\underline{U})} , \quad c \leq i < m \tag{4.4}$$

Assume that node $\underline{U}'_m$ is reached by the decoder before node $\underline{U}_\ell$ but after node $\underline{U}_m$. The decoder will extend both paths $\underline{U}$ and $\underline{U}'$ beyond level m, and path $\underline{U}'$ will be extended as long as its metric value does not fall below



$\Gamma_m^{(\underline{U})}$
$\Gamma^{(\underline{U})}$
$\Gamma_c^{(\underline{U})}$
$\Gamma_m^{(\underline{U}')}$
δ_1
$\Gamma^{(\underline{U}')}$
$\Gamma_{min}^{(\underline{U})}$
m
ℓ

(a) Eliminated computations on path $\underline{U}'$.



$\Gamma_m^{(\underline{U}')}$
$\Gamma_c^{(\underline{U})}$
$\Gamma^{(\underline{U})}$
δ_2
$\Gamma_m^{(\underline{U})}$
$\Gamma^{(\underline{U}')}$
$\Gamma_{min}^{(\underline{U})}$
m
ℓ

(b) Repeated computations on path $\underline{U}'$.

Figure (4.1) Repeated and eliminated computations on remerging paths.

$\Gamma_{\min}^{(\underline{U})}$. Since paths $\underline{U}$ and $\underline{U}'$ converge at level m, their metric increments are identical beyond level m. Therefore beyond level m the cumulative metrics of paths $\underline{U}$ and $\underline{U}'$ will always differ by the constant amount

$$\delta_1 = \Gamma_m^{(\underline{U})} - \Gamma_m^{(\underline{U}')} \qquad (4.5)$$

Consequently any exploration of path $\underline{U}'$ beyond level m is bound to fail. The unnecessary duplicated computations on path $\underline{U}'$ can be avoided by a generalization of the Z-J stack decoder that recognizes the convergence at level m, and eliminates, in its purging cycle, the redundant node $\underline{U}'_m$. Clearly, the elimination of $\underline{U}'_m$ from the stack eliminates with it the potential subtree of redundant paths issued from $\underline{U}'_m$ which can be explored. The same reasoning applies to all non-surviving paths that may converge on node $\underline{U}_m$ with a smaller metric than $\Gamma_m^{(\underline{U})}$

Consider now the case where $\underline{U}'_m$ converges on $\underline{U}_m$ with a larger metric, $\Gamma_m^{(\underline{U}')} > \Gamma_m^{(\underline{U})}$ (see Fig. (4.1b)). Relations (4.3) and (4.4) still hold and it is assumed that node $\underline{U}'_m$ is reached by the decoder after node $\underline{U}_m$. Now the metric values $\Gamma_i^{(\underline{U}')}$, $i \geq m$, are increased by the constant amount

$$\delta_2 = \Gamma_m^{(\underline{U}')} - \Gamma_m^{(\underline{U})} \qquad (4.6)$$

Clearly, any previous exploration of path $\underline{U}$ beyond level m is entirely wasted and must be repeated, because there is no way for the decoder to increase by δ_2 all the metric values $\Gamma_i^{(\underline{U})}$, $i \geq m$, except by the normal path extension procedure of the algorithm. Applying this reasoning to all the paths issued from node $\underline{U}_m$, in general the subtree of explored paths issued from $\underline{U}_m$ must be explored anew, but now the repetition yields a larger metric. In that case the introduction of a purging cycle reduces only the stack size of a Z-J decoder without affecting the computational effort. Summarizing we have the following theorem.

Theorem 4.1

Let $\{\underline{U}_m\}$ and $\{\underline{U}'_m\}$ be the subset of explored paths that have emerged from two converging nodes $\underline{U}_m$ and $\underline{U}'_m$, and let $\underline{U}_m$ be reached by the sequential decoder before $\underline{U}'_m$.

(a) If $\Gamma_m^{(\underline{U}')} \leq \Gamma_m^{(\underline{U})}$, then the subset of paths $\{\underline{U}'_m\}$ constitute fruitless redundant computations.

(b) If $\Gamma_m^{(\underline{U}')} > \Gamma_m^{(\underline{U})}$, then the subset of paths $\{\underline{U}_m\}$ are necessary repeated computations.

We have shown that by ignoring remergers, any sequential decoder performs useless computations and the Z-J increases unnecessarily the size of its stack. The addition of a purging cycle (whose purging rule exploits the reconvergence of the paths) to the Z-J algorithm will eliminate all redundant storage and fruitless computation. Clearly, exploitation of condition (a) above, is equivalent to sequential decoding on the trellis structure of the code, whereas condition (b) leaves the decoder on the tree structure of the code. By adding the purging cycle, the decoding of a node on the finally chosen path may require fewer (but never more) computations, and hence both the average value and variability of the Z-J algorithm may thus be reduced. Naturally, in practice these improvements must be weighed against the cost (in added complexity and storage of the decoder) of providing for the purging cycle. The purging cycle must therefore be the simplest possible.

4.2.1 Z-J Algorithm With a Purging Cycle

In order to eliminate the redundant computation and storage while keeping the search properties of sequential decoding, the Z-J algorithm has been modified to include a purging cycle in the following way: whenever two paths converge, the convergence is recognized, the two accumulated metrics are compared and only the path yielding the highest metric is stored in the stack. A new branch extension converging onto a node already in the stack is discarded if it does not increase the total metric at that node. However, if the total metric at the converging node is increased, then the information of the newly converging branch replaces the information of the node previously stored in the stack with the lower metric. With this procedure the purging cycle is conveniently imbedded in the path extension cycle. However it assures that all redundancy is removed from the stack since of all the paths converging at a node, only the best one is retained.

Looking at the flow diagram of the new algorithm depicted in Fig. (4.2), we see that after the top node is extended, a convergence test is performed in the box "CONVERGENCE?" to check whether the end node of each new branch extension has ever been visited before. If the end node is a new node, it is stored in the stack in the proper place defined by its total accumulated metric value. Everything is exactly like the Z-J algorithm. On the other hand, if a convergence is observed, the accumulated metrics of the two converging paths must be compared in order to choose the survivor. The comparison is performed at the box labelled "IMPROVEMENT?". If the comparison test is favorable to the new branch extension, the new information is entered in the stack location determined by the convergence test. But since the metric value for the node is now increased, its ordering in the stack may have to be modified. This substitution of node information and stack reordering takes place in the box labelled "MODIFY STACK". From that point on, decoding follows the Z-J algorithm.

Finally, if the comparison test "IMPROVEMENT?" is not favorable to the new branch extension, this new extension is simply discarded, in effect purging the stack of one useless location and decoding is resumed. Clearly, the elimination of the end node of a new branch extension eliminates with it all possible successors that may be explored, at a saving of both computation and stack storage.

This new algorithm retains all the features of the Z-J algorithm but in addition tends to exploit the trellis structure of the code by keeping only the

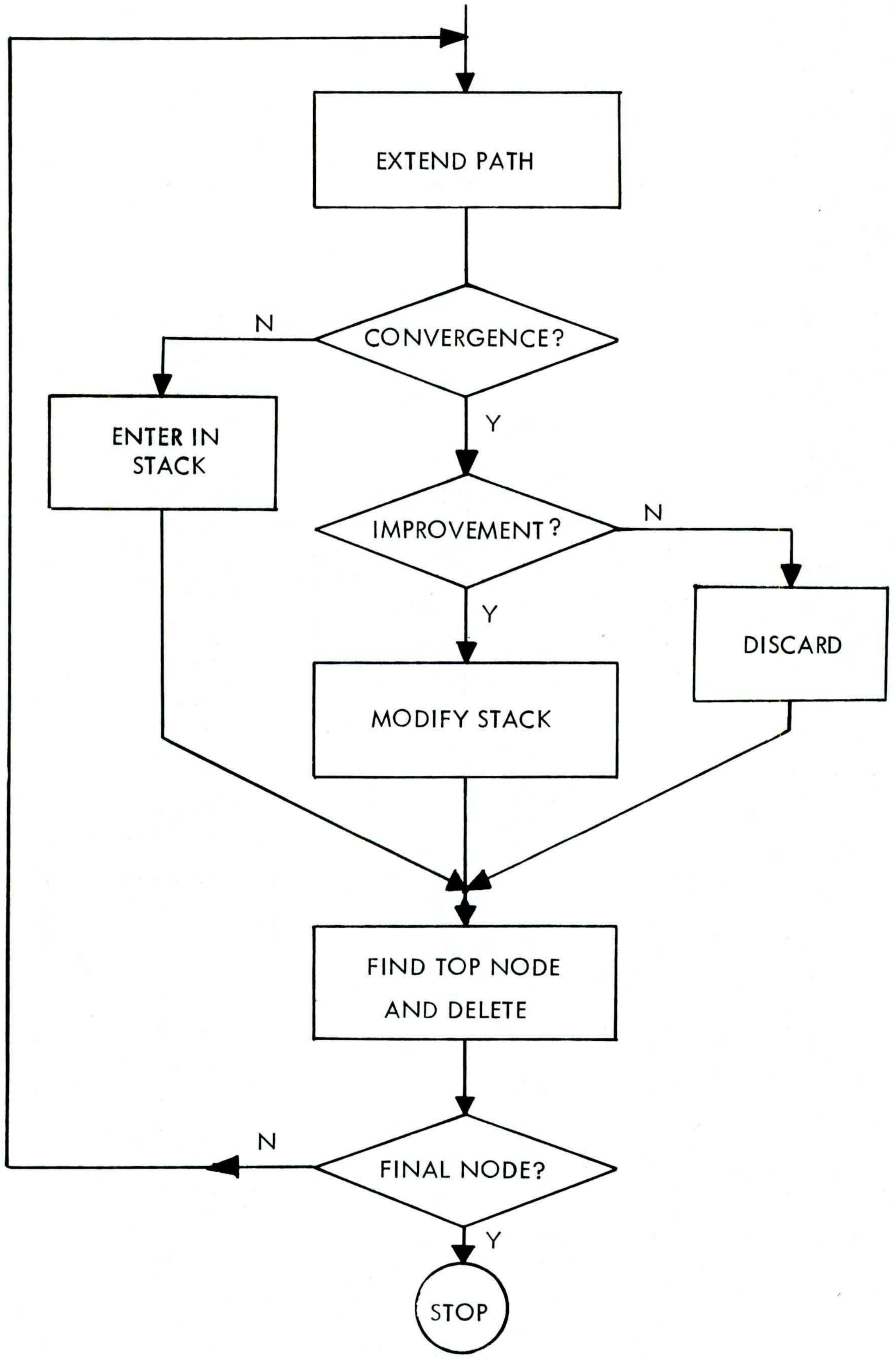


Figure (4.2) Flow diagram of a Z-J algorithm exploiting the paths convergences.

best path (the survivor) at each visited node. Hence, it may be regarded as a single-path generalized stack algorithm. Unfortunately, because of the single path extension cycle, the purging cycle removes the stack storage redundancy but not the entire computation redundancy.

Unlike Viterbi decoding, the survivor at a node is chosen from among the subset of explored paths, not all converging paths. Consequently, the survivor chosen by the decoder is not necessarily a true survivor. If a node in the stack corresponds to a true survivor, the new algorithm will never explore identical paths issued from that node. However for all the nodes in the stack, until the true survivor is found, identical computations may have to be repeated.

Another fundamental difference with Viterbi decoding is that since only the most likely path is extended, not all distinct states are visited at each level. Consequently during a dip of the correct path metric, back-up searches may be necessary, and hence the random nature of the computation still persists with an asymptotic Paretean distribution. Because the amount of computations is reduced while the probability of having a metric dip remains unchanged, intuitively we see that the Pareto exponent could increase. The improvement will depend on the occurrences of remergers, but will be small because it will be shown in Section 4.3.4 that the ensemble average probability of converging with the correct path decreases exponentially with the constraint length K.

4.2.2 Purging Procedure

To implement the above algorithm, some simple way must be found to determine whether a new branch extension terminates in a node already stored in the stack, and if so, to get all the data about that node. Using a label for each node stored in the stack would be acceptable if the concomitant search for convergence involved only a small segment of the stack. If a brute force scanning of the entire stack is required before each new-entry, then clearly the whole procedure would be worthless.

Before describing the method used we first present briefly the storage operation of the Z-J algorithm. A more detailed description of the algorithm as it was programmed is given in Appendix I. With each node stored in the stack, enough information must be stored along to allow for its proper ordering in the stack, and its possible extension should decoding resume from that node. In addition information about the predecessor node must be saved, so that once the entire path is decoded it may be easily recovered from the final node back to the origin. Therefore an entry in the stack consists of the following information stored in contiguous words of computer memory.

(1) The total accumulated metric (labelled "VALUE") needed for the proper ordering of the stack.

(2) The encoder state (labelled "STATE") to produce the code symbols on the branches issued from the extended node.

(3) The node depth level (labelled "DEPTH") to compare the code symbols of the extended branches with the corresponding symbols of the received branch.

(4) A back pointer (labelled "PATH POINTER") to recover the information sequence when the final node is reached.

(5) Another pointer (labelled "STACK POINTER") needed to help determine the top node of the stack.

Returning to the convergence problem, we recall that two branches will converge if their end nodes lie at the same depth and correspond to the same encoder state. Therefore, for two nodes lying at the same depth, a convergence test reduces to a mere comparison between their states. The state and depth of the end node of a new branch extension being obviously known, checking in the stack the state of every node having the same depth should not be too time consuming, because the number of nodes explored at any level, is in general far smaller than the number of distinct states.

In order to exploit this observation, all nodes stored in the stack and lying at the same depth, are linked together by a set of pointers labelled "LINK". Therefore to the original five registers that constitute an entry in the stack, we add the LINK register in which we store the address of the node previously stored in the stack at the same depth. If a node is the first one to be stored at that depth, the LINK contains zero. The LINK register is of course contiguous to the other registers.

Given the node level or depth of a new branch extension, the search for convergence is accomplished by the set of LINK pointers and an additional auxilliary array labelled "NODE POINTER". This is an array of length equal to the length in branches of the trellis, and whose first word contains the address of the last new node of level 1 stored in the stack, the second word contains the address of the last new node of level 2 stored in the stack, and so on. That is, the beginning of the chain of LINK pointers at any given depth resides in the NODE POINTER word specified by that depth. For penetrations not yet reached by the decoder, the corresponding NODE INDEX entry is zero. With this data structure, we see that the addresses of all the explored nodes having a particular depth and stored in the stack, can be retrieved by following the chain of LINK pointers whose starting point is contained in the NODE POINTER word specified by that depth. Having the address of a node in the stack, the VALUE, STATE, etc., information is immediately available, and the test for convergence and metric comparison readily performed. Details of the procedure is given in Appendix 1.

4.2.3 Speeding Up the Procedure

The convergence test must be carried out for each branch extended by the decoder, and although quite small, the time spent in searching for convergences should be kept to a minimum. In order to speed up the procedure we now show that due to the following property of the trellis, repeating the search for each and every branch issued from the same node is unnecessary. Regardless of the number of successors to a node, only one search for convergence is sufficient for all of them.

<u>Property 4.1</u>

If n nodes, $n \geq 2$ converge in one of their branch extensions, they converge in all q^d of them.

<u>Proof:</u>

Suppose n nodes $\underline{U}_i^{(1)}, \underline{U}_i^{(2)}, \ldots \underline{U}_i^{(n)}$, $n \geq 2$, lying at the same depth i converge at some node $\underline{U}_{i+1}^{(*)}$ in one of their q^d branch extensions (see Fig. (4.3)). Recalling that converging paths have identical data over their last (k-1) branches, the paths ending in nodes $\underline{U}_i^{(1)}, \underline{U}_i^{(2)}, \ldots \underline{U}_i^{(n)}$ have identical data over their last (k-2) branches, and the n converging branches carry the same d information symbols. Hence each of the q^d successors of the n nodes $\underline{U}_i^{(1)}, \underline{U}_i^{(2)}, \ldots \underline{U}_i^{(n)}$ having the same d information symbols will converge n-wise at a single node, and there are q^d such nodes.

<u>Conversely:</u>

If a pair of branches issued from two distinct nodes do not converge, these two nodes do not converge in any of their q^d branch extensions.

The above property and its converse show that whenever a branch extended from some node is stored in the stack as a new entry (no convergence), all other branches emerging from that nodes are also stored as new entires. Consequently they are linked together in a continuous way by the LINK pointers, the last entry pointing towards the location of the preceding entry, the latter

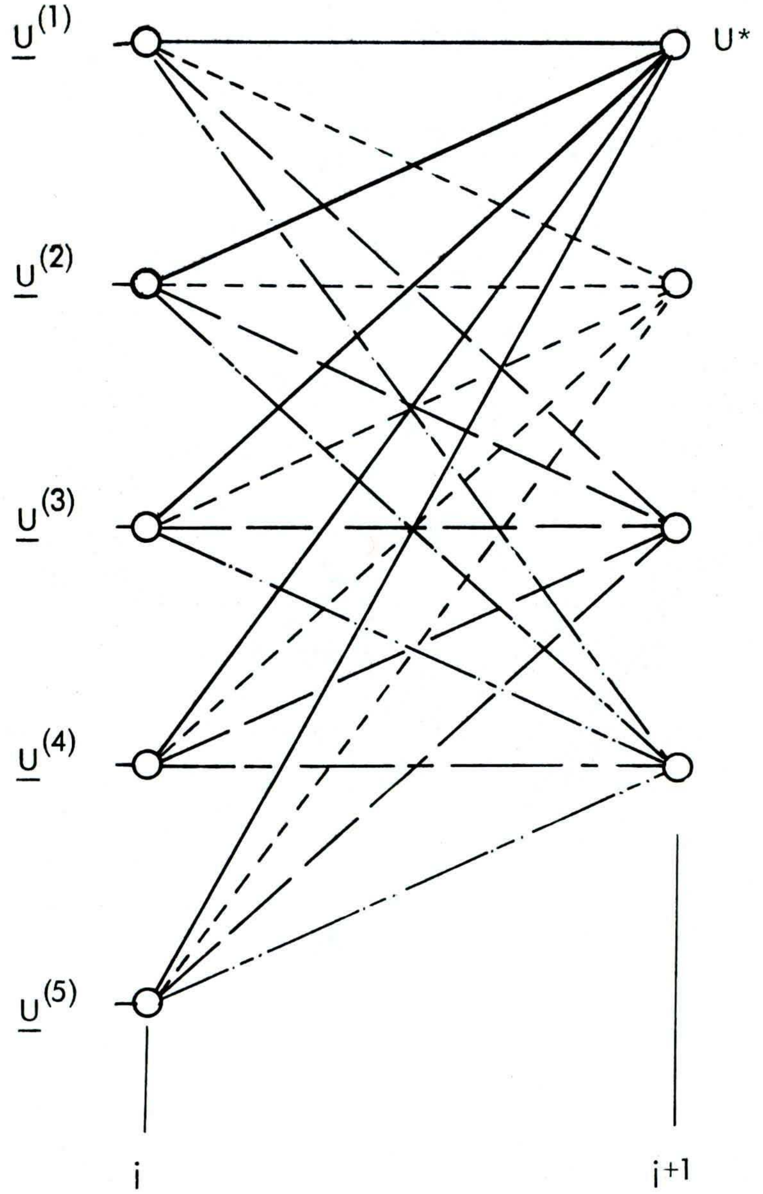


Figure (4.3) Illustration of property (4.1) for n = 5, q = d = 2.

towards its own preceding one, and so on. It follows that the search for convergence for all the q^d extensions of a node reduces to searching for the convergence of only that branch stored the last among the q^d. If there is convergence for that branch, there is also convergence for all other branches emerging from the same node whose locations in the stack are immediately obtained by the LINK pointers without searching. However, if there is no convergence for the last branch, there is no point in scanning the stack for the other branch extensions.

4.3 Z-J Algorithm With a Multiple Path Extension Cycle

We have shown how the addition of a purging cycle to the Z-J algorithm eliminates useless computations and hence help reduces the variability of the decoding effort. However, regarding the remerging of an incorrect path with the correct path as being a possible error event for sequential decoding, as the constraint length of the code becomes large, the effect of the purging cycle becomes minimal.

Keeping with our goal of reducing the variability of the decoding effort, we now examine the other main difference between the Z-J and Viterbi algorithms: the multiple path extension cycle. The introduction of an M-path extension cycle would close in some sense the gap between these two seemingly different decoding algorithms. In this section we first review some of the search properties of sequential decoding and then proceed to add a multiple path extension cycle to the Z-J algorithm.

4.3.1 Search Properties of Sequential Decoding

The decoded path is the path ultimately chosen by the decoder, and in the absence of errors this decoded path coincides with the correct path. We recall that in finding this decoded path the sequential decoder uses a biased metric in such a way that on the average the metric increases on the correct path and decreases on all incorrect paths. A typical correct path metric is shown in Fig. (3.5), and to be decoded some of its node will require a single computation, and some other nodes will require several computations.

Let $\underline{U}_j$, $1 \le j \le L$ be a node lying at level j on path $\underline{U}$ such that

$$\Gamma_j^{(\underline{U})} \le \Gamma_i^{(\underline{U})} \qquad j \le i \le L \tag{4.7}$$

Then $\underline{U}_j$ is called a breakout node (Gallager 1968). If (4.5) is a strict inequality, then $\underline{U}_j$ is called a strict breakout node. A node that is not a breakout node is called a non-breakout node. Fig. (4.4) shows a segment of a correct path metric where the breakout and non-breakout nodes are indicated. From the definition of a breakout node and with the help of Fig. (4.4) the following properties are obvious:

Property 4.2

A node on the correct path is a breakout node only if its correct branch metric increment is non-negative. If this increment is negative the node is non-breakout.

Property 4.3

Along any path, the metrics of all non-breakout nodes located between two consecutive non-adjacent breakout nodes are not smaller than the metrics of these two breakout nodes.

From these properties we see that a segment of the correct path with a non-decreasing accumulated metric does not necessarily correspond to a set of breakout nodes. In fact a node on a path can be labelled "breakout" with certainty only after observing the metric of this path in its entirety.

Considering a Z-J algorithm, suppose a strict breakout node $\underline{U}_i$ on a path $\underline{U}$ reaches the top of the stack. Then for any other node $\underline{V}_n$ stored in the stack we have

$$\Gamma_i^{(\underline{U})} \geq \Gamma_n^{(\underline{V})} \tag{4.8}$$

Because $\underline{U}_i$ is a strict breakout node, then regardless of whether or not its immediate successor $\underline{U}_{i+1}$ is a breakout node we have

$$\Gamma_{i+1}^{(\underline{U})} > \Gamma_n^{(\underline{V})} \tag{4.9}$$

If $\underline{U}_{i+1}$ is ever extended then its own successors $\underline{U}_{i+2}$ will also have a larger metric than $\Gamma_n^{(\underline{V})}$, and so on until the final node is reached. Therefore, $\underline{V}_n$ will never reach the top of the stack and cannot be extended. Moreover node $\underline{U}_i$ will belong to the decoded path.

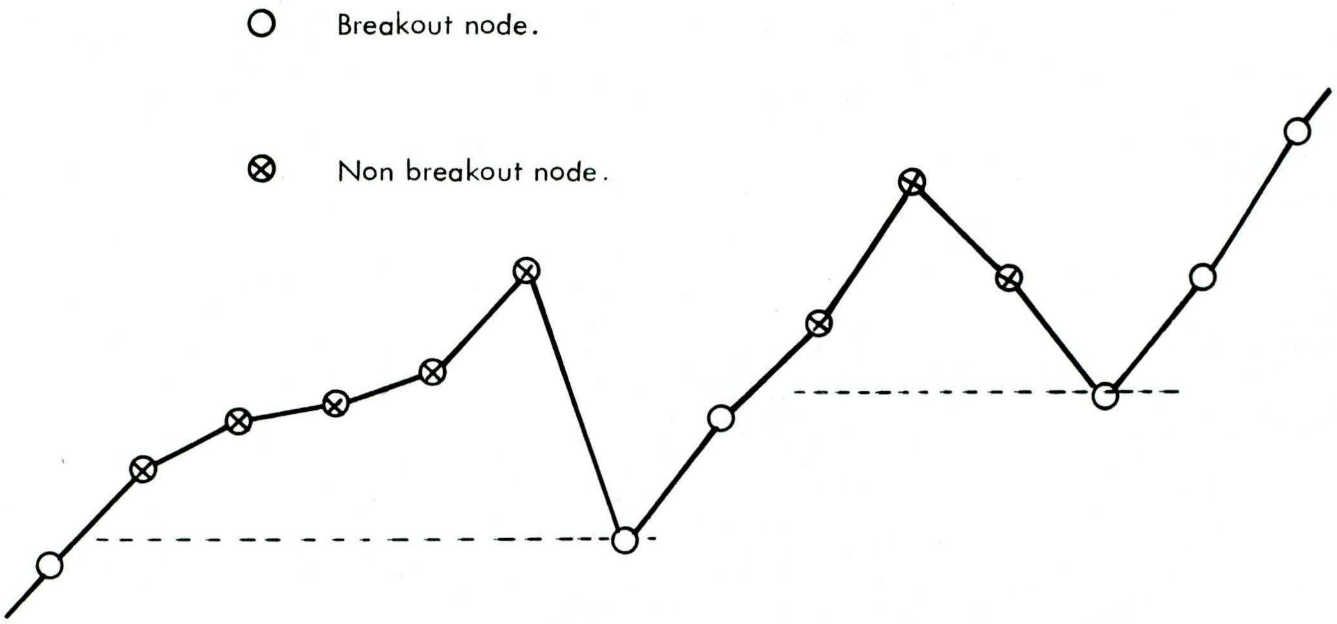


Figure (4.4) Correct path metric

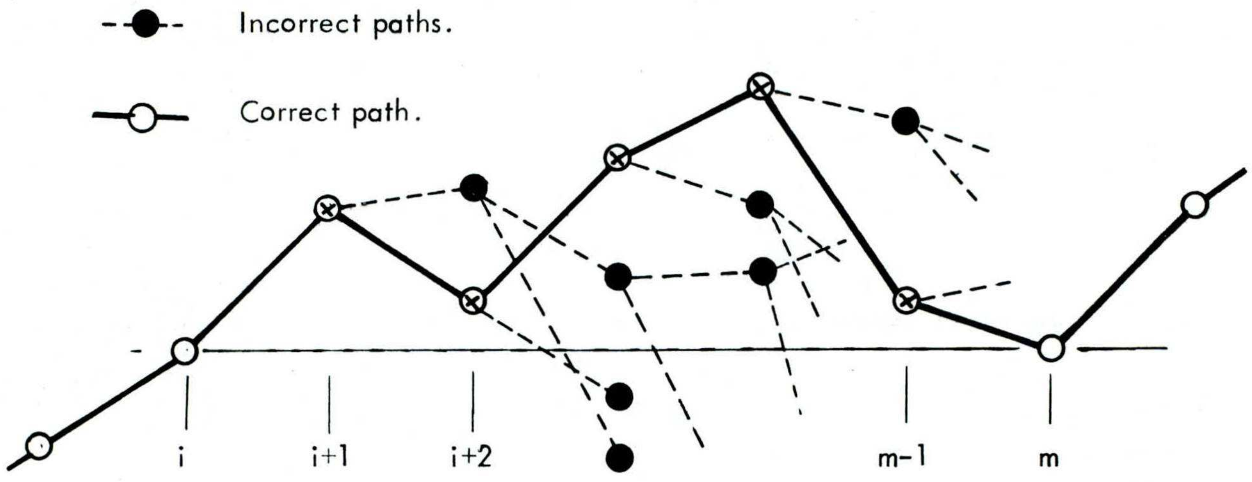


Figure (4.5) Correct and incorrect path metrics.

Assume now that of all the branches emerging from a node, at most only one can have a positive branch metric. Then of all the branches emerging from $\underline{U}_i$ only the one leading to node $\underline{U}_{i+1}$ will yield a larger accumulated metric than $\Gamma_i^{(\underline{U})}$. All other branch extensions will yield smaller metrics than $\Gamma_i^{(\underline{U})}$ and hence will never be further extended by the algorithm. Consequently, node $\underline{U}_{i+1}$ will also be on the decoded path and is decoded at a cost of a single computation. If node $\underline{U}_{i+1}$ is itself a strict breakout node, the same situation is repeated and clearly decoding consecutive adjacent breakout nodes of the decoded path requires a single computation.

Now let node $\underline{U}_{i+1}$ be a non-breakout node and suppose $\underline{U}_m$, $m > i + 1$ is the next breakout node on path $\underline{U}$. (See Fig. 4.5). Then

$$\Gamma_{i+1}^{(\underline{U})} \geq \Gamma_m^{(\underline{U})} > \Gamma_i^{(\underline{U})} \tag{4.10}$$

Assuming that $\underline{U}_m$ will eventually be extended, all descendants of $\underline{U}_{i+1}$ whose total metric is greater than $\Gamma_m^{(\underline{U})}$ will reach the top of the stack and will be extended before breakout node $\underline{U}_m$. The situation is repeated for all non-breakout nodes $\underline{U}_j$, $i < j < m$, and the decoder will extend all their descendants as long as their metric is larger than $\Gamma_m^{(\underline{U})}$. Clearly then, the only incorrect paths ever extended by the decoder emerge from previously extended non-breakout nodes on the decoded path, and these non-breakout nodes are always deeper in the tree than the last decoded breakout node. We have therefore the properties.

Property 4.4

The only incorrect paths the decoder will ever explore before extending a given node on the decoded path emerge from non-breakout nodes on the decoded path. These non-breakout nodes are located between the last decoded breakout node and that given node.

Property 4.5

A sequential decoder never backs-up beyond the level of the last decoded breakout node.

These properties were established for the decoded path in general, hence they also hold for the correct path in the absence of decoding errors. Any discrepancy between correct and decoded path corresponds to decoding errors, and in general the number of errors is small. Therefore unless otherwise specified, there is no loss of generality in considering the correct path as being the decoded path.

The above discussion shows that decoding proceeds smoothly when the decoder moves along breakout nodes on the correct path. Each decoded breakout node sets a limit to the depth of any future back-search. The decoding effort increases and becomes variable only when the decoder enters a series of non-breakout nodes on the correct path. Each of these non-breakout nodes becomes the root node of a potential subtree of incorrect paths that must be explored as long as their metric value of the next breakout node, say $\underline{U}_m$. Only after this exhaustive exploration of incorrect paths has been performed can breakout node $\underline{U}_m$ be extended and fast decoding resume with little effort. Any increase of the number of consecutive non-breakout nodes on the correct path and increase of the correct path metric dip, will increase the number of incorrect paths that must be explored, thus increasing the decoding effort. The distribution and average value of the number of computations necessary to decode one branch will depend on the distances (in number of branches) between consecutive breakout nodes on the correct path.

Using the concept of a breakout node, a Markov Chain model of the metric differences Δ_k (or dip values) on the correct path,

$$\Delta_k = (\Gamma_k - \min_{j \geq k} \Gamma_j) \tag{4.11}$$

has been developped by Massey and others (1969, 1972). In this representation the states of the chain are the possible dip values, including zero. We show in Appendix II that the average distance d_o between consecutive breakout nodes on the correct path is the reciprocal of the stationary probability of state zero. Clearly, any procedure that tends to reduce this average distance d_o, will convert in effect non-breakout nodes into breakout nodes. As these new breakout nodes set new limits to the back-searchings, depending on the procedure used, compared to sequential decoding, the resulting distribution of the computation may thus be reduced.

4.3.2 Multiple Path Extension

In order to reduce the variability of the decoding effort of the Z-J algorithm we extend the path extension cycle of the algorithm and allow the correct path to be extended before it reaches the top of the stack.

Suppose some node $\underline{U}_i$ on the correct path has just been extended by the Z-J algorithm, and let its correct path extension $\underline{U}_{i+1}$ be stored in the stack at the Mth position from the top. Thus there are in the stack (M-1) incorrect nodes with a metric not smaller than $\Gamma_{i+1}^{(U)}$. By the rules of the algorithm, node $\underline{U}_{i+1}$ will move up to the (M-1)th position in the stack only if none of the successors of the top node has a metric larger than $\Gamma_{i+1}^{(U)}$. Following each extension, the relative position of node $\underline{U}_{i+1}$ can vary anywhere from one place closer to the top, to (q^d-1) places farther away. Clearly the absolute minimum number of computations necessary to extend $\underline{U}_{i+1}$ is M whereas the maximum is unbounded if the tree is infinite. Suppose now the decision rule allows the extension of the M highest nodes in the stack for each path extension cycle. Then node $\underline{U}_{i+1}$ would be extended with certainty at a cost of exactly M computations. If the correct path were always included in each path extension cycle, obviously there would be no back searching by the decoder and the computational effort would be fixed at M computations per decoded branch. Since the correct path would never cease to be extended, in that respect this M-path generalized stack algorithm would be similar to Viterbi decoding. Naturally, when the channel is quiet and the decoder is moving along breakout nodes, the correct path is at the top of the stack and hence ($M-1$) computations are performed needlessly at every decoding cycle. Although speeding the extension of the correct path in general, compared to the Z-J algorithm, this new decision rule will entail a larger average decoding effort.

When the channel gets noisier and the correct path metric drops, the correct path may be located anywhere in the stack, and if M is not large enough, only incorrect paths will be extended for some decoding cycles. However the extension of the correct path will be resumed as soon as it reaches at least the Mth position in the stack.

By including the purging cycle described in Section 4.3 to take advantage of the remerging of the paths, it is clear that for $M = M_v$ this M-path generalized stack decoder becomes the optimum Viterbi decoder with a constant computational effort. However, for any smaller values of M, back searches for the correct path are possible and therefore a computational behaviour similar to that of sequential decoding is expected for periods of deep searches.

Since the average number of computations to decode one branch is at least equal to M, for applications where sequential decoding is suitable, one would choose $M < M_v$ to keep the average computational effort small, and accept some variation of the actual decoding effort. Clearly a trade-off exists between the variability of the computational effort and its average value.

In the class of generalized stack algorithms using a constant M-path extension cycle and a purging rule that exploits the remerging of the paths, the single path Z-J and Viterbi decoding algorithms may be considered the two extremes. One algorithm yields a variable but on the average small decoding effort while for the other the decoding effort is rather large but constant. The M-path stack decoder stands between these two extremes and exchanges a reduction of the variability of the computational effort for an increase of its average value. Depending on M, the behaviour of this decoder is closer to either the Z-J or the Viterbi decoder. An M-path stack decoder can be attractive in those applications where it may be preferable to have smaller variations of the computational effort than sequential decoding even if this implies a larger average number of computations per decoded digit.

4.3.3 An M-path Generalized Stack Algorithm

The M-path generalized stack algorithm may be looked on as the parallel operation of M single path interdependent Z-J algorithms working on the top, second, ..., Mth highest nodes of a unique stack. Since in general $M << M_v$, the biasing of the metric is kept in order to reduce back searches to a minimum. The redundancy in the stack is eliminated by a purging cycle that exploits the reconvergence of the paths as described in Section 4.3. The flow diagram of the generalized algorithm is given in Fig. (4.6) and the operations are summarized below:

(0) Initialization.

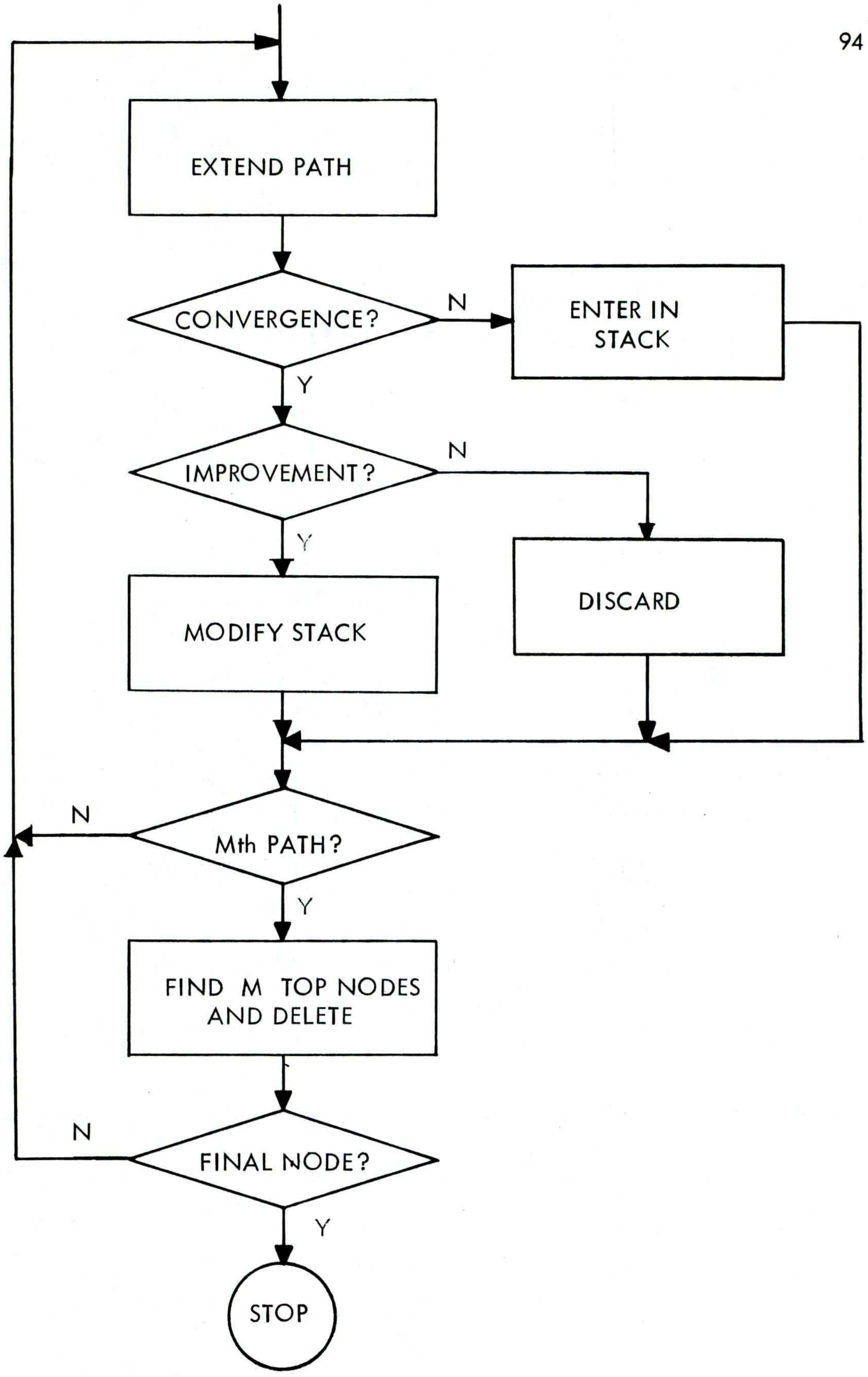


Figure (4.6) Flow diagram of the M-path algorithm.

(1) Extend the M top nodes of the stack and delete them from the stack.

(2) Test and exploit the convergences. Enter the new nodes in their proper place in the stack.

(3) Find the top M nodes of the stack. If the top node is the final node, stop. Otherwise go to (1).

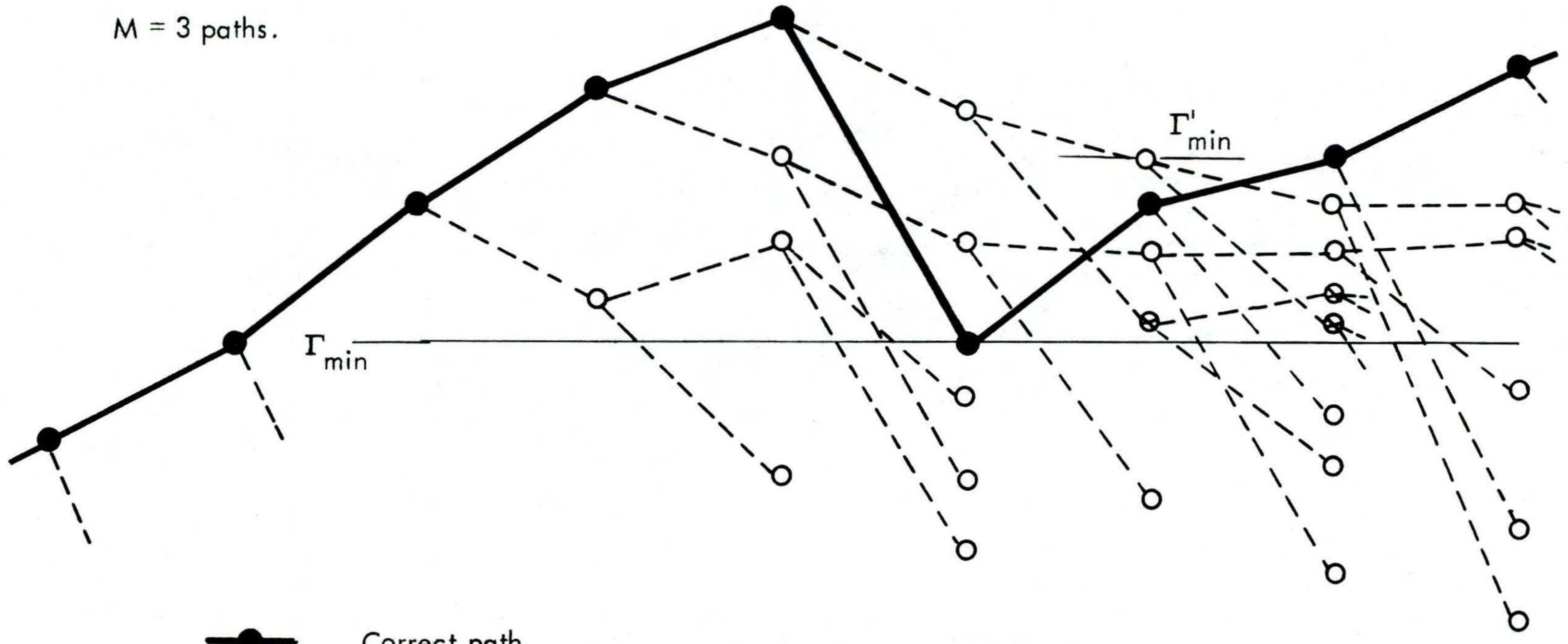
In this algorithm we have conveniently imbedded the purging cycle into the path extension cycle, just like the modified Z-J algorithm of Section 4.3. Hence this M-path generalized stack decoding algorithm is very similar to the single path Z-J algorithm. The main modification consists in finding in sequence the M highest metric nodes in the stack (and deleting them) before proceeding to their extension. As usual a computation is counted each time a node is extended into its successors. Consequently, the execution of step 1 of the M-path algorithm involves M computations.

4.3.4 Computational Behaviour of the M-Path Algorithm

The M-path generalized stack decoding algorithm (which we call the "M-path algorithm") retains the essential features of the single paths Z-J algorithm. For $M < M_v$, occasionally the correct path is not extended at every decoding cycle, and hence the number of computations necessary to decode one branch remains a random variable.

As long as the decoding of a node on the decoded path requires fewer computations with the M-path algorithm than with the single path Z-J algorithm, the computational effort is improved. However since breakout nodes on the correct path require a single computation for their decoding, no improvement is possible for those nodes.

When the correct path metric drops but the correct path is yet extended at every decoding cycle, then for all purposes this metric dip may be considered non-existent since it caused no back-searching. The corresponding non-breakout nodes behave as if they were breakout nodes, and hence the effective average distance between breakout nodes is reduced, reducing with it the variability of the decoding effort. The most interesting situation occurs when the first new breakout node on the correct path is extended while being at the Mth position in the stack. Ignoring the remergers, after the correct path is extended past the bottom of the dip, its metric increases while it generally decreases for the incorrect paths. Hence the relative position of the correct path in the stack improves. Consequently some of the incorrect paths that would otherwise be explored by the single path Z-J algorithm before the bottom of the dip is reached will not be extended by the M-path algorithm. This situation is illustrated in Fig. (4.7) where several incorrect nodes are not extended although they qualify for extension by a sequential decoder. Clearly with the usual sequential decoding, the entire subset of incorrect paths of interest must be exhausted before the bottom of the correct path metric may be reached whereas with the M-path algorithm, the nodes



━●━ Correct path.

--○-- Incorrect path.

⊗ Node possibly discarded by the 3-path decoder.

Figure (4.7) A small dip ignored by a 3-path decoder.

on the incorrect subset tend to be explored simultaneously, level by level. As shown in Fig. (4.7) this exploration may not be exhaustive. Such a parallel rather than serial exploration of the incorrect paths is in part responsible for the improvement of the distribution of the computations. The simultaneous multiple path extension may be seen as a method whereby, upon decoding a node on the correct path, the total number of nodes in the entire subset of incorrect paths is exchanged with that (smaller) number of incorrect nodes actually having a larger metric than the correct node. Consequently except for large correct path metric dips, the maximum number of computations to decode one branch will also be reduced by the M-path algorithm.

Another interesting feature of the M-path algorithm is that since the correct path may be extended through the dip while not at the top of the stack, the apparent minimum metric values of the top nodes does not correspond to the correct path minimum values, but to the set of nodes that occupied the top of the stack. In Fig. (4.7) the minimum metric value is Γ'_{min} for the 3-path decoder. It is as if the 3-path decoder crossed over the dip, raising its minimum metric value by the quantity

$$\delta_3 = (\Gamma'_{min} - \Gamma_{min}) \tag{4.12}$$

Obviously if the correct path is at the top of the stack during its extension through the bottom of the dip, there is no apparent reduction of the size of the dip and $\delta_3 = 0$. Hence δ_3 may be considered as a measure of the facility with which the dip was crossed over. The larger δ_3 is, the better the M-path decoder operates with respect to the single path sequential decoder.

Ignoring the reconvergence of the paths, an upper bound to the average number of computations necessary to decode one branch may be obtained for the M-path algorithm by following Gallager's derivation (1968) of the same bound for the Fano algorithm.

Consider some node $\underline{U}_j$ on the correct path $\underline{U}$ and an arbitrary node $\underline{U}'_{j+\ell}$ which is ℓ branches away on an incorrect path $\underline{U}'$ that diverged from the correct path at node $\underline{U}_j$. Let $\underline{U}_m$, $m > j$, be the first breakout node following $\underline{U}_j$ on the correct path, that is

$$\underset{i>j}{\text{Min}}\ \Gamma_i^{(\underline{U})} = \Gamma_m^{(\underline{U})} \triangleq \Gamma_{\min} \tag{4.13}$$

Assuming that no decoding errors occur, then node $\underline{U}_m$ will eventually be extended by the M-path stack decoder. We next observe that in general, for an arbitrary correct path metric dip, the M-path algorithm will explore the same subset of incorrect paths as the single path sequential decoder. The exploration will be performed M incorrect nodes at a time instead of one by one. Therefore incorrect node $\underline{U}'_{j+\ell}$ will be extended further if

$$\Gamma_{j+\ell}^{(\underline{U}')} \geq \Gamma_{\min} \tag{4.14}$$

Let $N(\underline{U}'_{j+\ell})$ be a binary random variable defined as

$$N(\underline{U}'_{i+\ell}) = \begin{cases} 1 & \text{if } (\Gamma^{(U')}_{i+\ell} \geq \Gamma_{min} \\ 0 & \text{otherwise} \end{cases} \tag{4.15}$$

Thus the number of computations C_i performed for the decoding of correct node $\underline{U}_i$ and exploration of the subset of incorrect nodes is bounded by

$$C_i \leq M + \sum_{\ell=1}^{\infty} \sum_{\underline{U}'_{i+\ell}} N(\underline{U}'_{i+\ell}) \tag{4.16}$$

where M corresponds to the number of computations for the decoding cycle that extends the correct path, and the summation corresponds to computations on incorrect paths.

From (4.15) we have

$$\overline{N(\underline{U}'_{i+\ell})} = \Pr(\Gamma^{(U')}_{i+\ell} \geq \Gamma_{min}) \tag{4.17}$$

and hence

$$\bar{C}_i \leq M + \sum_{\ell=1}^{\infty} \sum_{\underline{U}'_{i+\ell}} \Pr(\Gamma^{(U')}_{i+\ell} \geq \Gamma_{min}) \tag{4.18}$$

Over the ensemble of convolutional codes for which the branch symbols are statistically independent, Gallager (1968) showed that for any j, $j \geq 0$, and $B \leq E_o(1)$,

$$\Pr[\Gamma^{(U')}_{i+\ell} \geq \Gamma_{min}] \leq (\ell+1)\, e^{-\frac{V\ell(E_o(1)+B)}{2}} \tag{4.19}$$

where $E_o(1)$ is defined in Eq. (3.14) and where B is the metric bias.

The sum over $\underline{U}'_{i+\ell}$ involves fewer than $q^{d\ell} = e^{RV\ell}$ identical terms and thus

$$\bar{C}_i \leq M + \sum_{\ell=1}^{\infty} e^{\ell VR} \, (\ell+1)\, e^{-V\ell \left(\frac{E_o(1)+B}{2}\right)} \tag{4.20}$$

or

$$\bar{C}_i \leq (M-1) + \sum_{\ell=0}^{\infty} (\ell+1)\, e^{-V\ell \left[\frac{E_o(1)+B}{2} - R\right]} \tag{4.21}$$

For $R \leq \frac{1}{2} (E_o (1) + B)$ the sum converges and we obtain

$$\bar{C}_i \leq (M-1) + \left[1 - e^{-V\left(\frac{E_o(1)+B}{2} - R\right)}\right]^{-2} \tag{4.22}$$

and for $B = E_o(1)$ the bound is minimized to

$$\bar{C}_i \leq (M-1) + \left[1 - e^{-RV\left(\frac{E_o(1)}{R} - 1\right)}\right]^{-2} \tag{4.23}$$

The expression in brackets is recognized (Gallager 1968) as the bound on the average number of computations for the single path sequential decoder, under the same conditions, i.e., $R < E_o(1)$ and $B = E_o(1)$.

Let the random variables $C_{1\text{-path}}$ and $C_{M\text{-path}}$ represent the number of computations required to decode one branch for respectively the single path and the M-path sequential decoder. Then from Eq. (4.23) we have

$$\bar{C}_{M\text{-path}} = (M-1) + \bar{C}_{1\text{-path}} \tag{4.24}$$

Regarding $C_{M\text{-path}}$ simply as a translation by $(M-1)$ of $C_{1\text{-path}}$, then the distributions of $C_{M\text{-path}}$ are also a translated version of the distributions of $C_{1\text{-path}}$. In particular, for the finite moments $\overline{C^{\alpha}_{1\text{-path}}}$, $\alpha > 1$, we can use the generalized Chebyshev inequality

$$P (C_{1\text{-path}} \geq \ell) \leq \overline{C^{\alpha}_{1\text{-path}}} \; \ell^{-\alpha} \tag{4.25}$$

to obtain

$$P (C_{M\text{-path}} \geq \ell) = P (C_{1\text{-path}} + M-1 \geq \ell) \leq \overline{C^{\alpha}_{1\text{-path}}} (\ell - M+1)^{-\alpha} \tag{4.26}$$

Since we want an asymptotic behaviour, then for large ℓ, $\ell >> M$ the bound becomes

$$P (C_{M\text{-path}} \geq \ell) \approx k_M \ell^{-\alpha} , \quad k_M = \text{constant} \tag{4.27}$$

The above expression indicates that if the reconvergences of the paths are ignored, the distribution of the computations for the M-path decoder is asymptotically Paretean, with the same Pareto exponent as for the single path sequential decoder. This asymptotic result is not unexpected, as we have observed that if $1 < M < M_v$, the M-path decoder may still enter into deep searches and explore an exponentially increasing number of incorrect paths before reaching the correct path, just like a single path sequential decoder.

Although the Pareto exponent indicates no asymptotic improvement of the M-path algorithm over the single path algorithm, the actual values of the cumulative distribution $P (C \geq \ell)$ will depend on the scale factor k_M or k_1 of Eq. (4.27) and (3.12).

A sequential decoder may be in either of two modes of operation: a search mode where the correct path is not being extended, or a no search mode where the correct path is being extended.

The transition to a search mode occurs when the number of computations exceeds M. In order to compare the two algorithms in that mode of operation, we recall that when the correct path is at the Jth position from the top of the stack, each of the (J-1) incorrect nodes with a higher metric may be regarded as root nodes of subtrees of incorrect paths. The single-path decoder must explore these subtrees in a sequential fashion before getting to the correct path. Therefore even small dips of the correct path metric may require a relatively large number of computations. Such a situation does not occur with the M-path decoder because all dips that leave the correct path among the set of the M-highest metric nodes are simply ignored. Clearly then, for the correct path, to be at the (M+1)th position from the top of the stack is a far less likely event than requiring (M+1) computations before reaching the top of the stack. We have therefore

$$P(C_{M\text{-path}} \geq M+1) \leq P(C_{1\text{-path}} \geq M+1) \tag{4.28}$$

Hence although asymptotically the same, for $\ell > M$ the distribution of the computation, $P(C \geq \ell)$ is smaller for the M-path decoder than for the single path decoder. The variability of the computational effort is therefore reduced for $\ell > M$.

The superiority of the M-path decoder over the single path sequential decoder is further compounded when the purging cycle is taken into account. When a single-path sequential decoder exploits the convergences, the order in which the converging paths reach the converging nodes determines whether some computations may be eliminated or must be repeated. With the M-path decoder, the paths may now converge simultaneously. The unique survivor among these converging paths can be determined and no duplication of computations among these paths is ever possible. However, because the survivor at a node is not in general the survivor of all possible distinct paths converging at that node, the repetition of identical computations can still occur. Clearly, as M increases, the operation of the M-path algorithm approaches that of the Viterbi decoding algorithm , but unless $M = M_v$, the computational effort will maintain its variable nature.

4.3.5 <u>Error Probability</u>

The M-path sequential decoder being a sub-optimal sequence estimator, its error probability is lower bounded by the error probability of the optimal (Viterbi) decoder. We now show that it is upper bounded by the error probability of the single path sequential decoder.

Regardless of the decoder, an error occurs when an incorrect path is accepted when remerging with the correct path, that is when an incorrect path metric is larger than the correct path metric at the merging node. However, because of the sequential nature of the path extensions, some events may cause decoding errors for a sequential decoder, but not for an optimal decoder.

Consider an incorrect path $\underline{U}'$ which remerges with the correct path $\underline{U}$, ℓ branches, $\ell > k$ beyond the node $\underline{U}_j$ from which it diverged. By the rules of sequential decoding, path $\underline{U}'$ will be extended only if

$$\Gamma_i^{(\underline{U}')} \geq \operatorname*{Min}_{j \leq n} \Gamma_n^{(\underline{U})} \quad i \geq j \tag{4.29}$$

As shown in Fig. (4.8a), if at the remerging node we have

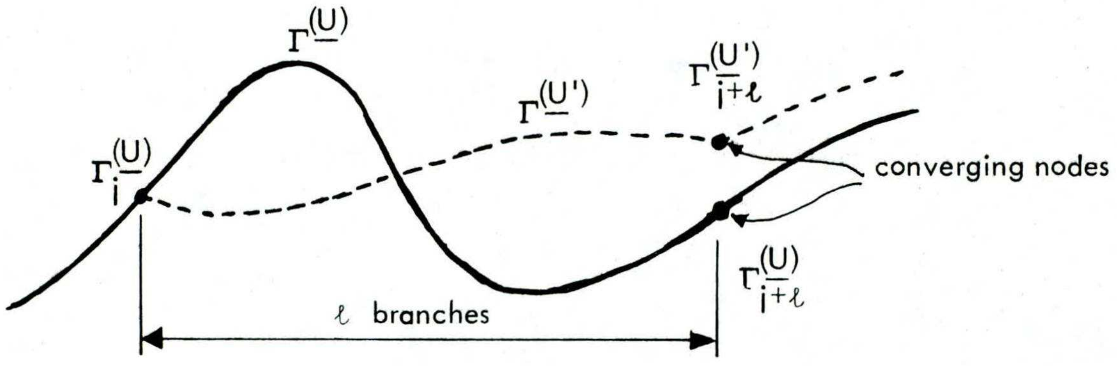
$$\Gamma_{j+\ell}^{(\underline{U}')} > \Gamma_{j+\ell}^{(\underline{U})} , \tag{4.30}$$

then an error would occur with both the sequential and optimum decoder.

Now suppose we have

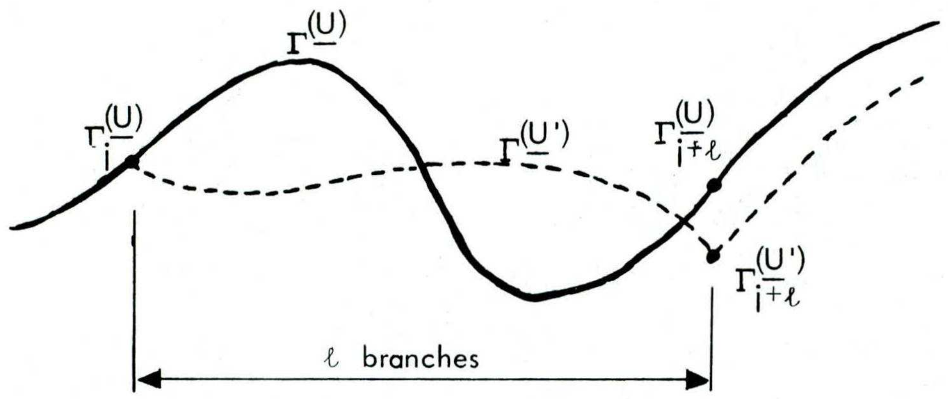
$$\Gamma_{j+\ell}^{(\underline{U}')} \leq \Gamma_{j+\ell}^{(\underline{U})} \tag{4.31}$$

If both paths $\underline{U}'$ and $\underline{U}$ reach simultaneously the converging node, the incorrect path is eliminated and condition (4.31) entails no error. However, with the single path decoder, the incorrect path $\underline{U}'$ may reach the remerging node $\underline{U}_{j+\ell}$ before correct path $\underline{U}$. Consequently, if node $\underline{U}_{j+\ell}$ is a strict break out node, the decoder will never back-up to extend the correct path $\underline{U}$ to node $\underline{U}_{j+\ell}$, and decoding errors will occur (see Fig. 4.8b). However, as shown in Fig. (4.8c) if node $\underline{U}_{j+\ell}$ is not a break out node then condition (4.31) does not lead to decoding errors.



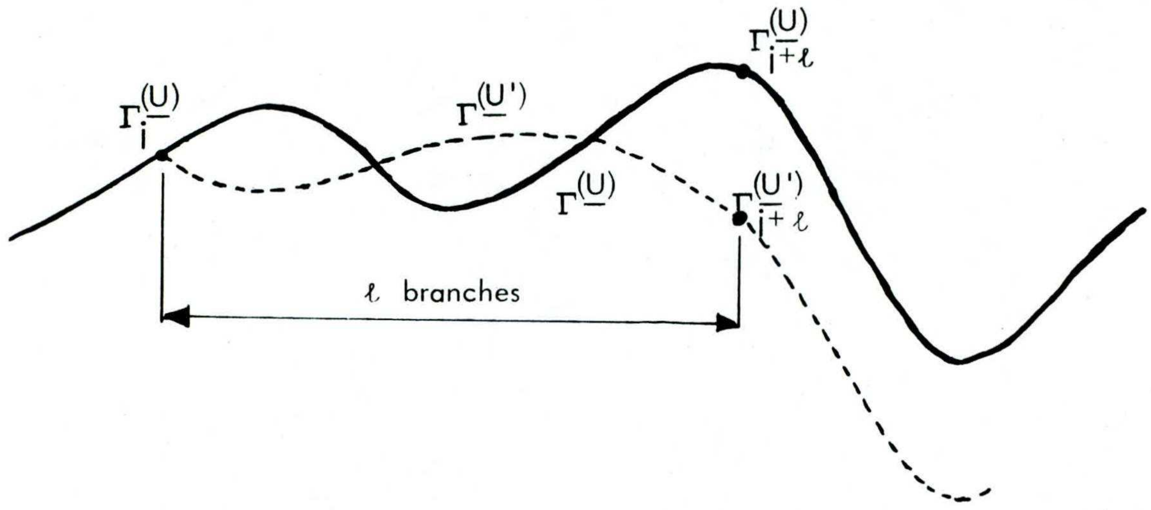
$\Gamma^{(\underline{U})}$
$\Gamma^{(\underline{U}')}_{i+\ell}$
$\Gamma^{(\underline{U}')}$
$\Gamma^{(\underline{U})}_{i}$
converging nodes
$\Gamma^{(\underline{U})}_{i+\ell}$
ℓ branches

(a) Uncorrectable error-event.



$\Gamma^{(\underline{U})}$
$\Gamma^{(\underline{U})}_{i}$
$\Gamma^{(\underline{U}')}$
$\Gamma^{(\underline{U})}_{i+\ell}$
$\Gamma^{(\underline{U}')}_{i+\ell}$
ℓ branches

(b) Correctable error event for M-path algorithm.



$\Gamma^{(\underline{U})}_{i+\ell}$
$\Gamma^{(\underline{U})}_{i}$
$\Gamma^{(\underline{U}')}$
$\Gamma^{(\underline{U})}$
$\Gamma^{(\underline{U}')}_{i+\ell}$
ℓ branches

(c) Correctable error event for both single and M-path algorithm.

Figure (4.8) Error events.

Let a "proto-error event"(Forney —) be the event in which some incorrect path $\underline{U}'$ has a metric $\Gamma_{j+\ell}^{(\underline{U}')}$ at the point of merger with the correct path such that

$$\Gamma_{j+\ell}^{(\underline{U}')} \geq \underset{n \geq j}{\text{Min}} \; \Gamma_n^{(\underline{U})} \tag{4.32}$$

Then from the above discussion on the error events, no error can occur without a proto-error event whereas a proto-error event may not lead to an error. Therefore the error probability of sequential decoding is upper bounded by the probability of proto-error $P_1(E)$. Using the union bound, the ensemble average proto-error probability bound is

$$\overline{P_1(E)} \leq \sum_{\underline{U}'_{j+\ell}} \overline{\Pr\left[\Gamma_{j+\ell}^{(\underline{U}')} \geq \underset{n > j}{\text{Min}} \; \Gamma_n^{(\underline{U})} \right]} \tag{4.33}$$

There are less than $q^{d\ell} = e^{\ell RV}$ incorrect paths that remerge with the correct path after diverging in $(K+\ell)$ branches. Using Eq. (4.19) we have

$$\overline{P_1(E)} \leq \sum_{\ell=0}^{\infty} e^{\ell RV} (K+\ell+1)\, e^{-(K+\ell)\left(\frac{E_o(1)+B}{2}\right)V}, \quad B \leq E_o(1) \tag{4.34}$$

$$\overline{P_1(E)} \leq e^{-KV\left(\frac{E_o(1)+B}{2}\right)} \left[\sum_{\ell=o}^{\infty} K\, e^{-\ell V\left(\frac{E_o(1)+B}{2} - R\right)} + \sum_{\ell=o}^{\infty} (\ell+1)\, e^{-\ell V\left(\frac{E_o(1)+B}{2} - R\right)} \right] \tag{4.35}$$

For $R < \frac{1}{2}\left(E_o(1) + B\right)$ both sums converge to give

$$\overline{P_1(E)} \le e^{-KV\left(\frac{E_o(1)+B}{2}\right)} \left[\frac{K}{1 - e^{-V\left(\frac{E_o(1)+B}{2} - R\right)}} + \frac{1}{\left[1 - e^{-V\left(\frac{E_o(1)+B}{2} - R\right)}\right]^2} \right] \tag{4.36}$$

The bound decreases with increasing B, but is only valid for $B \le E_o(1)$. For $B = E_o(1)$ the bound is minimized to

$$\overline{P_1(E)} \le e^{-KV E_o(1)} \left[\frac{K}{\left[1 - e^{-VR\left(\frac{E_o(1)}{R} - 1\right)}\right]} + \frac{1}{\left[1 - e^{-VR\left(\frac{E_o(1)}{R} - 1\right)}\right]^2} \right] \tag{4.37}$$

$$\overline{P_1(E)} \le e^{-KV E_o(1)} \cdot \frac{K+1}{\left[1 - e^{-VR\left(\frac{E_o(1)}{R} - 1\right)}\right]^2}, \quad R < E_o(1) \tag{4.38}$$

Because some of the proto-error events that lead to actual error events for the single path decoder may be rejected by the M-path decoder we have

$$\overline{P(E)}_{M\text{-path}} \le \overline{P(E)}_{1\text{-path}} \le \overline{P_1(E)} \tag{4.39}$$

Since none of the incorrect paths satisfying inequality (4.31) lead to decoding errors for the optimal decoder, we have

$$\overline{P(E)}_{\text{Viterbi}} \leq \overline{P(E)}_{M\text{-path}} \leq \overline{P(E)}_{1\text{-path}} \leq \overline{P_1(E)} \qquad (4.40)$$

Summarizing, we have proved the following theorem.

<u>Theorem 4.2</u>

The error probability of the M-path sequential decoder is lower bounded by the error probability of the optimum decoder and upper bounded by the error probability of the single path sequential decoder.

To summarize, by improving the distribution of the computations of the single path sequential decoding, the new M-path algorithm alleviates one of the main drawbacks of sequential decoding. However if only a subset of the distinct paths are extended in the search for the most likely transmitted sequence ($M \leq M_v = 2^{K-1}$) a Pareto distribution of the computation seems to be asymptotically inescapable. By including a purging cycle, the trellis structure of the convolutional code is exploited whenever possible and futile computations are avoided. The error probability of the M-path decoder is upper bounded by the error probability of the single path decoder, but the average decoding effort is larger for the M-path decoder than for the single path decoder.

4.4 Computer Simulation Results

In order to compare the performances of the Z-J decoding algorithm and the M-path algorithm, both algorithms were programmed in FORTRAN and run under identical conditions on the IBM 360/75 computer at McGill University. In this section we describe the simulated system and present some of the results of the simulation.

4.4.1 System Description

The simulated communication system consists of a rate $1/V$ convolutional encoder, a modulator, a Gaussian noise channel, a demodulator and a sequential decoder. The input data was assumed to be a sequence of equally probable and mutually independent binary symbols which the encoder maps into a binary code sequence. Antipodal signalling was assumed, with a signal energy E_s given by

$$E_s = \frac{1}{V} E_b \tag{4.41}$$

where E_b is the energy per transmitted bit. The jth transmitted symbol is

$$s_j(t) = \pm \sqrt{E_s}\, \varphi(t) \tag{4.42}$$

where $\varphi(t)$ is a convenient unit energy pulse waveform.

To the transmitted signal, the channel adds a zero mean white Gaussian noise with power spectral density of $\frac{N_o}{2}$ watts/Hz. The optimum demodulator (which is a filter matched to $\varphi(t)$), delivers an output r_i' corresponding to the code symbol $s_i(t)$. Normalizing this output by dividing by $\sqrt{N_o/2}$ yields

$$r_i = \pm\sqrt{\frac{2E_s}{N_o}} + n_i \tag{4.43}$$

where n_i is now a zero-mean, unit variance Gaussian random variable.

To facilitate the digital processing the continuous r_i's are quantized. Soft 8-level quantization with evenly spaced boundaries was used as it entails only a small (0.25 dB) degradation of (E_b/N_o) compared with an infinitely fine quantization (Jacobs 1967). The quantization intervals are specified in Fig. (4.9).

With the received signal quantized, the modulator, channel and demodulator can be considered as being a discrete memoryless channel with 2 equally likely inputs and 8 outputs. All the details of the mudulator, channel and demodulator are contained in the transition probabilities of this new channel. These transition probabilities are a function only of the signal-to-noise ratio E_s/N_o and for the quantization scheme of Fig. (4.9) are given in Table 4.1. Some other parameters of interest are given in Table 4.2.

In the simulation, in order to determine the actual quantized output symbol of this new channel given the input symbol, the unit interval is segmented in 8 sub-intervals, each of which has a length proportional to the corresponding calculated transition probability. For each channel use, a pseudo- random number

TABLE 4.1

Transition from input 0	Transition Probability		
	E_b/N_o = 3.0 dB	E_b/N_o = 3.5 dB	E_b/N_o = 4.0 dB
0	0.465818	0.499207	0.534573
1	0.194822	0.191554	0.186748
2	0.159057	0.150101	0.140110
3	0.101653	0.092072	0.082287
4	0.050853	0.044208	0.037828
5	0.019912	0.016613	0.013610
6	0.006102	0.004886	0.003832
7	0.001783	0.001359	0.001012

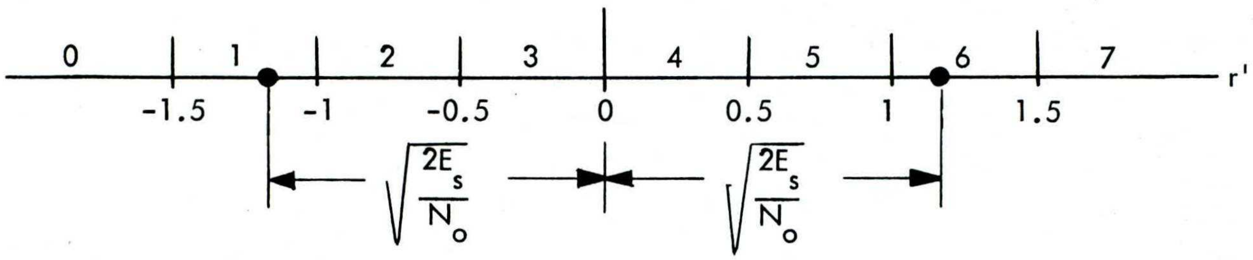


Figure (4.9) Channel output quantization.

uniformly distributed over the interval [0, 1] is generated, and the channel output symbol then corresponds to the subinterval which contains the value of this pseudo-random number.

The best rate 1/2 convolutional codes of short constraint lengths ($K \leq 7$) obtained by Odenwalder (1970) were used. Short constraint length codes were chosen to exhibit the effect of the reconvergence of the paths, and to permit the comparison with simulation results of the Viterbi decoder reported by Heller and Jacobs (1971).

Branch metric values are computed by adding the metric values of the two symbols on the branch, according to Eq. (3.8), with a bias equal to the rate in bits/symbol. The metric values are rounded to integer values and are listed in Table 4.3.

Since the simulated channel is symmetric from the input and since the code is linear, the probability of decoding error is independent of the particular input information sequence. Consequently, for convenience, an all-zero information sequence was assumed to be transmitted and thus the coded sequence is also an all-zero sequence. This eliminates the need for a convolutional encoder at the transmitting end.

The data sequence was divided in blocks of 500 bits each, after which a tail of zeros was inserted to clear all registers.

TABLE 4.2

E_b/N_o (dB)	R_{comp}	R/R_{comp}	α
3.0	0.532	0.939	1.21
3.5	0.577	0.866	1.54
4.0	0.623	0.803	1.89

TABLE 4.3

Metric Values

E_b/N_o (dB)	Symbols	Received Symbols							
		0	1	2	3	4	5	6	7
3.0	0	4	4	3	-1	-9	-21	-36	-60
	1	-60	-36	-21	-9	-1	3	4	4
3.5	0	4	4	3	-1	-9	-23	-39	-64
	1	-64	-39	-23	-9	-1	3	4	4
4.0	0	4	4	3	0	-9	-24	-41	-68
	1	-68	-41	-24	-9	0	3	4	4

In the quantized version of the Z-J algorithm (Jelinek 1969 a), the top node of the stack is chosen at random among the nodes stored in the highest non-vacant substack. In our simulation the true top node of the stack was determined by scanning the nodes stored in the highest non-vacant substack. However for the M-path algorithm , $M > 1$, the remaining (M-1) paths were chosen at random as in the Jelinek's scheme.

Through preliminary simulation, it was observed that whenever the decoder ceased to advance deeper in the tree and was searching back, extending a single path instead of M paths in the path extension cycle would improve slightly both the overall distribution of the computations and the average decoding effort. This new decision rule does not alter the essential properties of the M-path algorithm , and consequently all simulation runs were performed using this decision rule on the path extension cycles. The elimination of the redundancy was the purging rule for the purging cycles. No other purging rule was considered.

4.4.2 Results of the Simulation

In order to compare the performances of the new M-path algorithm with the Z-J algorithm , both algorithms were run under strictly identical conditions, using the same code and the same noise sequence.

The principal output of the simulation was a histogram of the number of computations per search, from which the cumulative distribution could easily be obtained. A search consists of all computations done between first reaching some depth N and first penetrating to depth (N + 1). This method of collecting the data was used because it was simpler to implement than the counting of the number of computations done in the subset of incorrect paths emerging from the correct nodes. Both methods however yield essentially the same result (Niessen 1965, Heller 1967).

Along with the distribution of computations, at the end of each run the program also delivered the average number of computations per decoded bit, the total number of errors, the total number of converging nodes that were stored (converging with a larger metric) and eliminated (converging with a smaller metric), and the cumulative distribution of the metric dips encountered by the top nodes of the stack.

The following data was also collected at the end of each decoded block: number of errors, size of the stack, total accumulated metric of the decoded path, number of converging nodes stored and eliminated, largest metric dip encountered by the top nodes of the stack, and the search depth or maximum number of nodes which the decoder backs up during a search.

Distribution of the Computations

Fig. (4.10a-d), show the empirical cumulative distributions of the number of computations per search for $M = 4$, and $K = 5$, $(E_b/N_o) = 4$ dB; $K = 6$, $(E_b/N_o) = 3.5$ dB and 3.0 dB; $K = 7$, $(E_b/N_o) = 3.0$ dB.

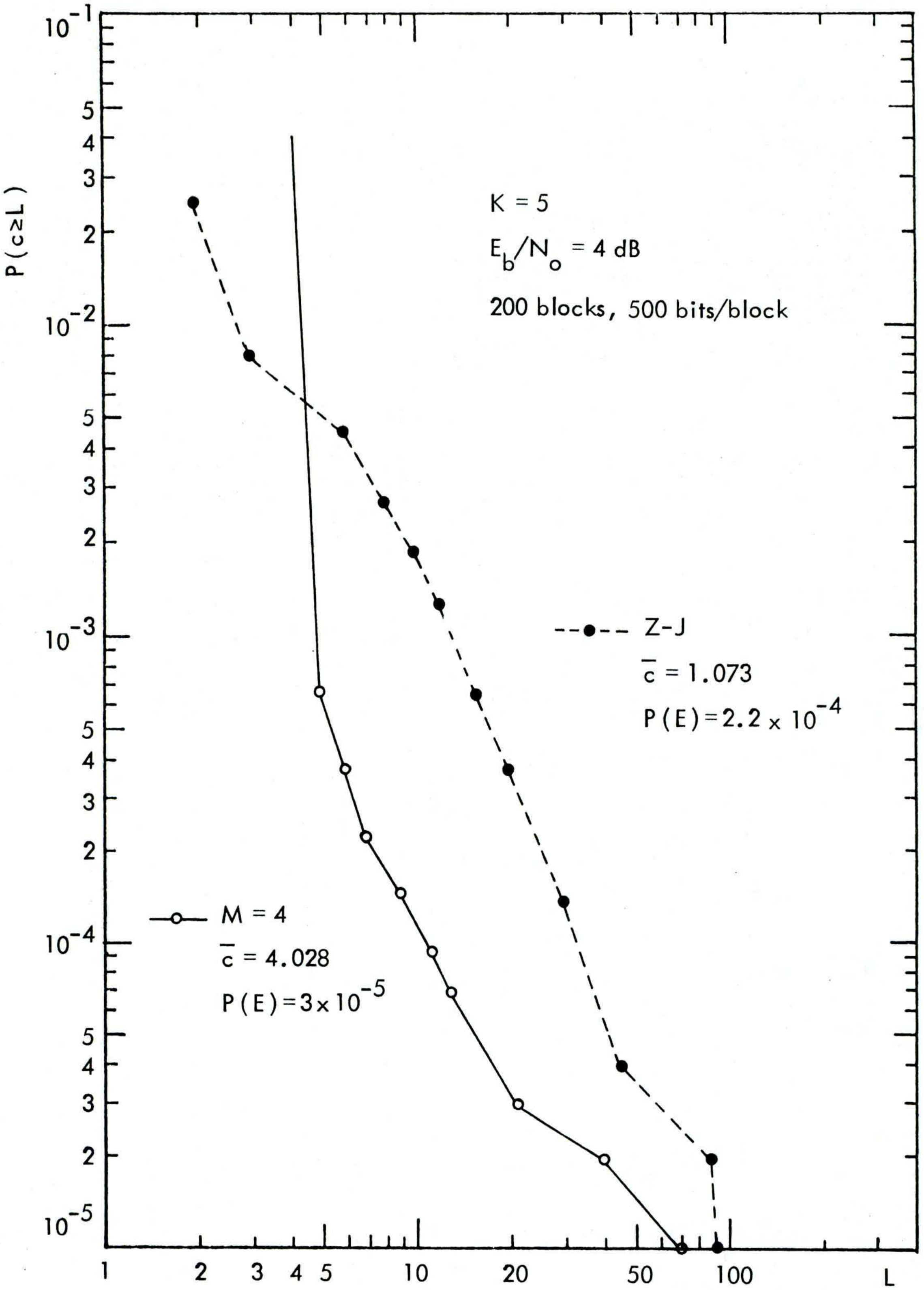


Figure (4.10a) Empirical distribution of computations per search for the M-path algorithm.

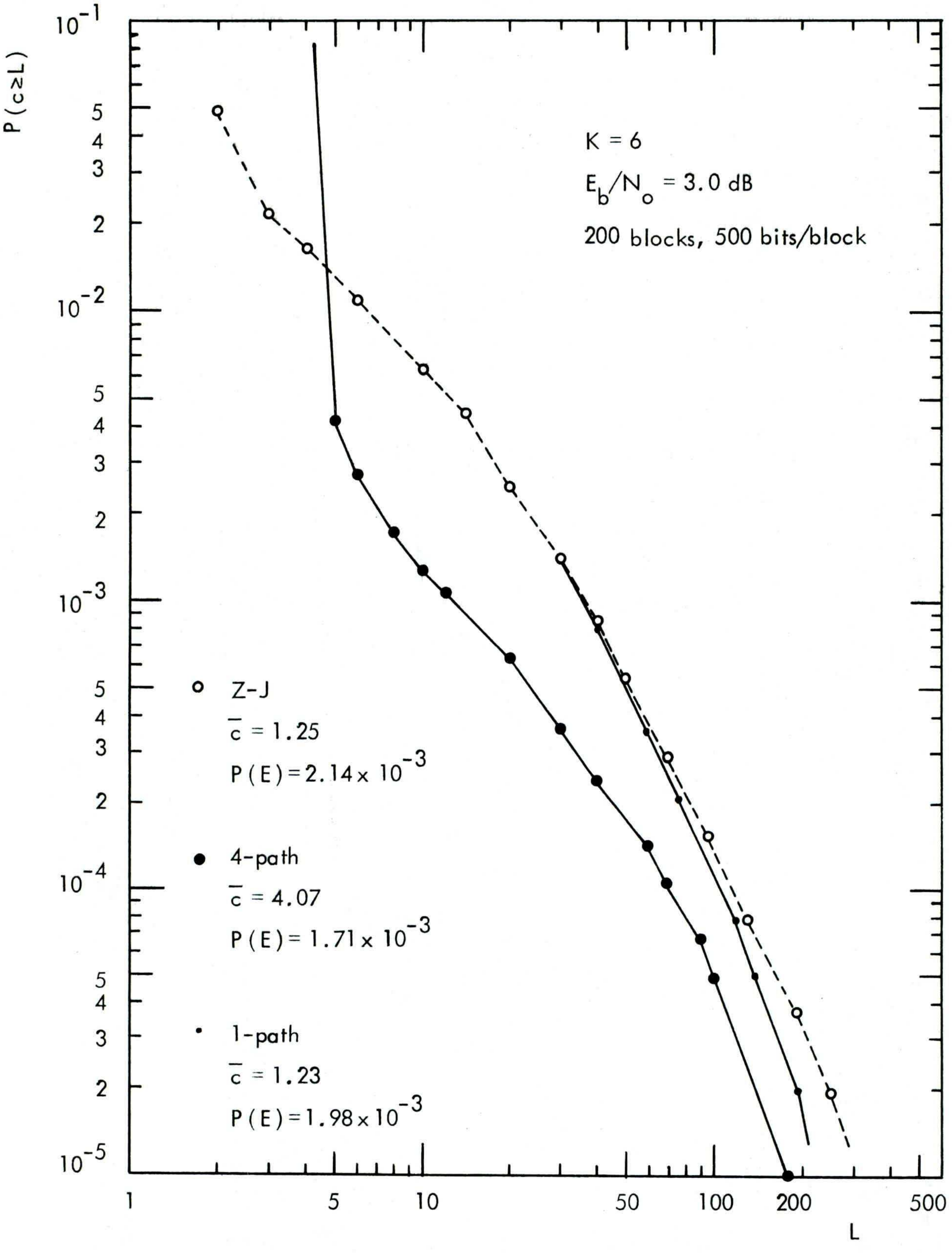


Figure (4.10b) Empirical distribution of computations per search for the M-path Algorithm.

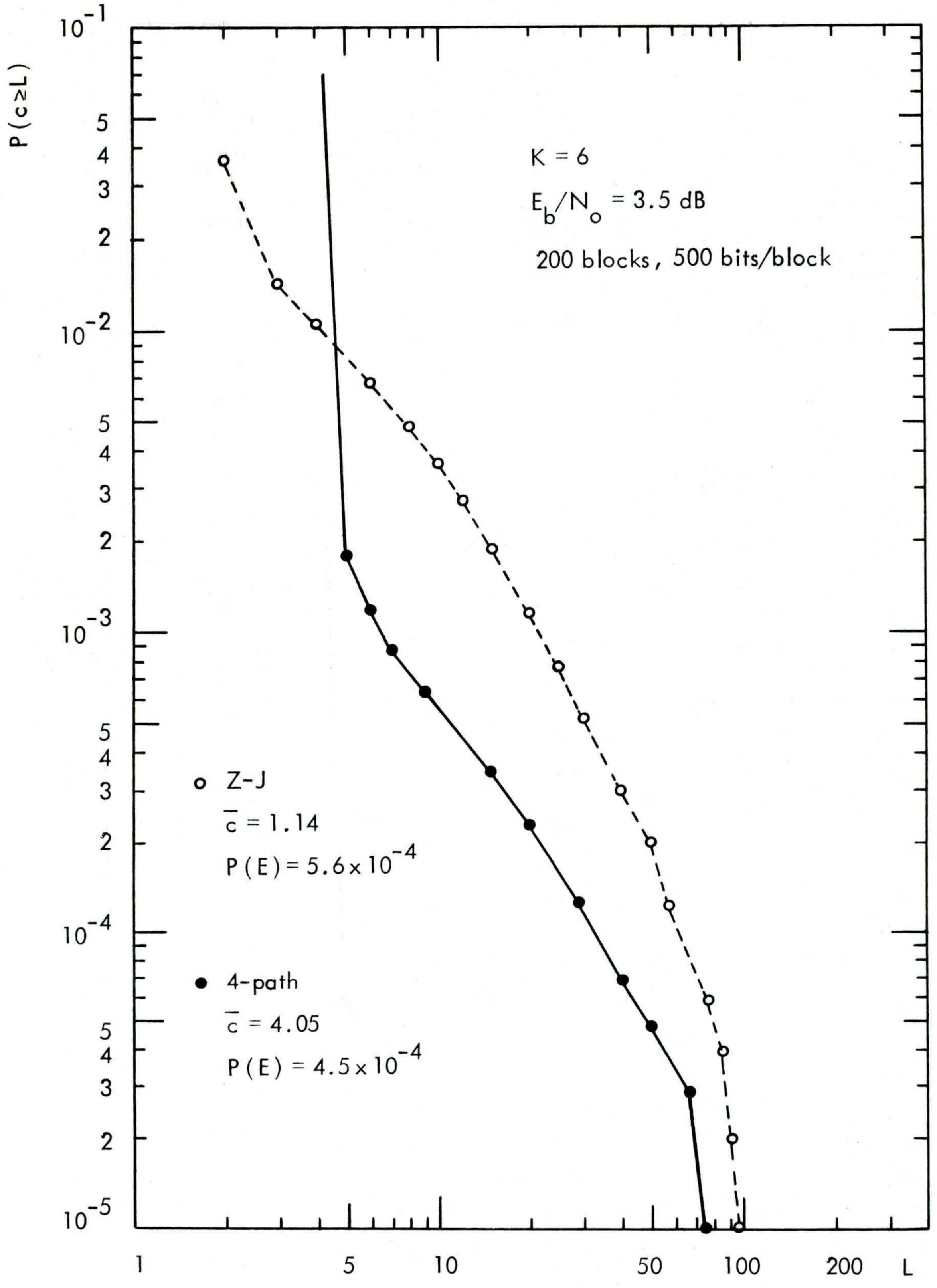


Figure (4.10c) Empirical distribution of computations per search for the M-path Algorithms.

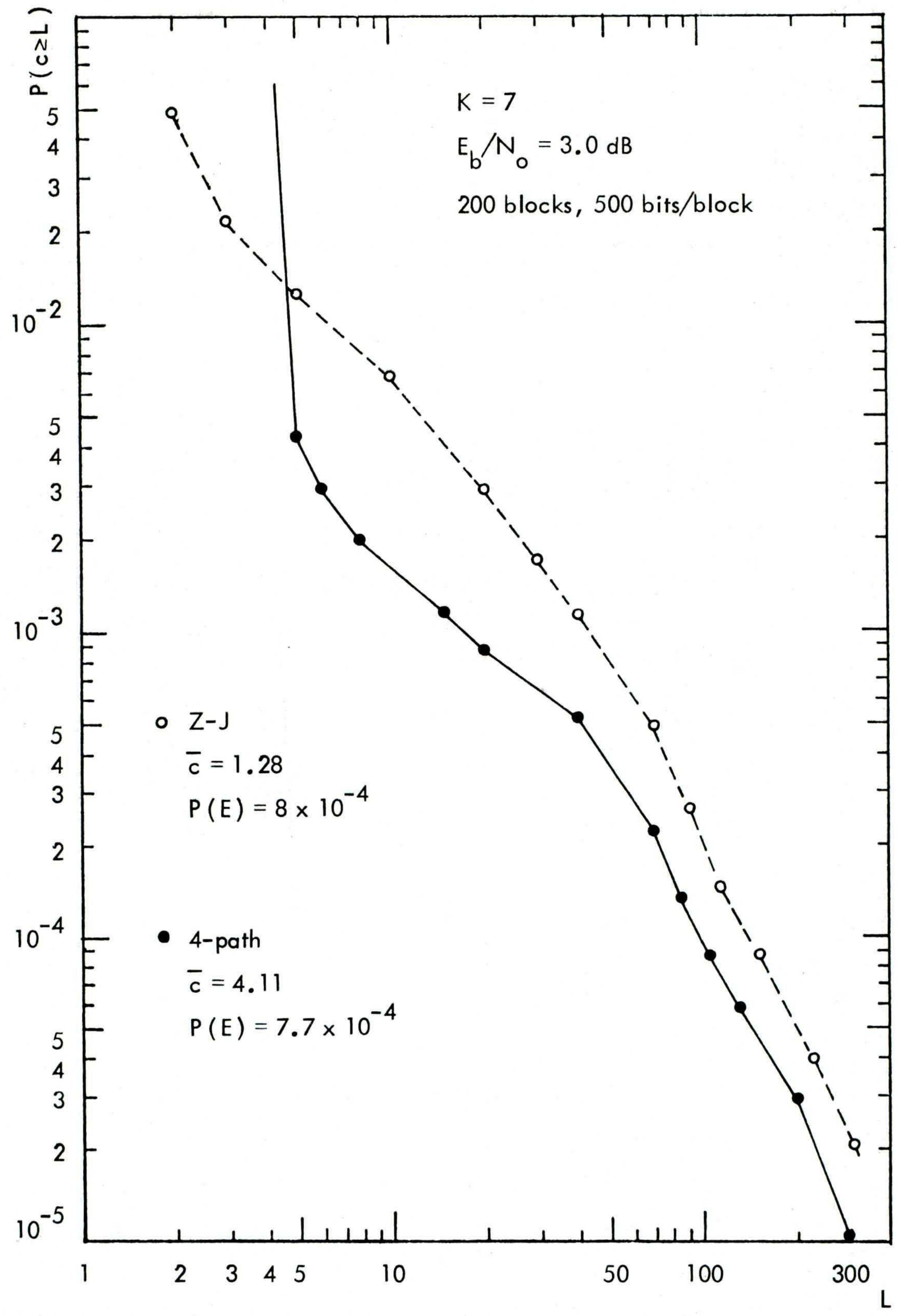


Figure (4.10d) Empirical distribution of computations per search for the M-path algorithm.

As predicted the 4-path algorithm shows a marked improvement over the Z-J algorithm run under the same conditions and the identical noise sequence: the distribution curves of the 4-path algorithm are always below the corresponding curves of the single path stack algorithms for $C > 4$.

The no-search ($C \le 4$), and search ($C > 4$) modes of operation are well displayed. The sharp transition between the two modes of operation supports the assertion that going from M to ($M+1$) computations, ($M>1$), is a far less probable event for the M-path algorithm than it is for the single path algorithm . In fact it demonstrates that even small dips of the decoded path metric may cause a relatively large decoding effort for the single path decoder while being entirely ignored by the M-path decoder. The ratios of the observed values of $P (C \ge M+1)$ for the Z-J algorithms and the M-path decoder are listed in Table 4.4 for $M = 3$ and 4. This limited comparison shows that for a given (E_b/N_o) the ratio increases very fast with M and decreases slowly with K. The dependence on M is obvious whereas the decrease with K indicates that the occurence of remergers is in part responsible for the superiority of the M-path decoder. For a given K and M, the ratio increases with (E_b/N_o), which is intuitively satisfying since the probability of having large dips (for which the M-path decoder is no better than the Z-J decoder) decreases with increasing (E_b/N_o).

When plotted on log/log paper, a Pareto distribution appears as a straight line whose slope has a magnitude equal to the Pareto exponent. An

TABLE 4.4

(E_b/N_o) dB	K	$\frac{P(C \geq M+1)_{Z-J}}{P(C \geq M+1)_{M\text{-path}}}$		
		M* = 3	M = 4	M = 1
4	4	3.57		
	5	2.52	8.1	
3.5	6	2.0	4.73	
3.0	6		3.45	1.01
	7	1.48	2.88	

* Different noise sequence from M = 4 and Z-J.

accurate estimate of the tail of the distribution was difficult to obtain within a reasonable amount of computing time. For each run 200 blocks of 500 bits each were decoded, and only very few events contributed to the data points on the tail, hence reducing the reliability of these data points. However the respective comparisons with the Z-J algorithm are valid since both algorithms were run under strictly identical conditions. The initial advantage of the 4 path algorithm for small searches decreases gradually for longer searches and as in Fig. (4.10 a-d), the curves for the 4-path and Z-J algorithms tend to run parallel to each other in the tail, with the 4-path curve always below the Z-J curve. A run with $M = 1$ (Z-J algorithm with a purging cycle) does not show much improvement for small searches, but the slope of the tail is slightly larger than for the Z-J algorithm. This tends to support the conjecture that the Pareto exponent is increased when the Z-J algorithm exploits the reconvergence of the paths.

<u>The average number of computations</u> was observed to follow approximately the relation

$$\bar{C}_{M\text{-path}} = (M-1) + \bar{C}_{Z\text{-}J}$$

as predicted earlier (see Eq. (4.24)). Table 4.5 lists the average values obtained for the simulation, and as expected, the average decoding effort is rather insensitive to the constraint length.

TABLE 4.5

K	E_b/N_o (dB)	M	Average Comp./bit	Average storage per block	Average Comp./bit Z-J
4	4.0	3*	3.022	2795	1.08
5	4.0	3*	3.029	2958	1.083
		4	4.028	3860	
6	3.5	3*	3.063	3037	1.147
		4	4.049	4006	
	3.0	4	4.074	4014	1.260
7	3.0	3*	3.164	3139	1.292
		4	4.108	4090	

* different noise sequence from $M = 4$ and Z-J.

The average number of nodes stored in the stack per decoded block of 500 bits is listed in Table 4.5. For a Z-J decoder working on a binary tree, this average number $\bar{S}$ is

$$\bar{S} = (2L\,\bar{C} + 1) \tag{4.44}$$

where L is the block length in bits and $\bar{C}$ the average number of computations per decoded bit. Table 4.5 shows the reduction in storage due to the exploitation of the convergences.

Cumulative distribution of the metric dips of the top nodes of the stack. We recall that one of the features of the multiple path extension is to allow the correct path to cross over the bottom of its metric dip while not being at the top of the stack. It was shown that this event may cut down on the number of incorrect paths to be explored. Consequently the apparent size of the metric dip of the top nodes of the stack is smaller than that of the decoded path, by a quanitity δ_3. (See Eq. 4.12). Fig. (4.11 a-d), give an empirical distribution of the apparent decoded path metric dips for the Z-J algorithm and the 4-path algorithm. It shows that the reduction in size may reach the equivalent of several maximum values of the branch metric for relatively small dips, but the improvement decreases as the size of the dips increases.

Path Remerging

A new branch extension is eliminated (stored) if it converges with a smaller (larger) metric on a node previously stored in the stack. Table 4.6 lists the

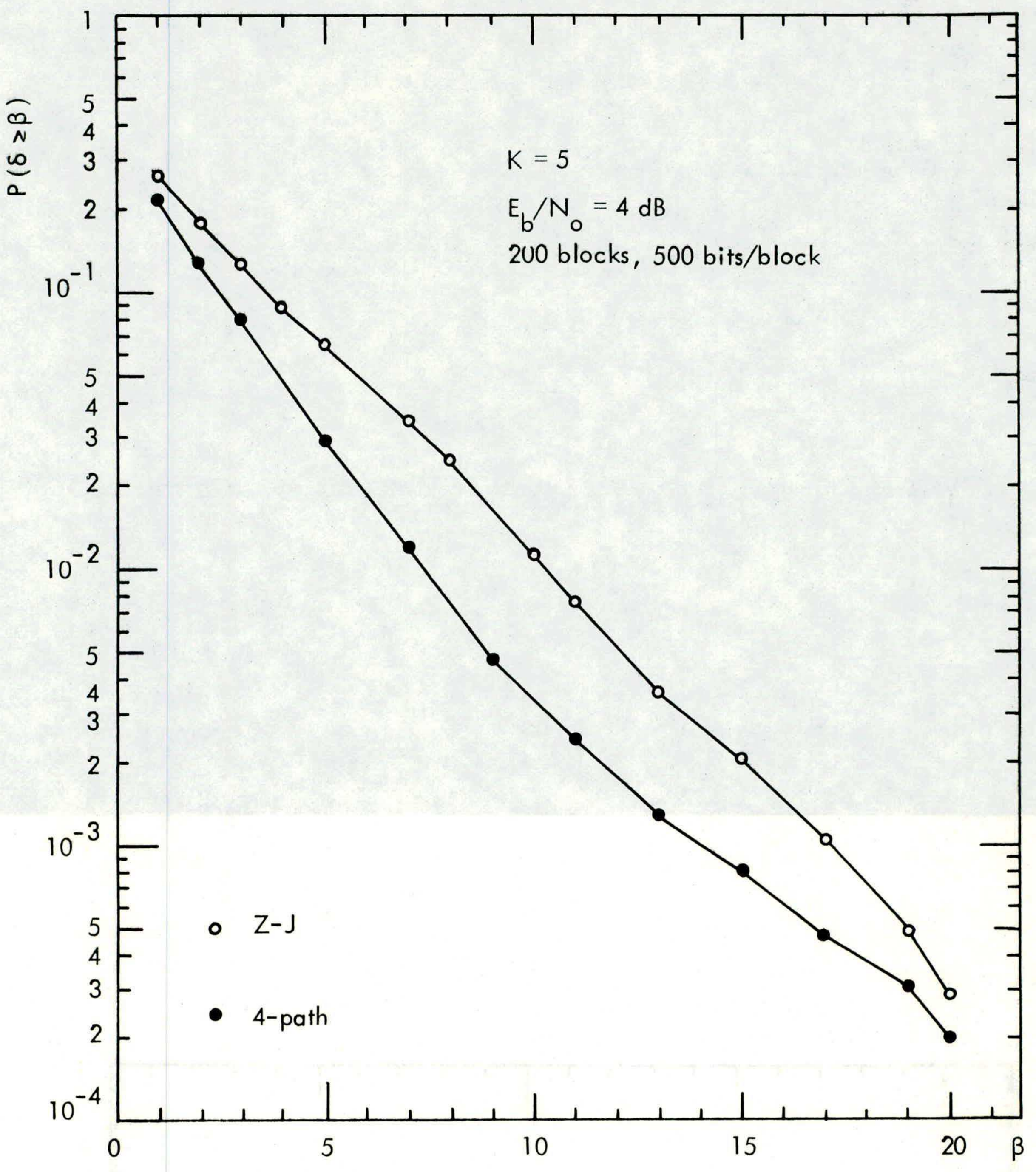


Figure (4.11a) Empirical distribution of the metric dips of the top nodes. (dip unit = Max. branch metric).

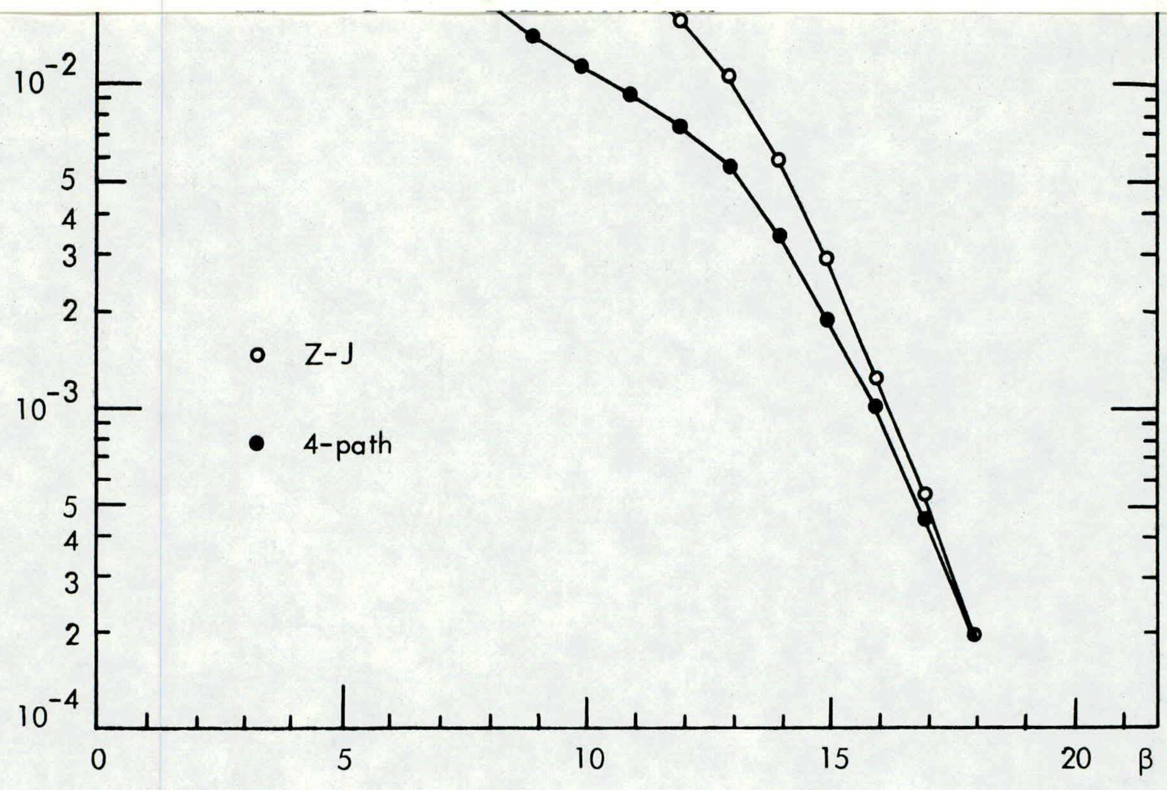


Figure (4.11b) Empirical distribution of the metric dips of the top nodes. (dip unit = Max. branch metric).

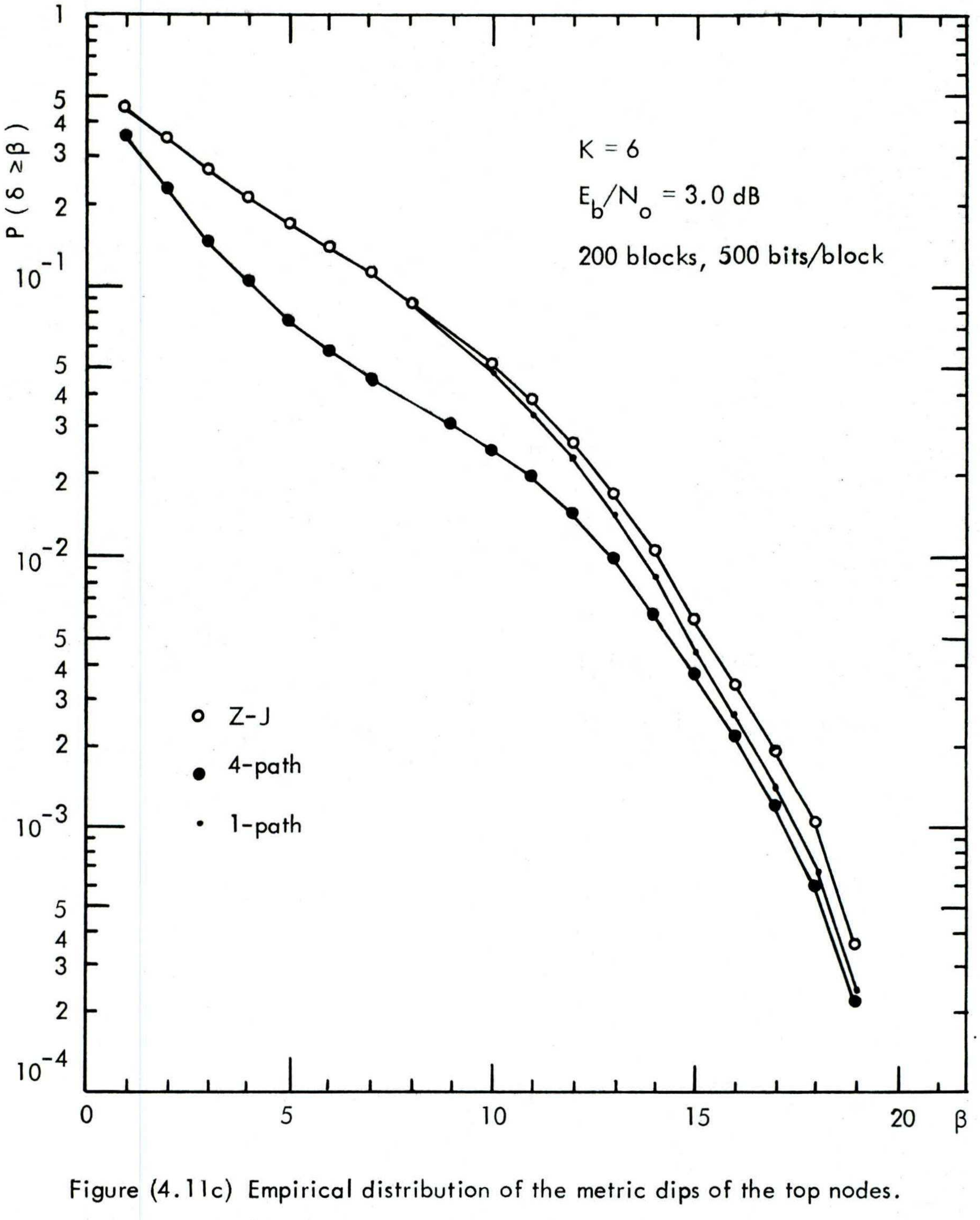


Figure (4.11c) Empirical distribution of the metric dips of the top nodes. (dip unit = Max. branch metric).

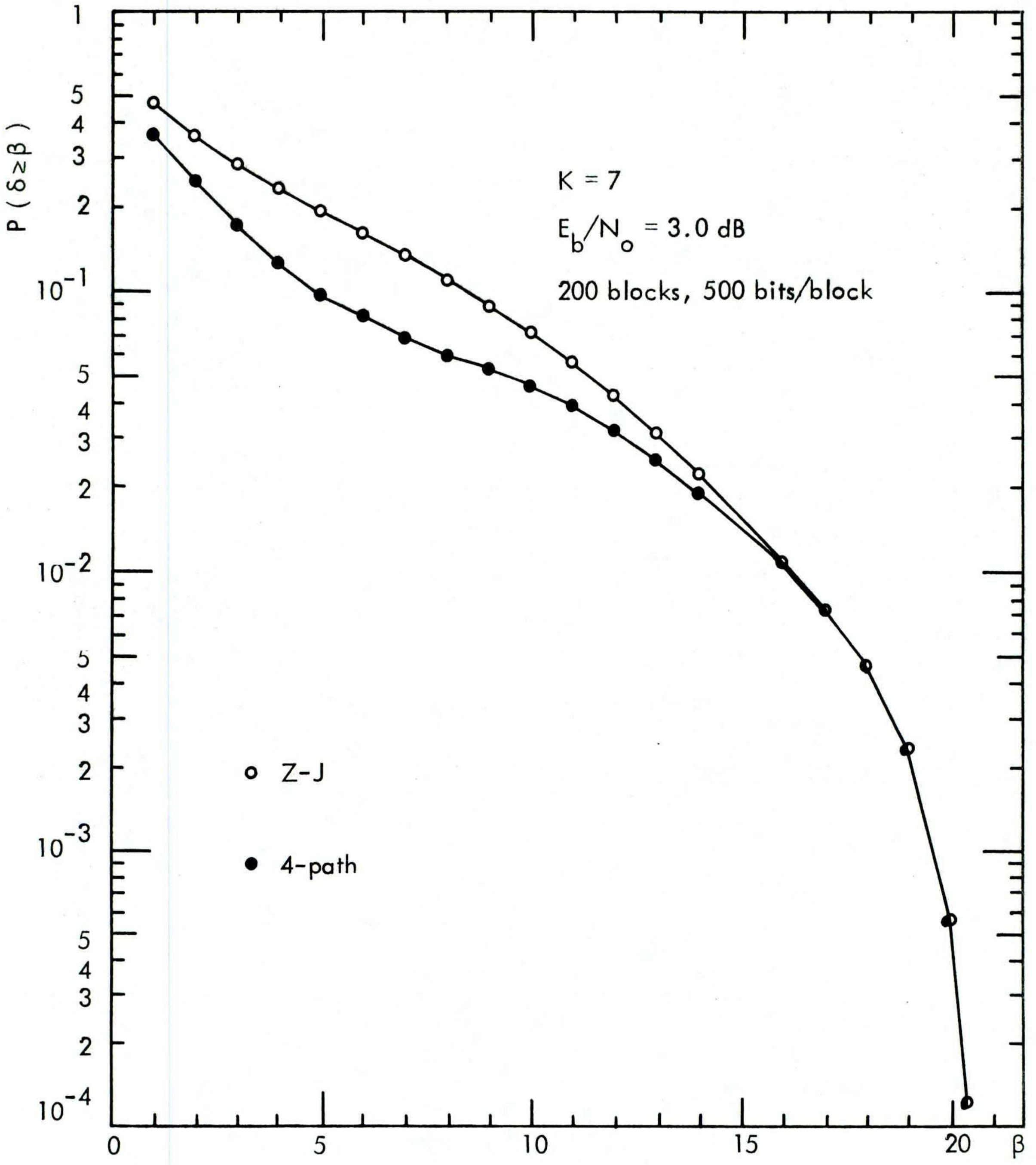


Figure (4.11d) Empirical distribution of the metric dips of the top nodes. (dip unit = Max. branch metric).

number of nodes eliminated and stored during the simulation. It shows that for $M = 1$ the number of remerging events is not very large, but increases very fast with increasing M, and decreasing both K and (E_b/N_o). Fixing K and (E_b/N_o), the number of remerging events should be an exponential function of M, saturating at $L2^{K-1}$ for $M = M_v$. Since the number of states and proto-error events decrease exponentially with K, fixing (E_b/N_o) and M, the number of convergences also decreases exponentially with K.

Because the incorrect paths tend to have a decreasing cumulative metric, as expected the number of nodes eliminated is larger than the number of nodes stored.

From Theorem 4.1 we know that when a remerging node is eliminated, redundant computations of the Z-J algorithm are eliminated by the new algorithm, whereas if the remerging node is stored, then previous computations may have to be repeated. This duplication of computations shows in some sense a limitation of the new algorithm in its ability to reduce the variability of the computational effort. In our simulation computations may be duplicated only when a new extension converges with a larger metric on a previously extended node. Table 4.6 lists the number of these events under the column "repeat". It shows conclusively that only very few convergences are of the "repeat" type. From the data collected after each decoded block, it is observed that this number is sparsely distributed among the various blocks. In fact, it occured only once, for $K = 7$ and $(E_b/N_o) = 3$ dB that most (516 out of a total of 638) of the "repeat"

TABLE 4.6

K	$(\frac{E_b}{N_o})$	M	Nodes eliminated	Nodes stored	Repeat	Errors M-path	Errors Z-J
4	4	1**	668	458	1	44	49
		3**	29979	14487	435	31	
5	3.5	1*	1156	760	16	81	
	4	1**	352	230	5	16	22
		3**	8808	5130	329	3	
		4*	25737	12927	167	3	
6	3.0	1*	1317	959	39	198	214
	3.5	1**	447	323	11	51	56
		3**	2793	2315	313	43	
		4*	7263	4497	196	45	
7	3.0	1**	1286	816	35	84	80
		3**	2405	2707	638	81	
		(4*)	2575	1789	178	77	

NOTES:	-	200 blocks, 500 bits/block.
	*	Same noise sequence as for Z-J algorithm.
	**	Different noise sequence from above.
	( )	Unreliable because of erasures.

convergences took place within a single block. Therefore it may be conjectured that the vast majority of the remergers occur among the end nodes of the explored paths and hence the extended paths are grouped together at about the same depth.

Error Probability

Results on the bit error counts listed in Table 4.6 amply verify our assertion that the error probability for the M-path algorithm is upper bounded by the error probability for the Z-J algorithms (see Theorem 4.2). By using the same noise sequence on the Z-J and the M-path algorithms, $M > 1$, the effect of the multiple path extension was demonstrated by observing that each block for which there was a reduction of the number of errors, the decoded path had a larger metric value with the M-path algorithms than with the Z-J algorithm.

For $M = 1$ no theoretical modification of the error mechanism is introduced, the decoded paths always have the same metric value as for the Z-J algorithm. Hence for both algorithms the error probability bounds are the same. In fact from our discussion on the mechanism of the error events, the number of stored converging nodes may be considered an upper bound on the number of decoding errors. The observed reduction (see Table 4.6) of the error count with the 1-path algorithm comes from the way ties are resolved. We recall that nodes are extracted from the stack in the first-in last-out mode. If two paths remerge with the same metric value, the Z-J algorithm stored them both, but should one of them be extended, the algorithm will extend the path stored the last. With the 1-path algorithm clearly in case of ties the most recent path is eliminated by the purging cycle. Since the correct path has a generally increasing metric it is normally ahead in the tree of the other explored paths. Therefore it appears that when an incorrect path merges with the correct path, the correct path reaches the first the merging point, and hence in case of ties the incorrect path is eliminated. Clearly, such a tie is a proto-error event that may cause an error for the Z-J algorithm but not for the 1-path algorithm.

To summarize, these simulation results verify the predicted improvements over sequential decoding. Both the distribution of the computations and the maximum number of computations per search were reduced by the M-path algorithm. However the distribution of computations remain asymptotically Pareto. As predicted, the error probability was also improved and is nearly equal to that of an optimal decoder even when M is relatively small. The M-path algorithm is "closer" to the optimal decoding than sequential decoding, but remains asymptotically a sequential decoding algorithm. The modification of the Z-J algorithm is very modest, and the determination of the true top node of the stack was found to be quite practical.

All improvements on the performance were obtained at a cost of a larger average decoding effort and stack storage. Although the purging cycle tends to reduce somewhat these two parameters, the importance of the remergers decreases very fast as K increases, and for $K \geq 10$ the exploitation of the convergences may be dropped. Unless some other purging rule (such as the elimination of the nodes stored at the bottom of the stack) is used, in practice the purging cycle may be removed from the generalized algorithm when K becomes large. However in all of our simulation we used the same purging rule. No other purging rule was considered.

In order to reduce further the variability of the computation and/or the average decoding effort, several variants to the M-path algorithm were attempted:

(1) Forced Convergence. The decision rule is modified in such a way that whenever a newly extended path is one branch short of a possible remerger, this path is further extended regardless of whether or not it qualifies for extension.

Compared to the M-path algorithm, this variant is more complex, the average decoding effort is slightly increased and the distribution of the computation is not improved.

(2) Varying the Number of Paths of successive path extension cycles. Regardless of the position of the correct path in the stack, the M-path algorithm extends the same number of paths at each decoding cycle. But when the correct path is at the top of the stack such a decision rule entails the extension of useless paths. By using some of the available information contained in the stack the decision rule could thus be modified to reduce the unnecessary decoding effort. These variants of the M-path algorithm are treated in the next chapter.

CHAPTER V

ADAPTIVE SEQUENTIAL DECODING

In the M-path algorithm no information which could reduce the unnecessary decoding effort is used to modify the decision rule. In this chapter we examine how to use the available information contained in the stack about past decoding, to help improve further the distribution of the computation without unduly increasing the average decoding effort. The object is thus to adapt the decoding effort to the current requirement of the correct path extension.

5.1 Variants to the M-path Algorithm

We recall that with the simultaneous multiple path extension, small dips of the correct path metric are effectively ignored by the decoder. Consequently, when the channel is quiet and there are no dips (or only small ones) of the correct path metric, fewer than M computations may be needed to extend the correct path at each decoding cycle. However, severe noisy periods induce large metric dips of the correct path, sending the decoder into a search mode of operation regardless of M, $M < M\nu$.

One way to reduce the average decoding effort $\bar{C}_{M\text{-path}}$ of the M-path algorithm without altering the distribution of the computation is simply to eliminate those futile extensions during calm periods of the channel. When the channel is quiet the metric of the correct path increases while that of the incorrect paths decrease and in general the cumulative metric of the correct path is much larger than that of the incorrect paths. By extending M paths, some incorrect paths are extended although their metric values are far smaller than that of the correct path. The likelihood of these incorrect paths contributing to the extension of the correct path is thus very small. Extending these incorrect paths needlessly increases the average decoding effort without helping much to improve the distribution of the computation. Therefore, instead of extending M paths regardless of their metric values, it may be preferable to extend <u>up to</u> M paths, provided the metric difference between any of these paths and the top of the stack is within some reasonable bound Γ_Δ. With this decision rule unnecessary computations are avoided during calm periods of the channel noise. However, when the correct path undergoes a large metric drop, its position in the stack also drops away from the top and the full complement of the M paths is needed. If Γ_Δ is chosen in such a way that under these circumstances all M paths are extended, then compared to the M-path algorithm, the distribution of computation is not expected to be degraded. Naturally if Γ_Δ is too small, the average decoding effort $\overline{C}$ decreases but the benefit of the M-path extension is reduced, whereas if it is too large, no improvement on $\overline{C}$ will be obtained. We know that error events occur only when the correct path metric undergoes a dip. Clearly, then by extending fewer than M paths when there is no dip, this procedure should not degrade the error probability corresponding to the M-path algorithm.

This variant to the M-path algorithm was programmed and tested for $K = 6$; $R = 1/2$; $(E_b/N_o) = 3.5$ dB; $M = 4$ and $\Gamma_\Delta = 130$. This value of Γ_Δ was chosen arbitrarily and corresponds to the minimum branch metric value of the correct path. In order to compare the results with the 4-path algorithm, 200 blocks of 500 bits were decoded and the same noise sequence was used.

The cumulative distribution of the number of computations per search is plotted in Fig. (5.1). It shows hardly any difference with the distribution obtained using the 4-path algorithm (see Fig. 4-10b). The probability of the transition from the non-search mode to the search mode, $P(C \geq M+1)$ varied only between $1.82\ 10^{-3}$ and $1.89\ 10^{-3}$ indicating that even with this decision rule, small dips are also ignored. Most important, the average number of computations $\overline{C}$ was reduced significantly while the number of errors remained unchanged as expected. $\overline{C}$ was reduced from 4.049 to 3.532 which represents a saving of over 250 computations (or over 500 stack locations) per decoded block at no cost to the performance. Table 5.1 below summarizes the results obtained for the 4-path algorithm and its variant with $\Gamma_\Delta = 130$.

TABLE 5.1

	$\overline{C}$	Average stack size/block	Convergences		Repeat	Errors
			Stored	Eliminated		
M = 4	4.049	4006	4497	7263	196	45
variant $\Gamma_\Delta = 130$	3.532	3483	3650	5763	169	45

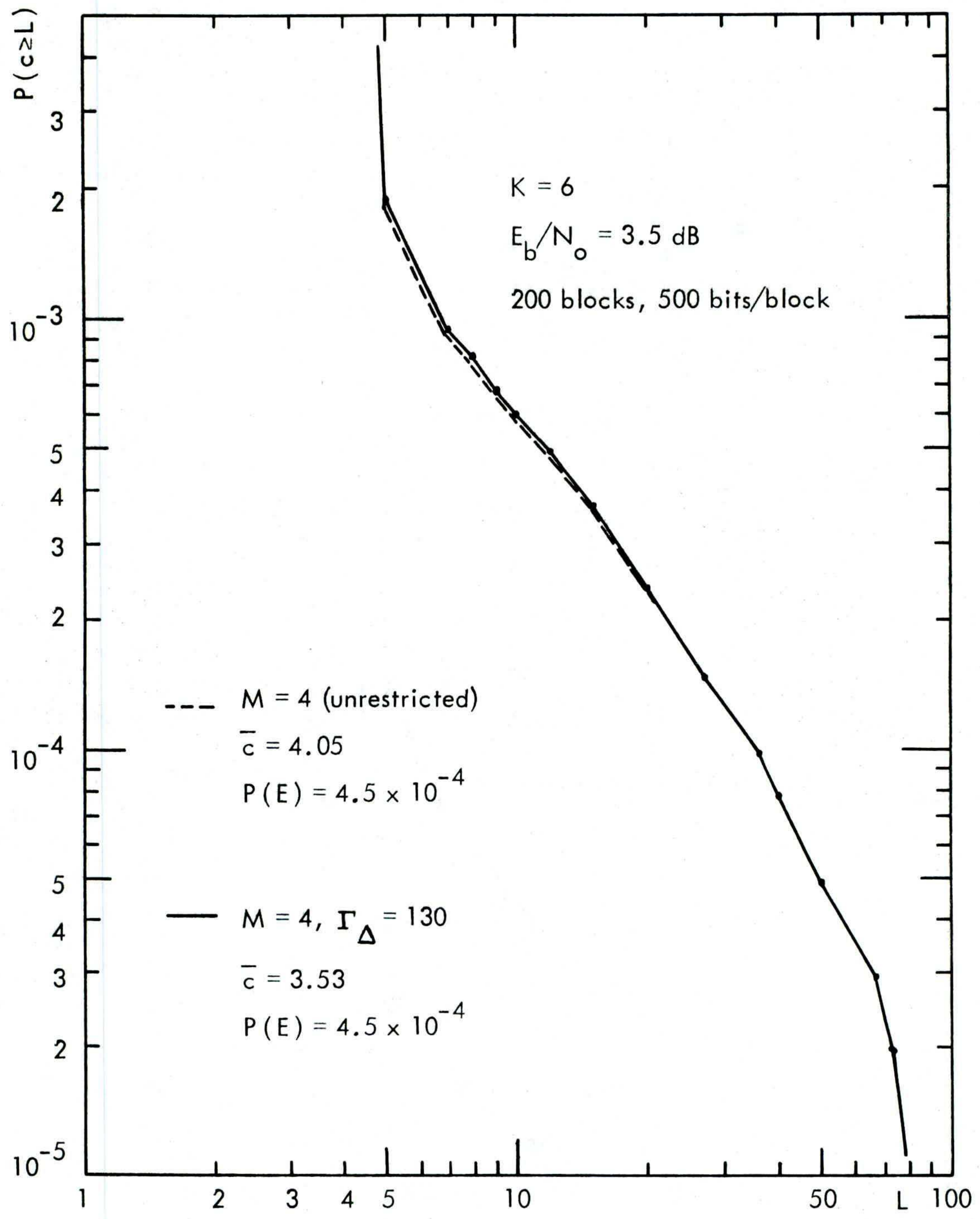


Figure (5.1) Empirical distribution of computations per search for the variant Γ_Δ of the M-path algorithm.

Although no attempt was made to optimize Γ_{Δ}, these results demonstrate that a significant fraction of the computational effort of the M-path algorithm is performed needlessly. These computations do not contribute much to the improvement of neither error probability nor distribution of the computations and hence may be avoided by the simple test on Γ_{Δ}.

Instead of reducing the average decoding effort without degrading the distribution of computations, we may also want to improve upon the distribution of computations without affecting the average decoding effort $\overline{C}_{M\text{-path}}$. Clearly more than M paths must be extended. But to prevent the resulting average decoding effort from exceeding $\overline{C}_{M\text{-path}}$, more paths must be extended only when the noise becomes severe and the correct path metric goes through a dip.

In order to implement this new decision rule, instead of monitoring the correct path metric (which is unknown) the decoder monitors the metric values of the successive top nodes of the stack. As long as these metric values do not decrease, the rule is to extend M paths. Furthermore the restriction on Γ_{Δ} can be added to help reduce the average decoding effort. Whenever the metric of the top node falls below the maximum value ever reached previously, then M', M' > M, paths are extended.

This new decision rule was implemented and tested, but in order to simplify the procedure, instead of comparing the actual metrics of the top nodes, the program compared the levels of the substacks containing these top nodes. As long as the current top node belongs to the highest substack ever reached, the top node metric is not considered in a dip. If the substack containing the current top node is lower than this highest substack, then the top node metric is considered in a dip and M' paths are extended. We call this variant of the M-path algorithm the "variable (M/M') - path algorithm".

In order to compare this algorithm with the M-path algorithm, a variable (4/6)-path was run under strictly the same conditions as the 4-path algorithm. The Γ_Δ restriction was used on the (4/6)-path algorithm only when no dip was detected. Again the same $K = 6$ code, same noise sequence and same 200 blocks of 500 bits with $E_b/N_o = 3.5$ dB were used.

The distribution of the number of computations per search obtained in this run is shown in Fig. (5.2), where the curve for the 4-path algorithm is also plotted for reference. The distribution for the (4/6)-path algorithm shows a substantial improvement over the 4-path algorithm where both algorithms are in the search mode (more than M' computations). Because 6 paths were always extended when a dip of the top node metric is detected, the effect of the Γ_Δ restriction on the average number of computations is greatly reduced. Hence although increased with the new algorithm, a fair comparison of the average decoding effort with the 4-path algorithm using Γ_Δ is difficult. However, compared to the unrestricted 4-path algorithm, we see that the improvement on the distribution of computation was achieved at only a minimal increase of the average decoding effort.

In order to test further this variable path algorithm, a run with 2/6 paths was also made under strictly the same conditions as above. The resulting

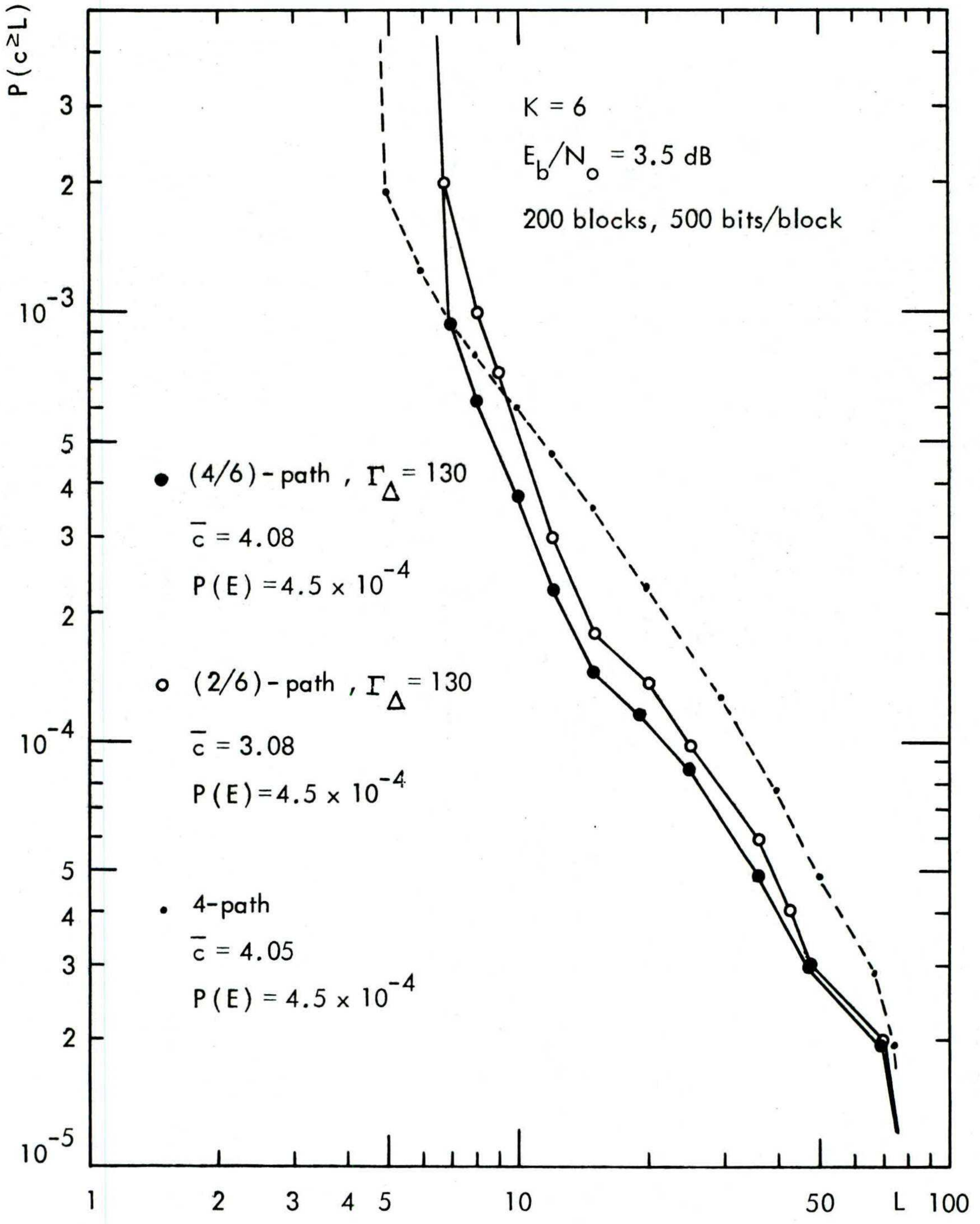


Figure (5.2) Empirical distribution of computations per search for the variable (M/M') - path algorithm.

distribution of computations also plotted in Fig. (5.2), shows again a marked improvement over the distribution of the 4-path algorithm for searches involving more than 10 computations. Most important, this improvement was obtained at a significant reduction of the average decoding effort from 4.08 to 3.08. Hence moderately long searches are definitely better resolved with the variables (2/6)-path algorithm than with a constant 4-path algorithm.

Both (4/6)-path and (2/6)-path algorithms give the same value of $P (C \geq 6)$ which indicates that the occurences of metric dips for the top node of the stack is not much affected by the number of paths regularly extended. The over all distribution of computation for the (2/6)-path algorithm is only slightly worse than for the (4/6)-path algorithms, but for very long searches corresponding to large dips of the correct path metric, both algorithms behave similarly, and asymptotically their distributions are identical.

In conclusion, by modifying the decision rule according to observed values of the top node of the stack, and by properly varying the number of extensions as the decoding progresses, we showed that the overall computational behaviour of the M-path algorithm can be improved. Although increasing the number of simultaneously extended paths will always reduce the variability of the decoding effort, clearly most of these computations are wasted when the channel is quiet. Therefore, always extending the same number of paths regardless of the location of the correct path in the stack induces an unnecessarily large average decoding effort. Viterbi decoding is the limit of this procedure. Since the back searches must be reduced at the smallest possible cost, the above variations of the M-path algorithm suggest that instead of using a fixed decision rule, a clever decoder will use a rule that decides on just enough paths to include the decoded path in the present and future path extension cycles. On what information such a decision rule must be based, is examined in the next section.

5.2 Ideal Adaptive Stack Algorithm

Suppose the current breakout node value $\Gamma_k \min = \underset{j \geq k}{\text{Min}} \; \Gamma_j^{(\underline{U})}$ for all nodes $\underline{U}_k$, $k = 1, 2, \ldots,$ of the correct path $\underline{U}$ were always known to a stack decoder. Using this information, consider the decision rule whereby to decode node $\underline{U}_k$, all nodes stored in the stack with a metric not smaller than $\Gamma_k \min$ are simultaneously extended. Clearly with this procedure the correct path is extended at every decoding cycle, hence there is no backing up, and all extended paths lie at the same tree depth (see Fig. 5.3). Assuming a purging cycle exploits the reconvergence of the paths, we now compare this idealistic algorithm with both the Z-J and Viterbi decoding algorithms.

We first observe that with this algorithm as with the Viterbi algorithm, an error occurs only when the correct path is eliminated in favor of an incorrect path that merged with it with a larger metric. No other error mechanism is introduced by the algorithm, and the paths not extended do not contribute in any way to the decoding of the correct path. Therefore the error performance is that of the Viterbi decoder, and hence this algorithm is an optimum decoding algorithm.

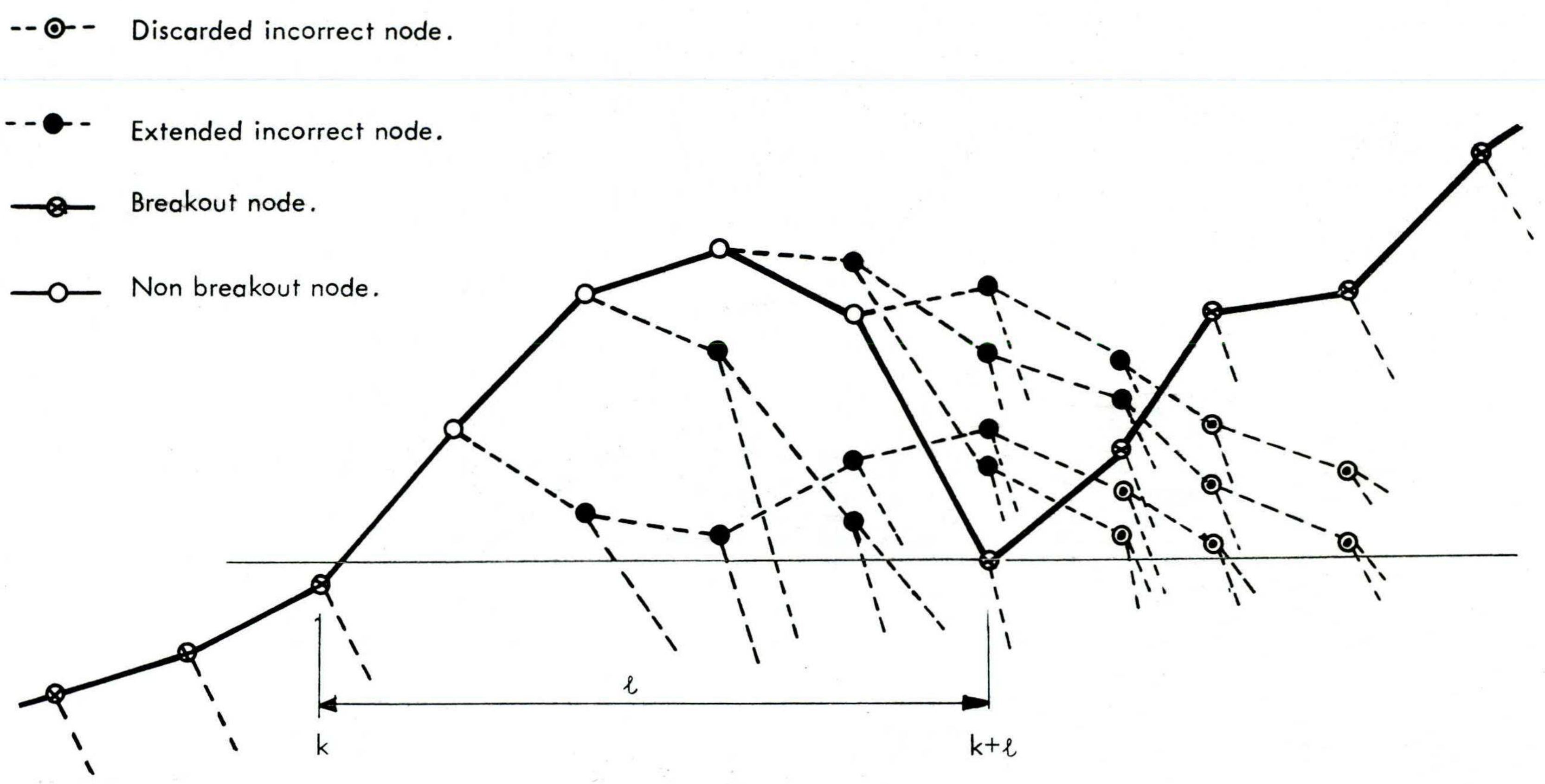


Figure (5.3) Path metric values showing the nodes that are discarded by the ideal stack algorithm.

As shown in Fig. (5.3) the number of computations to decode one branch of the correct path is not constant. For a binary tree there are 2^{K-1} distinct nodes lying at the same level, so the maximum number of computations per decoding cycle is also 2^{K-1}. On the other hand the minimum number of computations is equal to 1. Consequently the average number of computations for decoding one branch of the correct path is

$$1 \le \bar{C}_{ideal} \le 2^{K-1} \tag{5.1}$$

The ideal algorithm is optimum while requiring less computational effort than the Viterbi decoding algorithm. We now show that $\bar{C}_{ideal} \le \bar{C}_{Z-J}$ where $\bar{C}_{Z-J}$ is the average decoding effort of the Z-J algorithm.

Assume that nodes $\underline{U}_k$ and $\underline{U}_{k+\ell}$ of Fig. (5.3) are two consecutive non-adjacent breakout nodes of the correct path $\underline{U}$ and assume that node $\underline{U}_K$ was at the top of the stack when it was extended. As shown in Fig. (5.3), the horizontal line of metric value $\Gamma_{\overline{k+\ell}}^{(U)}$ acts as an absorbing barrier, and no node that crosses this line will be extended by either the sequential or the ideal algorithm. Clearly, assuming no decoding errors, all nodes located above the absorbing line and lying between tree levels ℓ and $(\ell+k)$ will be decoded by both algorithms. However, beyond tree level $(\ell+k)$, a sequential decoder will keep extending these incorrect paths as long as they are above the horizontal absorbing line (a possibly unbounded process), whereas the ideal algorithm will extend only those nodes above the new current breakout node value, $\Gamma\text{min}_{k+\ell+j} \ge \Gamma_{\overline{k+\ell}}^{(U)}$, $j = 1,2,\dots$.

Clearly, in the absence of errors, no computation made by the ideal algorithm may be avoided by a sequential decoding algorithm, but some of the computations of the sequential decoder may be avoided by the ideal algorithm. Specializing for the Z-J algorithm we have

$$\bar{C}_{\text{Ideal}} \leq \bar{C}_{Z-J} \tag{5.2}$$

From the other properties we also have

$$\text{Max } C_{\text{ideal}} \leq C_{\text{Viterbi}} \tag{5.3}$$

$$P(E)_{\text{Ideal}} = P(E)_{\text{Viterbi}} \tag{5.4}$$

Moreover this algorithm is non sequential since it uses information about nodes in the unexplored part of the tree.

Summarizing, for this algorithm, both the average value and distribution of the number of computations to decode one branch are upper bounded by the corresponding values of sequential decoding. The maximum number of computations is limited to M_v, the number of computations per decoding cycle of Viterbi decoding. Furthermore the effective average distance between breakout nodes on the correct path is reduced to 1, just like Viterbi decoding. No other algorithm in the class of generalized stack algorithms can perform better. Therefore we have the theorem.

Theorem 5.1

The best generalized stack algorithm uses a purging rule that exploits the reconvergence of the paths, and a decision rule that extends all nodes stored in the stack with a metric at least equal to the current breakout node value.

This decoding algorithm is of course impossible to implement since the crucial information about the metric value of the current breakout node is not available to any real decoder. However, this ideal algorithm may be regarded as a bound on any decoding strategy using a stack. In fact the computational behaviour of real algorithms is the direct consequence of the particular way with which the uncertainty about the current breakout node value is resolved.

The Viterbi decoding algorithm removes all uncertainty about the current breakout node value by taking a pessimistic point of view, and assumes that at any level, all distinct incorrect paths have a larger metric than the correct path. At each tree level then, the breakout node value is assumed to correspond to the metric value of the (2^{K-1})th node stored in the stack. Therefore 2^{K-1} computations are required for each decoding cycle, leading to a maximum average decoding effort but no variability; the decoder is always in a no-search mode of operation.

Apart from not using a purging cycle, the Z-J algorithm takes an opposite view and assumes that at any tree level, the current breakout node value corresponds to the value of the top node of the stack. Hence, only a single path is extended for each decoding cycle. This leaves a considerable uncertainty about the true current breakout node value which the decoder removes by searching back and forth in the tree. With this assumption, the sequential decoder tends to minimize the average decoding effort, but is not prevented from exploring some incorrect paths over a considerable length and the maximum number of computations to decode one branch of the decoded path is unbounded. With this procedure clearly the decoder is in the no-search mode of operation only when decoding breakout nodes of the correct path. All non-breakout nodes correspond to the search-mode of operation.

The M-path algorithm stands between the two techniques above, and assumes the current breakout node value to correspond to the metric of the Mth node stored in the stack. For $M < 2^{K-1}$ the uncertainty about the true current breakout node value is not entirely resolved, but compared to sequential decoding it is obviously reduced. The back and forth searching is therefore also reduced, reducing with it the variability of the computation. As shown in Fig. (5.3), if at any level between 2 consecutive non-adjacent breakout nodes $\underline{U}_k$ and $\underline{U}_{k+\ell}$ there are no more than M nodes with metric values larger than $\Gamma^{(\underline{U})}_{k+\ell}$, then between these two nodes the M-path decoder is in no-search mode. (Equivalently, the metric dip between nodes $\underline{U}_k$ and $\underline{U}_{k+\ell}$ has been ignored by the M-path decoder). Clearly then, it is the search mode of operation that causes the undesirable variability of the computational effort, and any procedure that keeps the decoder away from that mode of operation will reduce the variability of the computation.

In all of these practical algorithms no useful information imbedded in the stack is used to reduce unnecessary computations. The effect of the constant path extension is to exchange the variability with the average value of the decoding effort. It appears that any stack algorithm which always extend the same number of paths is bound to improve either the average decoding effort or its variability, and as one parameter is improved, the other one is degraded. The variants of the M-path algorithm represent one step toward the relaxing of the constant path extension, and attempt to use some information about past stack behaviour to improve the overall decoding effort.

In approaching the operation of the ideal stack algorithm, the decision rule of any real stack algorithm cannot be based on the current breakout node value. However, a practical decoder can use the available information imbedded in the stack to adapt each path extension cycle to the current requirement of the correct path metric extension.

5.3 An Adaptive Stack Algorithm

Short of using the current breakout node values, we now present a practical adaptive algorithm (Haccoun & Ferguson, 1973) which determines the number of paths to be extended according to the relative metric values of the top nodes of the stack.

It is observed that at any tree level, when the accumulated metric of the correct path is maximum over its metric past, the correct path is usually at the top of the stack. If we assume this top node to be a breakout node, then only a single computation is necessary for its decoding. On the other hand, whenever the correct path metric drops, it corresponds to non breakout nodes and is usually not at the top of the stack. Depending on the size of the dip, a varying number of paths must be extended in the path-extension cycle to include the correct path.

The correct path is obviously not known by the decoder, but for the purpose of determining the number of paths to be extended in each path extension cycle, the values of the top node of the stack may be used instead. As long as the top node metric is the largest over all preceding top node values, a single path is extended. Whenever the metric of a new top node drops below the largest value ever reached, depending on the size of that drop, a variable number of paths is extended. This adaptive procedure is obviously not the ideal one, but is has the advantage of being simple to implement since monitoring the metrics of the top nodes of the stack as the decoding proceeds presents hardly any difficulty.

Let $\Gamma_n^{(Top)}$, $n = 1, 2, \ldots,$ be the metric value of the top node of the stack for the current decoding cycle, and let $\Gamma_n^{(max)}$ be the largest metric value ever obtained up to the present time,

$$\Gamma_n^{(\max)} = \operatorname{Max}_{\ell} \; \Gamma_\ell^{(\mathrm{Top})} \qquad \ell < n, \; n = 1, 2, \ldots \tag{5.5}$$

Define the current dip as

$$D = \Gamma_n^{(\max)} - \Gamma_n^{(\mathrm{Top})} \tag{5.6}$$

If $D \le 0$, there is no dip, only the top node is extended, and the maximum metric value is updated

$$\Gamma_{n+1}^{(\max)} = \Gamma_n^{(\mathrm{Top})} \tag{5.7}$$

If $D > 0$, a correct path metric dip is assumed and the decoder extends the $M(D)$ highest nodes in the stack, where $M(D)$ is a non-decreasing function of D. The maximum metric value is then unchanged

$$\Gamma_{n+1}^{(\max)} = \Gamma_n^{(\max)} \tag{5.8}$$

This algorithm is a very simple extension of the variable (M/M')-path algorithm. Taking for reference the maximum metric value observed over the entire past, the decoding effort is adapted to the size D of the apparent correct path metric dip (i.e. the dips of the top nodes). The exact number of paths to be extended will depend on the function $M(D)$ which is recognized as the decision rule. Since $M(D)$ is non-decreasing with D, we expect the distribution of the computation to be improved over sequential decoding without increasing too severely the average decoding effort. Moreover, as a consequence of the large number of paths simultaneously extended during large dips, the error probability should also improve over sequential decoding.

The algorithm attempts to extend the correct path at every decoding cycle, but the uncertainty about the true breakout node value is not entirely removed. Therefore the algorithm is not expected to remain all the time in a non-search mode of operation. Hence, occasionally the correct path will not be included in the $M(D)$ extended paths and the algorithm will enter a search-mode of operation. Consequently, an asymptotic sequential decoding behaviour is expected. If $M(D)$ is subjected to some maximum value M_{max}, a search mode could correspond to more than M_{max} computations to decode one branch of the correct path. The average decoding effort will depend on the particular function $M(D)$ chosen, and the probability to enter a search mode of operation will depend on both $M(D)$ and M_{max}. Hence $M(D)$ should be chosen to yield the smallest search mode probability at the smallest average decoding effort.

5.4 Model of the Adaptive Stack Algorithm

Consider the adaptive stack algorithm of the previous section using some non-decreasing function $M(D)$. Assume that $M(D)$ is such that if between two consecutive breakout nodes the dip of the correct path metric is less than some value H, then the algorithm maintains a non-search mode of operation. Under these conditions, between the two breakout nodes, all non-breakout nodes of the correct path are decoded by a single decoding cycle. These nonbreakout nodes become "pseudo breakout nodes" and the whole correct path metric dip is ignored by the decoder. Whenever the correct path metric dips by more than H, then the decoder does not extend enough paths to guarantee the decoding of the correct path at every decoding cycle and thus enters in a search mode of operation. Under these conditions the non-breakout nodes of the correct path remain non-breakout nodes.

The nodes of the correct path may be classified in two categories: the search nodes requiring possibly more than one decoding cycle to be decoded, and the no search nodes requiring exactly one decoding cycle. As shown in Fig. (5.4) the no-search nodes are the breakout nodes and pseudo-breakout nodes of the correct path. Moreover, between two consecutive non-adjacent breakout nodes, the nodes are either all pseudo-breakout (no-search nodes) or all non-breakout (search nodes).

When decoding no-search nodes the decoder is in a no-search mode of operation, and when decoding search nodes, it is in a search mode of operation. An important parameter is then the probability that starting from a breakout node, the decoder enters into a search mode of operation. We call this probability the "search mode probability".

The average distance d_o between breakout nodes of the correct path was shown to be a measure of the variability of the decoding effort for sequential decoding. With the adaptive sequential decoding algorithm, no-search nodes behave like breakout nodes, and clearly the average distance between no-search nodes will be smaller than d_o. The reduction will naturally depend on H.

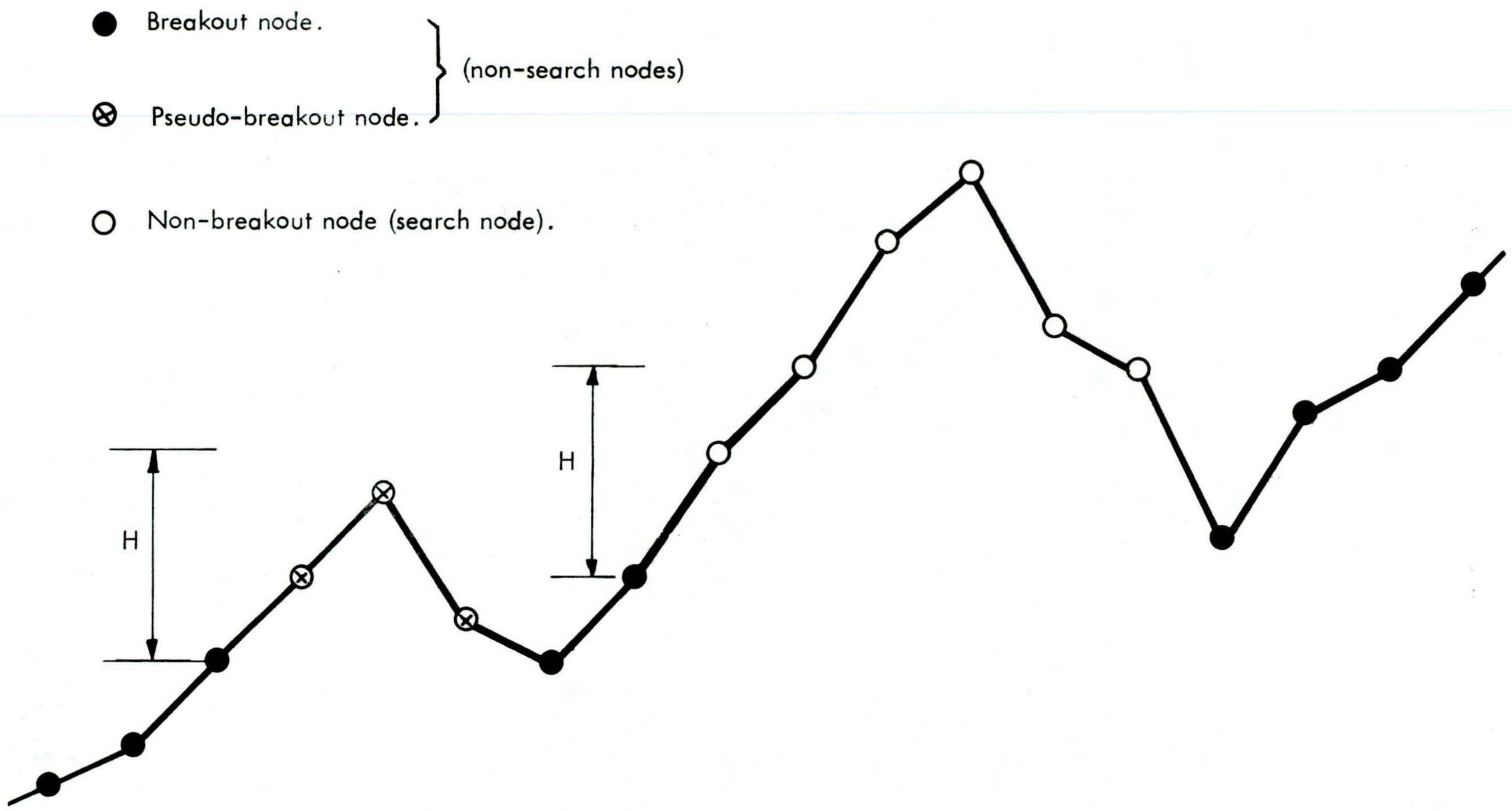


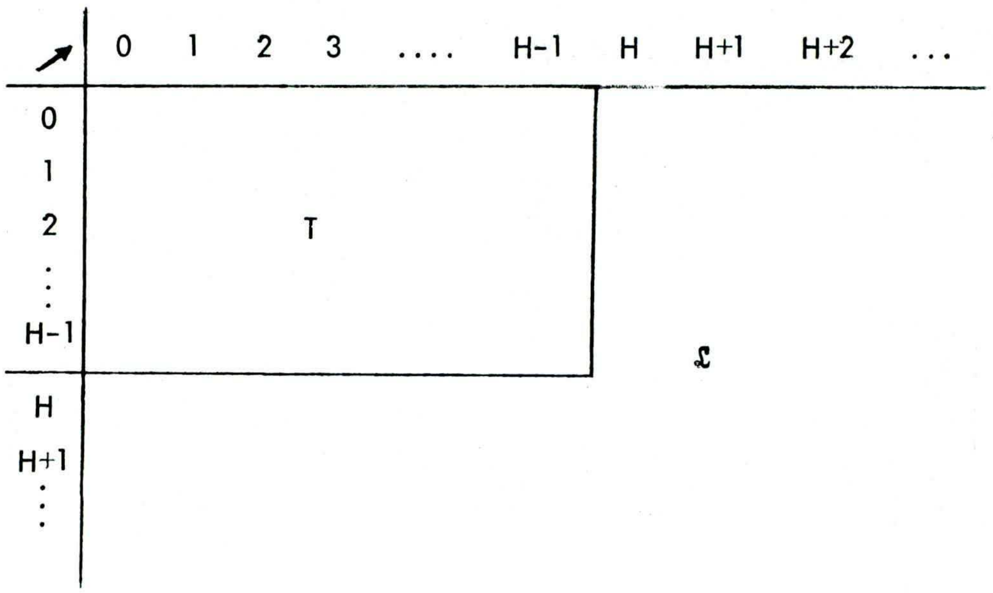
Figure (5.4) Search and non-search nodes on the correct path.

Using the Markov chain model of the correct path, we now establish the expressions for the average distance between no-search nodes and for the search mode probability of this model of the adaptive algorithm.

5.4.1 Average Distance Between No-Search Nodes

We recall that in the Markov chain model of the metric differences on the correct path, the states Δ_K are the metric differences between a node and its smallest succeeding value on the correct path. Under the assumptions of our adaptive algorithm all no-search nodes have a state smaller than H. Consequently, if the decoder starts from the zero state (breakout node) and returns to it without ever visiting any state at or beyond H, all states in the path are no-search nodes.

Let us partition the transition probability matrix of the Markov chain into a submatrix T containing all states smaller than H, and let $\mathfrak{L}$ be the set of all other states not in T.



Define

$$_{\mathfrak{L}}f_{oo}^{(n)} = P\,[\,\Delta_n = 0,\ \Delta_j \neq 0,\ \Delta_j \notin \mathfrak{L},\ \ j = 1, 2, \ldots n-1 \mid \Delta_o = 0] \tag{5.9}$$

as the first return probability to the zero state without every visiting any state belonging to $\mathfrak{L}$ (the so-called taboo set).

Since all pseudo-breakout nodes between two consecutive breakout nodes are adjacent, and since they require only one decoding cycle to be decoded, the average distance between the no-search nodes is

$$H^d_o = \sum_{n=1}^{\infty} 1\ .\ _{\mathfrak{L}}f_{oo}^{(n)} + \sum_{n=1}^{\infty} n\ .\ {}^{c}f_{oo}^{(n)} \tag{5.10}$$

where ${}^{c}f_{oo}^{(n)}$ is the complement to $_{\mathfrak{L}}f_{oo}^{(n)}$, that is

$$f_{oo}^{(n)} = {}_{\mathfrak{L}}f_{oo}^{(n)} + {}^{c}f_{oo}^{(n)} \tag{5.11}$$

where $f_{oo}^{(n)}$ is the first return probability to state 0. Eq. (5.11) is valid since the events of returning to state 0 can be classified into 2 disjoint sets depending on whether all states visited "en route" belong to T or not.

Defining

$$h_1 = \sum_{n=1}^{\infty} {}_{\mathfrak{L}}f_{oo}^{(n)} \tag{5.12}$$

and

$$h_n = {}^{c}f_{oo}^{(n)} \quad , \ n > 1 \tag{5.13}$$

1-Path Sequential Decoding: Since $H = 1$, then the taboo set $\mathcal{L}$ consists of all non-breakout nodes and clearly ${}_{\mathcal{L}}f_{oo}^{(n)} = 0$ if $n > 1$. Hence from Eq. (5.17) we get

$$d_o - {}_1d_o = 0 \tag{5.18}$$

as required.

Viterbi Decoding: All nodes are no-search nodes, and the taboo set $\mathcal{L}$ is empty. Therefore,

$${}_{\mathcal{L}}f_{oo}^{(n)} = f_{oo}^{(n)} \tag{5.19}$$

and hence

$$\sum_{n=1}^{\infty} f_{oo}^{(n)} = 1 \tag{5.20}$$

Therefore, Eq. (5.16) becomes

$${}_{\infty}d_o = d_o + 1 - \sum_{n=1}^{\infty} n\, f_{oo}^{(n)} \tag{5.21}$$

or

$${}_{\infty}d_o = d_o + 1 - d_o = 1 \tag{5.22}$$

As expected the average distance between the no-search nodes (that is all the nodes) of the correct path is equal to 1 for the Viterbi decoder. The same result applies to the ideal decoder.

Naturally, the average distance between no-search nodes ${}_H d_o$ for our adaptive algorithm is bounded by 1 and d_o and the reduction over sequential decoding is given by the infinite sum of Eq. (5.17). We now show that this infinite sum can be evaluated from the set of transition probabilities of the Markov chain of the correct path.

Suppose the Markov chain admits in general J transitions away from the origin with probabilities p_i, $i = 0, 1, \ldots J$, and Q transitions toward the origin with probabilities q_i, $i = 1, 2, \ldots Q$. Without loss of generality let $J > Q > H$, then the transition probability matrix of the chain is

$$\begin{array}{c|cccccccccc}
\nearrow & 0 & 1 & 2 & \cdots & H-1 & H & \cdots & J & J+1 & \cdots \\
\hline
0 & p_o + \sum_{i=1}^{Q} q_i & p_1 & p_2 & \cdots & p_{H-1} & p_H & \cdots & p_J & 0 & \\
1 & \sum_{i=1}^{Q} q_i & p_o & p_1 & \cdots & p_{H-2} & p_{H-1} & \cdots & p_{J-1} & p_J & \\
2 & \sum_{i=2}^{Q} q_i & q_1 & p_o & \cdots & p_{H-3} & p_{H-2} & \cdots & p_{J-2} & p_{J-1} & \\
\vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & \\
H-1 & \sum_{i=H-1}^{Q} q_i & q_{H-2} & q_{H-3} & \cdots & p_o & p_1 & \cdots & p_{J-H} & & \\
H & \sum_{i=H}^{Q} q_i & q_{H-1} & q_{H-2} & \cdots & p_1 & p_o & \cdots & & & \\
\vdots & & & & & & & & & &
\end{array}$$

Writing the transition probabilities in general as

$$P_{ij} = \Pr (\Delta_{n+1} = j \mid \Delta_n = i) \tag{5.23}$$

then from Eq. (5.9) we have the following relations

$$_{\mathfrak{L}}f_{oo}^{(n+1)} = \sum_{j=1}^{H-1} P_{oj} \; _{\mathfrak{L}}f_{jo}^{(n)} \qquad n = 1, 2, \ldots \tag{5.24}$$

and

$$_{\mathfrak{L}}f_{\ell o}^{(n+1)} = \sum_{j=1}^{H-1} P_{\ell j} \; _{\mathfrak{L}}f_{jo}^{(n)} \qquad n = 1, 2, \ldots \tag{5.25}$$

where

$$_{\mathfrak{L}}f_{jo}^{(n)} = \Pr [\Delta_n = 0, \Delta_i \neq \mathfrak{L}, \Delta_i \neq 0, i = 1,2,\ldots n-1 \mid \Delta_o = j] \tag{5.26}$$

is the first entrance probability to state 0 from state j in exactly n steps, without every entering the taboo set $\mathfrak{L}$ "en route".

The use of Eqs. (5.24) and (5.25) in evaluating $\sum_{n=1}^{\infty} {_{\mathfrak{L}}f_{oo}^{(n)}}$ and $\sum_{n=1}^{\infty} n \; {_{\mathfrak{L}}f_{oo}^{(n)}}$ of Eq. (5.17) suggests the following matrix representation.

Define the vector

$$\underline{F}_o = \begin{bmatrix} _{\mathfrak{L}}f_{oo}^{(1)} \\ _{\mathfrak{L}}f_{10}^{(1)} \\ _{\mathfrak{L}}f_{20}^{(1)} \\ \vdots \\ _{\mathfrak{L}}f_{H-1,0}^{(1)} \end{bmatrix} = \begin{bmatrix} P_{oo} \\ P_{10} \\ P_{20} \\ \vdots \\ P_{H-1,0} \end{bmatrix} \tag{5.27}$$

and let [A] be the H x H matrix whose 1st column is zero

$$[A] = \begin{bmatrix} 0 & p_1 & p_2 & \cdots & p_{H-1} \\ 0 & p_o & p_1 & \cdots & p_{H-2} \\ 0 & q_1 & p_o & & p_{H-3} \\ 0 & q_2 & q_1 & & p_{H-4} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & q_{H-2} & \cdots & q_1 & p_o \end{bmatrix} \tag{5.28}$$

than from Eqs. (5.24) and (5.25) we obtain

$$\underline{F}_1 = \begin{bmatrix} \mathfrak{L}f^{(2)}_{oo} \\ \mathfrak{L}f^{(2)}_{10} \\ \vdots \\ \mathfrak{L}f^{(2)}_{H-1,0} \end{bmatrix} = [A]\,\underline{F}_o \tag{5.29}$$

Likewise we obtain

$$\underline{F}_2 = \begin{bmatrix} \mathfrak{L}f^{(3)}_{oo} \\ \mathfrak{L}f^{(3)}_{10} \\ \vdots \\ \mathfrak{L}f^{(3)}_{H-1,0} \end{bmatrix} = [A]\,\underline{F}_1 = [A]^2\,\underline{F}_o \tag{5.30}$$

and in general

$$\underline{F}_n = [A]^n\,\underline{F}_o \tag{5.31}$$

Let the vector $\underline{F}_n$ be the (n+1)th column of a matrix $[B]$ having H rows and an infinite number of columns

$$[B] = [\underline{F}_o \vdots \underline{F}_1 \vdots \underline{F}_2 \vdots \ldots \vdots \underline{F}_n \vdots \ldots] \tag{5.32}$$

or

$$[B] = [\underline{F}_o \vdots [A]\underline{F}_o \vdots [A]^2 \underline{F}_o \vdots \ldots \vdots [A]^n \underline{F}_o \vdots \ldots] \tag{5.32}$$

The elements of b_{1i} of the first row of $[B]$ are the $\{ {}_{\mathfrak{L}}f_{oo}^{(n)} \}$ of interest. Consequently

$$\sum_{n=1}^{\infty} {}_{\mathfrak{L}}f_{oo}^{(n)} = \sum_{i=1}^{\infty} b_{1i} \tag{5.33}$$

Factoring the common matrix $[A]$, Eq. (5.32) becomes

$$[B] = [(I \vdots [A] \vdots [A]^2 \vdots \ldots \vdots [A]^n \vdots \ldots) \underline{F}_o] \tag{5.34}$$

The matrix power series $I + [A] + [A]^2 + \ldots [A]^n + \ldots$ will converge to $(I - [A])^{-1}$ if the eigenvalues of $[A]$ are less than 1. Since $[A]$ is the truncation of a stochastic matrix, the sum converges. Letting

$$\underline{X} = [I - [A]]^{-1} \underline{F}_o \tag{5.35}$$

Then the sum of the elements of the first row of $[B]$ is the first element x_1 of the vector $\underline{X}$. Thus

$$\sum_{n=1}^{\infty} {}_{\mathfrak{L}}f_{oo}^{(n)} = x_1 \tag{5.36}$$

Likewise each element of the sum $\sum_{n=1}^{\infty} n \; {}_{\ell}f_{oo}^{(n)}$ is an element of the first row of the infinite matrix

$$[B]^* = [\underline{F}_o \vdots 2[A]\underline{F}_o \vdots 3[A]^2\underline{F}_o \vdots \ldots \vdots (n+1)[A]^n\underline{F}_o \vdots \ldots] \quad (5.37)$$

$$= [(I \vdots 2[A] \vdots 3[A]^2 \vdots \ldots \vdots (n+1)[A]^n \vdots \ldots)\underline{F}_o] \quad (5.38)$$

The sum of the columns converges to the vector

$$\underline{Y} = ([I - [A]]^{-1})^2 \underline{F}_o \quad (5.39)$$

and hence

$$\sum_{n=1}^{\infty} n \; {}_{\ell}f_{oo}^{(n)} = y_1 \quad (5.40)$$

where y_1 is the first element of the vector $\underline{Y}$.

Observing that $\underline{F}_o$ and $[A]$ are known, and that H is finite, the computations of Eq. (5.35) and (5.39) can easily be carried out on a computer.

We have therefore proved the theorem.

Theorem 5.2

If for an adaptive stack algorithm all non-search nodes have a metric dip smaller than some value H, the effective reduction of the average distance between breakout nodes is

$$d_o - {}_Hd_o = y_1 - x_1 \quad (5.41)$$

where y_1 and x_1 are the first elements of the vectors $([I - [A]]^{-1})^2 \underline{F}_o$ and $[I - [A]]^{-1} \underline{F}_o$ respectively.

5.4.2 Search Mode Probability

Keeping the same assumptions for our adaptive stack algorithm, we now determine the probability P_H that starting from a breakout node, the decoder enters into a search mode of operation. Partitioning the transition probability matrix of the Markov chain into the sub-matrix T and the set of states $\mathcal{L}$ whose values are equal or larger than H, P_H is the probability of crossing over into the set $\mathcal{L}$ before returning to state 0, given the starting state was the zero state. Lumping all states belonging to the set $\mathcal{L}$ into a unique absorbing state, P_H is then the probability of being absorbed before returning to state 0. Although P_H can be determined this way, its complement P_H^c,

$$P_H^c = 1 - P_H \tag{5.42}$$

is readily obtained if we observe that it is the probability to return to state zero before every crossing over to the taboo set $\mathcal{L}$. This probability was encountered in the derivation of the average distance between no-search node, that is

$$P_H^c = \sum_{n=1}^{\infty} {}_{\mathcal{L}}f_{oo}^{(n)} \tag{5.43}$$

where ${}_{\mathcal{L}}f_{oo}^{(n)}$ is given by Eq. (5.9). Therefore

$$P_H = 1 - \sum_{n=1}^{\infty} {}_{\mathcal{L}}f_{oo}^{(n)} \tag{5.44}$$

We can check this expression for the two extreme cases:

<u>Sequential Decoding (H = 1)</u>

Since ${}_{\mathcal{L}}f_{oo}^{(n)} = 0 \quad n > 1$, then

$$P_1 = 1 - [p_o + \sum_{i=1}^{Q} q_i] \tag{5.45}$$

which is the probability of not remaining at the zero state.

<u>Viterbi Decoding (H = ∞)</u>

In this case the taboo set $\mathcal{L}$ is empty and clearly

$$P_\infty = 1 - \sum_{n=1}^{\infty} f_{oo} = 0 \tag{5.46}$$

As expected the probability of entering a search mode is zero.

Consequently, using the closed form solution of Eq. (5.44) given by Eq. (5.36), we have the theorem.

<u>Theorem 5.3</u>

Starting from a breakout node, the probability for the decoder to enter a search-mode of operation is given by

$$P_H = 1 - x_1 \tag{5.47}$$

where x_1 is the first element of the vector

$$\underline{X} = [I - [A]]^{-1} \underline{F}_o$$

Observing that the probability of returning to zero from any state $j \leq H-1$ before reaching the taboo set $\mathcal{L}$ is the $(j+1)$th element of the vector $\underline{X}$, a corollary to the theorem is,

Corollary

The probability to enter a search mode of operation from any no-search node of state $j \leq H-1$ is

$$P_{H,j} = 1 - x_{j+1} \tag{5.48}$$

where x_{j+1} is the $(j+1)$th element of the vector $\underline{X}$.

Computation of P_H for different values of H was carried out for the D.M.C. using a 3 bit quantization described in Section 4.5.1. For a rate 1/2 code, the branch metric values and their probability assignment can be easily obtained from Table 4.1 and Table 4.3. The set of probabilities $\{q_i\}$ and $\{p_i\}$ of the associated Markov chain are given in Table 5.2 and Table 5.3 for $E_b/N_o = 3.5$dB and 3.0 dB, respectively. The calculated search mode probabilities are shown in Fig. (5.5) for different values of H. From these curves, the probability P_H appears to be exponentially decreasing with H. The probability to enter a search mode of operation increases as (E_b/N_o) decreases, and as expected, the curve of P_H for 3.0 dB appears above the curve for 3.5 dB.

TABLE 5.2

Transition Probabilities $E_b/N_o = 3.5$ dB

$q_8 = 0.477151$	$P_{36} = .001467$
$q_7 = 0.207368$	$P_{40} = .000900$
$q_6 = 0.022530$	$P_{46} = .000276$
$q_3 = 0.127200$	$P_{48} = 4.32\times10^{-4}$
$q_2 = 0.027640$	$P_{60} = 1.877\times10^{-3}$
	$P_{61} = 4.08\times10^{-4}$
$P_2 = .008477$	$P_{62} = 1.62\times10^{-4}$
$P_5 = .061074$	$P_{65} = 2.50\times10^{-4}$
$P_6 = .013271$	$P_{73} = 1.2\times10^{-4}$
$P_{10} = .008141$	$P_{78} = 2.4\times10^{-5}$
$P_{18} = .001954$	$P_{87} = 4.5\times10^{-5}$
$P_{19} = .022951$	$P_{103} = 1.32\times10^{-5}$
$P_{20} = .004987$	$P_{128} = 2.0\times10^{-6}$
$P_{24} = .003059$	
$P_{32} = .001469$	
$P_{35} = .006750$	

TABLE 5.3

Transition Probabilities $E_b/N_o = 3.0$ dB

$q_8 = .4364452$	$P_{32} = 8.062\text{x}10^{-3}$
$q_7 = .2101586$	$P_{33} = 1.941\text{x}10^{-3}$
$q_6 = .0252991$	$P_{37} = 1.2405\text{x}10^{-3}$
$q_3 = .1343121$	$P_{42} = 3.965\text{x}10^{-4}$
$q_2 = .032372$	$P_{45} = 6.206\text{x}10^{-4}$
	$P_{56} = 2.3558\text{x}10^{-3}$
$P_2 = .0103333$	$P_{61} = 3.625\text{x}10^{-4}$
$P_5 = .0671917$	$P_{69} = 1.813\text{x}10^{-4}$
$P_6 = .016177$	$P_{72} = 3.72\text{x}10^{-5}$
$P_{10} = .010338$	$P_{81} = 7.1\text{x}10^{-5}$
$P_{17} = .026309$	$P_{96} = 2.18\text{x}10^{-5}$
$P_{18} = .008920$	$P_{120} = 3.2\text{x}10^{-6}$
$P_{22} = .004048$	
$P_{30} = .002025$	

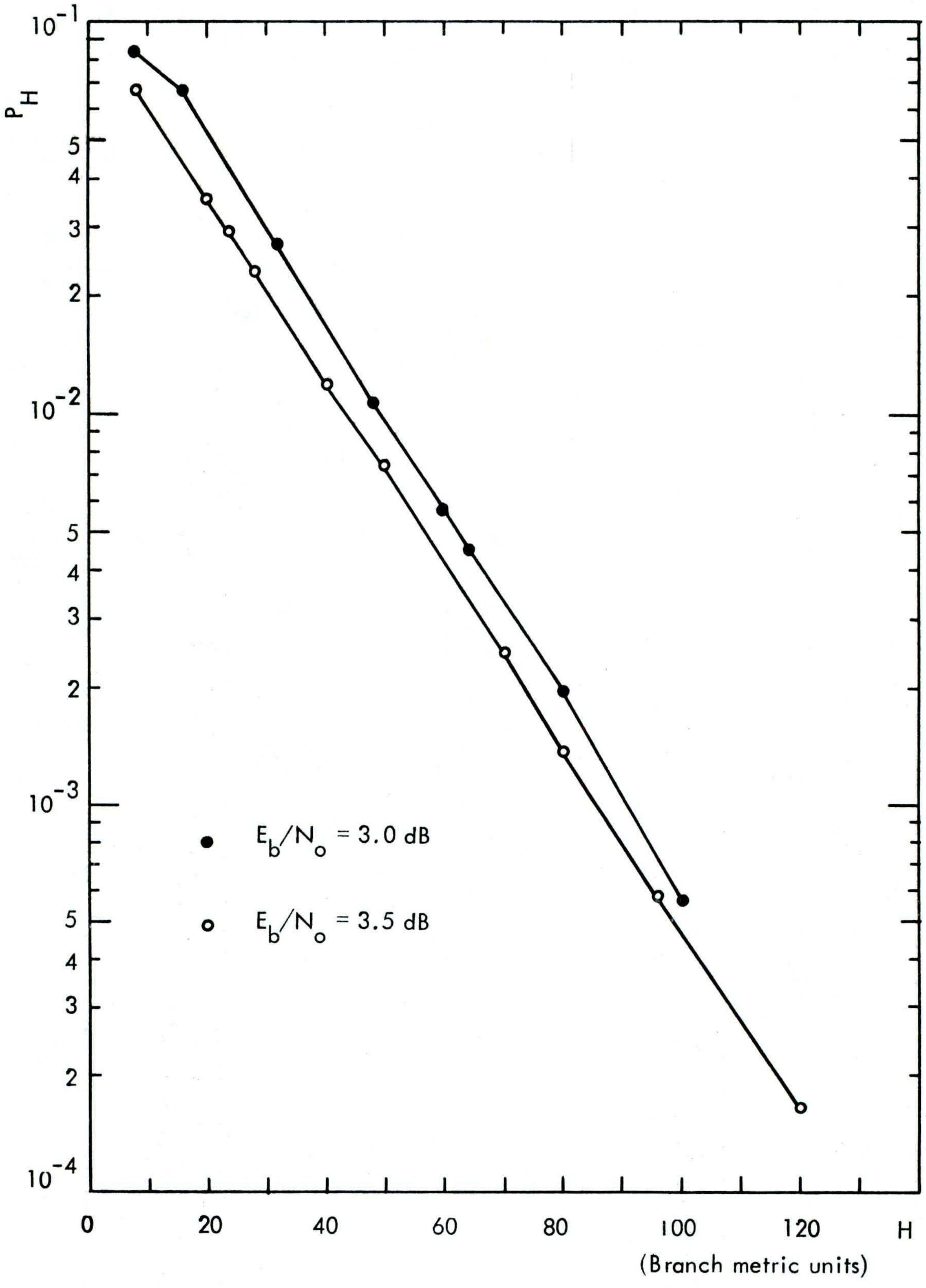


Figure (5.5) Calculated search mode probabilities.

The exponential dependency of P_H on H is not unexpected when we consider P_H as the probability that the correct path metric dips by H or more. The Markov chain approach has the advantage of including in the calculation the particular quantization scheme and the actual set of branch metrics used.

5.5 Computer Simulation Results

The adaptive stack algorithm described in Section (5.3) was programmed as an extension of the (M/M')-path algorithm. The same purging rule of exploiting the reconvergence of the paths was used on the purging cycle. The decision rule is to extend $M(D)$ paths at each path extension cycle, where $M(D)$ may be any function specified by the user. In order to simplify the procedure, the current dip value D of Eq. (5.6) was quantized in substack units: Instead of comparing the exact values $\Gamma_n^{(\max)}$ and $\Gamma_n^{(\mathrm{Top})}$ to determine D, the program compared the levels of their corresponding substacks. The simulation runs were performed with only the following linear functions of D:

$$M(D) = \begin{cases} a + D, & 1 \le a \le 6, \quad D > 0 \\ 1 & , \quad D \le 0 \end{cases} \tag{5.49}$$

and

$$M(D) = \begin{cases} a + 2D, & 0 \le a \le 2, \quad D > 0 \\ 1 \ , & D \le 0 \end{cases} \tag{5.50}$$

For each case, $M(D)$ was limited to a maximum number $M_{\max}$.

The top node of the stack is determined exactly whereas the other nodes of a same path extension cycle are picked at random from the highest non-vacant substacks.

In order to compare the results with those obtained with previous algorithms, the same codes, channels and noise sequences were used over runs of the same lengths, and again a single path was extended when the decoder failed to penetrate deeper in the tree.

Fig. (5.6) - (5.9) show the empirical distributions of the number of computations per search obtained for the rate 1/2 code of constraint length $K = 6$, $(E_b/N_o) = 3.5$ dB and the functions $M(D)$ of Eq. (5.49) and Eq. (5.50).

For convenience the distribution for the single-path sequential decoder run under identical conditions is also plotted in Fig. (5.6). As expected the adaptive algorithm shows a great improvement over the single path sequential decoding algorithm at a cost of a very modest increase of the average decoding effort. For each curve the no-search $(C \leq M_{max})$ and the search $(C > M_{max})$ modes of operation are well displayed. For $(C \leq M_{max})$, the distribution decreases very steeply, whereas for $(C > M_{max})$ the straight line Pareto behaviour of sequential decoding is quite apparent. The search mode probability of the model of the algorithm corresponds to the transition point between the two modes of operation.

From Fig. (5.6) and Fig. (5.7), we note that the distribution of the computation and the search mode probability both improve with the constant "a" of the function $M(D)$. This is simply due to the fact that a given metric drop

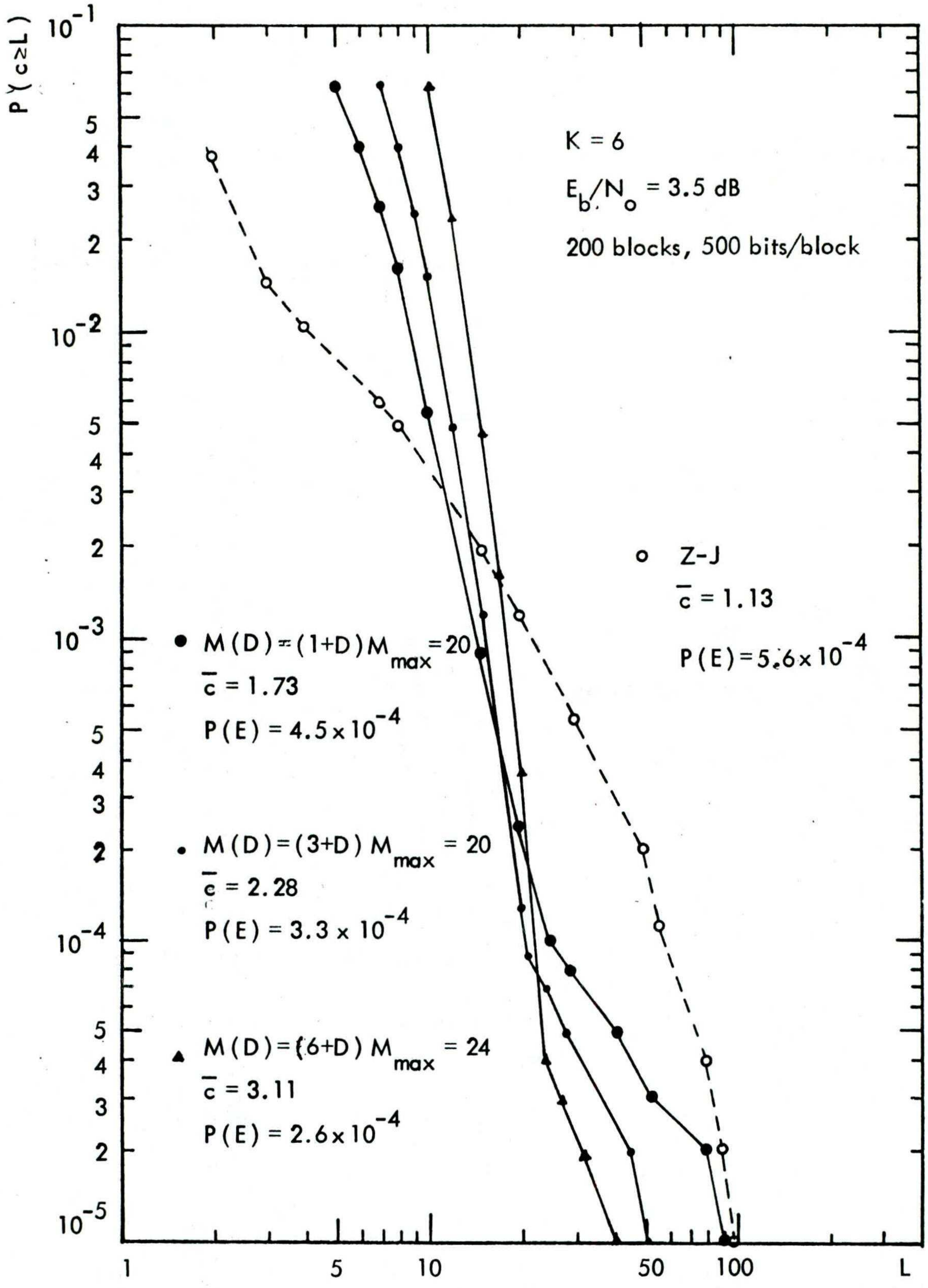


Figure (5.6) Empirical distribution of computations per search for the Adaptive algorithm.

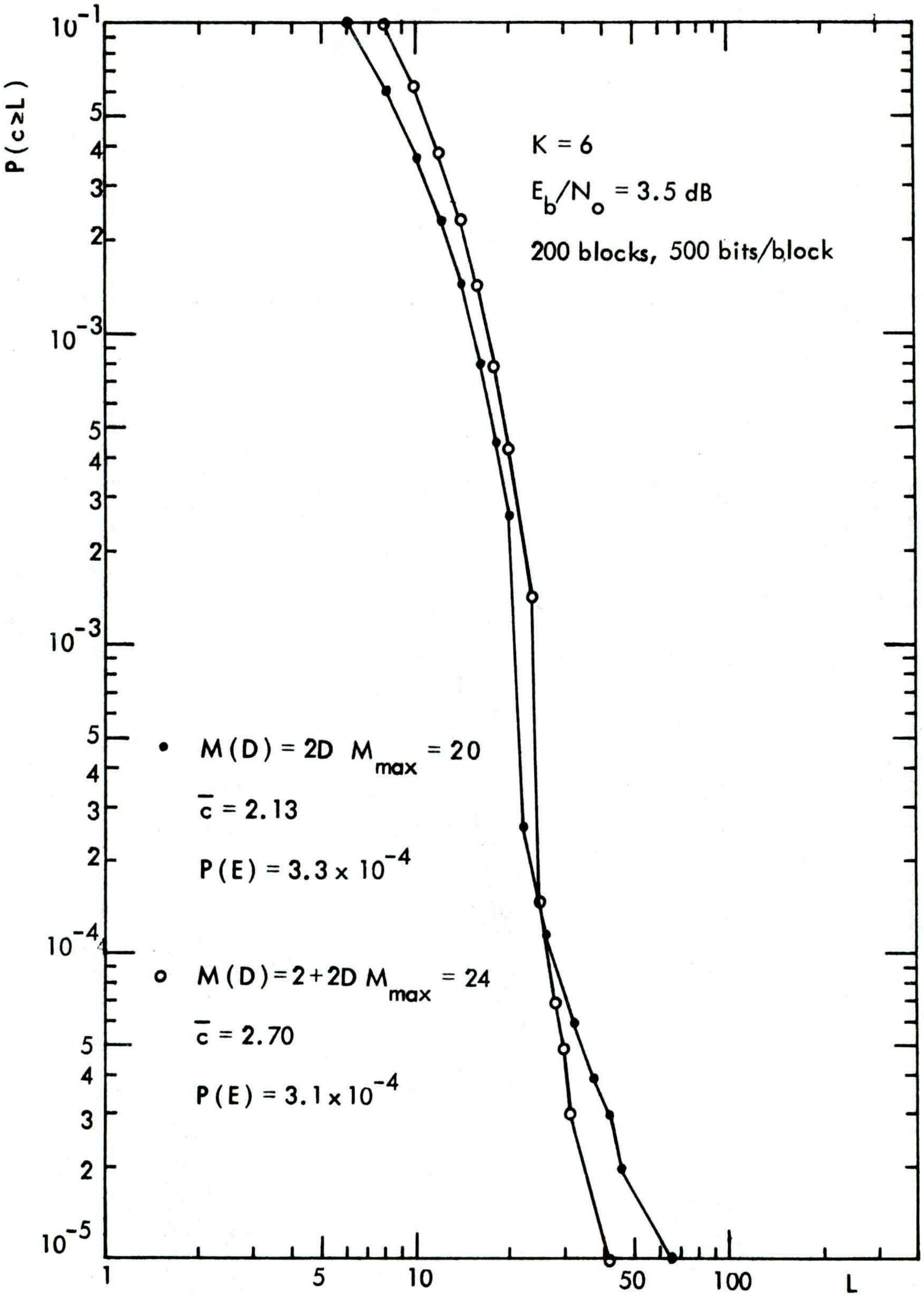


Figure (5.7) Empirical distribution of computations per search for the adaptive algorithm.

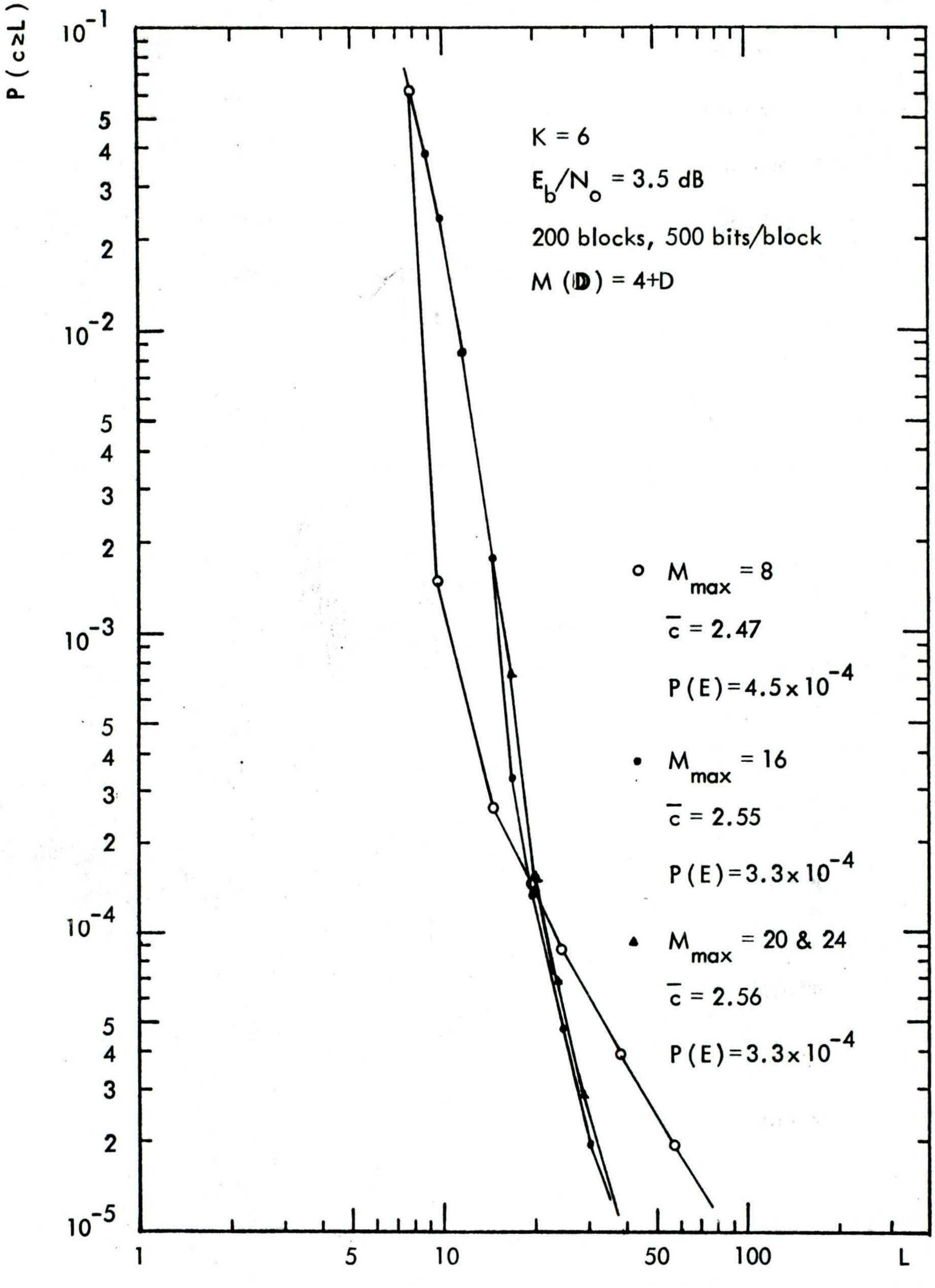


Figure (5.8) Empirical distribution of computations per search for the Adaptive algorithm. Effect of M_{max}.

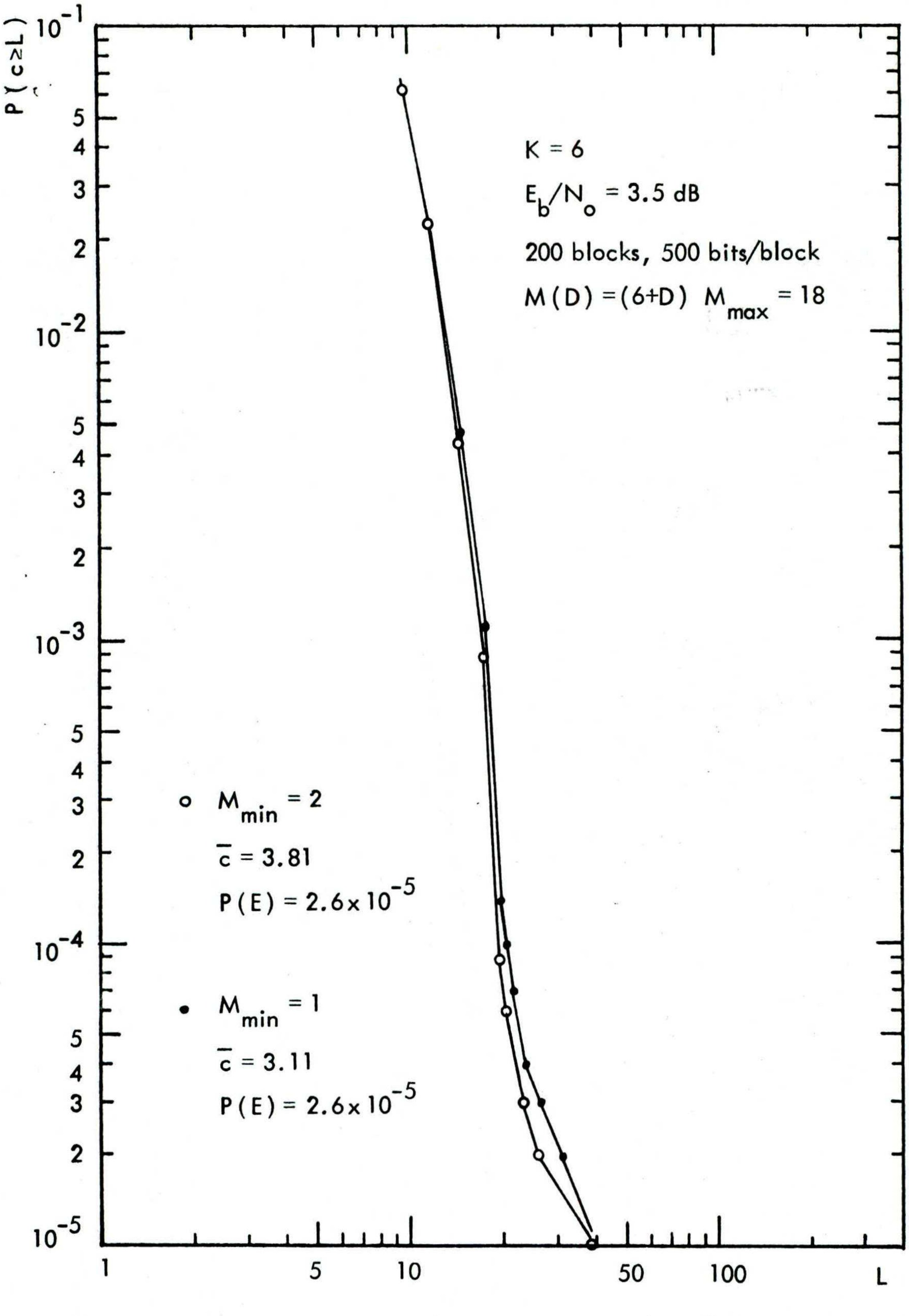


Figure (5.9) Empirical distribution of computations per search for the Adaptive algorithm. Effect of M_{min}.

triggers the extension of a larger number of paths when the constant "a" is increased. In general for any choice of $M(D)$, as soon as a metric dip is detected, the decoder should extend some minimum number of paths regardless of the size of the dip. Naturally as a consequence the average number of computations increases, but as shown in Fig. (5.6) and Fig. (5.7) the observed maximum number of computations is significantly reduced.

For $K = 6$, a Viterbi decoder requires 32 computations per decoded branch. By comparison, with $M(D) = (2+2D)$ and $M(D) = (6+D)$ we obtained a maximum of 40 computations with an average of only 2.70 and 3.11 respectively. Although these computations are more complex than those of the Viterbi decoder, such a large reduction of the average decoding effort without an undue increase of the maximum, may make the adaptive algorithm an attractive alternative for Viterbi decoding.

Fig. (5.8) shows that for a given $M(D) = 4 + D$, the distribution of computation, search mode probability, maximum number of computations and slope of the tail of the distribution all improve with increasing M_{max}. However, these improvements seem to saturate at an M_{max} value of 20 computations. Moreover, increasing M_{max} appears to have only a very small influence on the average decoding effort which increased only very slowly with M_{max}. Consequently, there is an optimal value of M_{max} at which an adaptive algorithm should operate.

Fig. (5.9) shows that extending more than 1 path when no dip is detected increases the average decoding effort with hardly any improvement on the distribution of the computation. This tends to support the assertion that most of the time, when the metric of the top node of the stack is rising, the correct path is at the top of the stack.

Results for $K = 7$ and 8 with $(E_b/N_o) = 3$ dB, and $K = 9$ with $(E_b/N_o) = 2.5$ dB are shown in Fig. (5.10) to (5.12). Again as expected the two modes of operation of the algorithm are well displayed. The limited comparison between Fig. (5.10) and Fig. (5.11) indicates that for the same $M (D)$, M_{max} and (E_b/N_o) the distribution of the computation is rather unsensitive to K for the non-search mode of operation, whereas the slope of the distribution for the search mode of operation increases with a decreasing K. This may be explained by the large increase of the number of convergences as K decreases. However, as shown in Table 5.4, for the same $M (D)$, the search mode probability and improvement over sequential decoding both decrease with (E_b/N_o).

The error probability $P (E)$ and the average number of computations $\bar{C}$ per decoded branch are indicated on each figure of the distribution curves. As expected, $\bar{C}$ increases with the value of $M (D)$, but for a given $M (D)$ and (E_b/N_o) it is rather insensitive to either M_{max} (see Fig. 5.8) or the constraint length K (see Fig. (5.10) and Fig. (5.11)). Naturally, it increases with the constant "a" of the function $M (D)$, (see Fig. (5.6)) and with decreasing

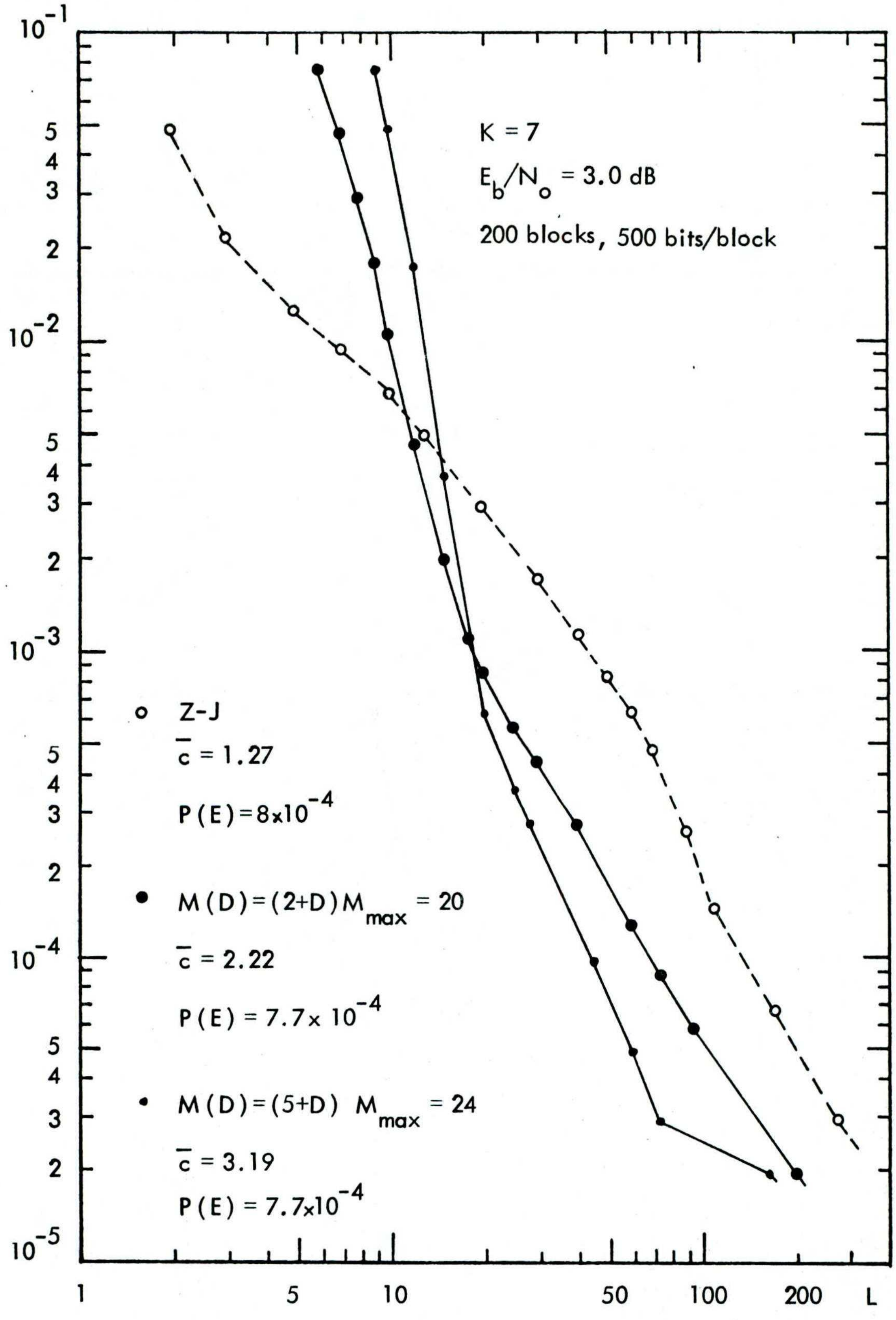


Figure (5.10) Empirical distribution of computations per search for the Adaptive Algorithm.

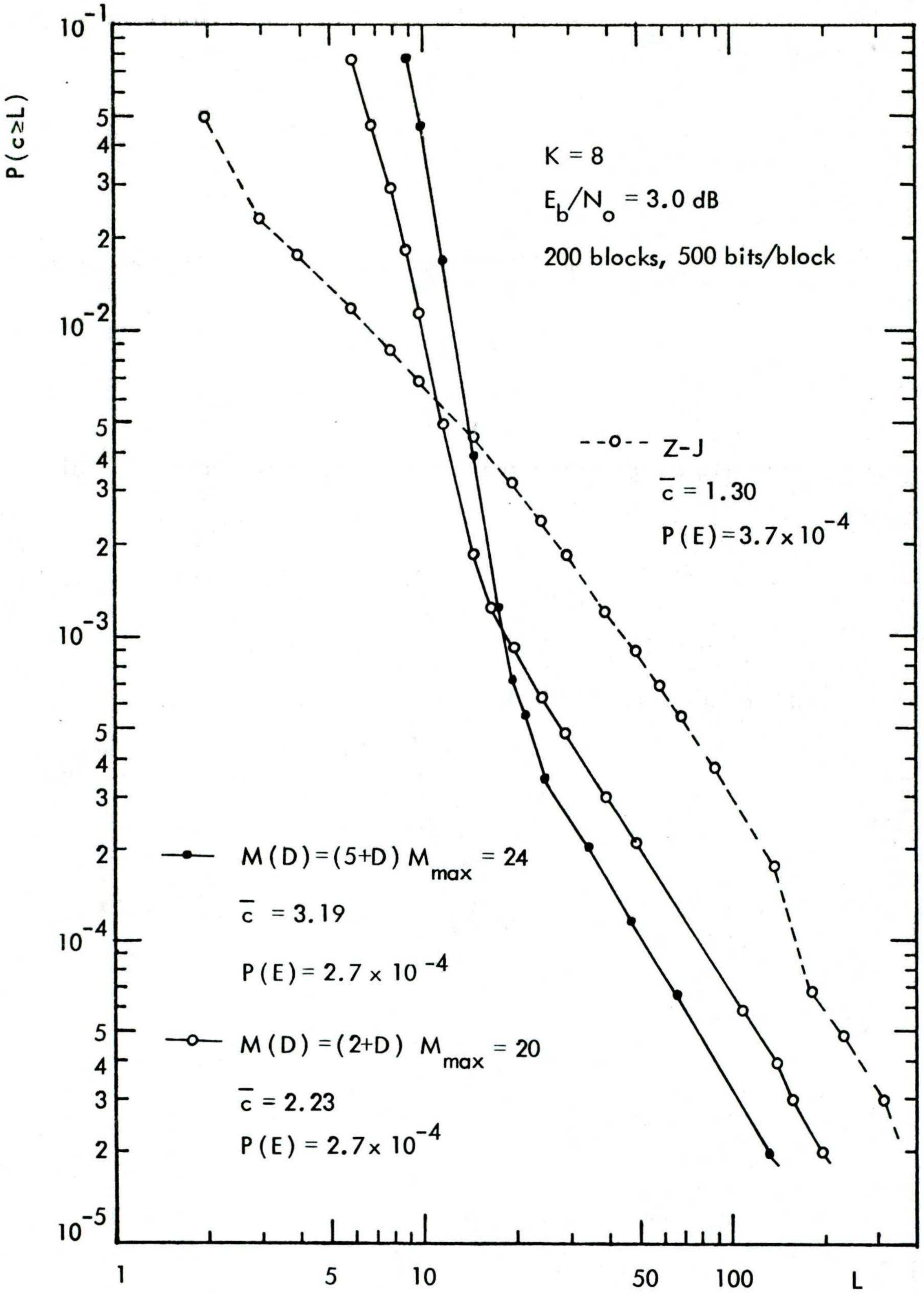


Figure (5.11) Emperical distribution of computation per search for the adaptive algorithm.

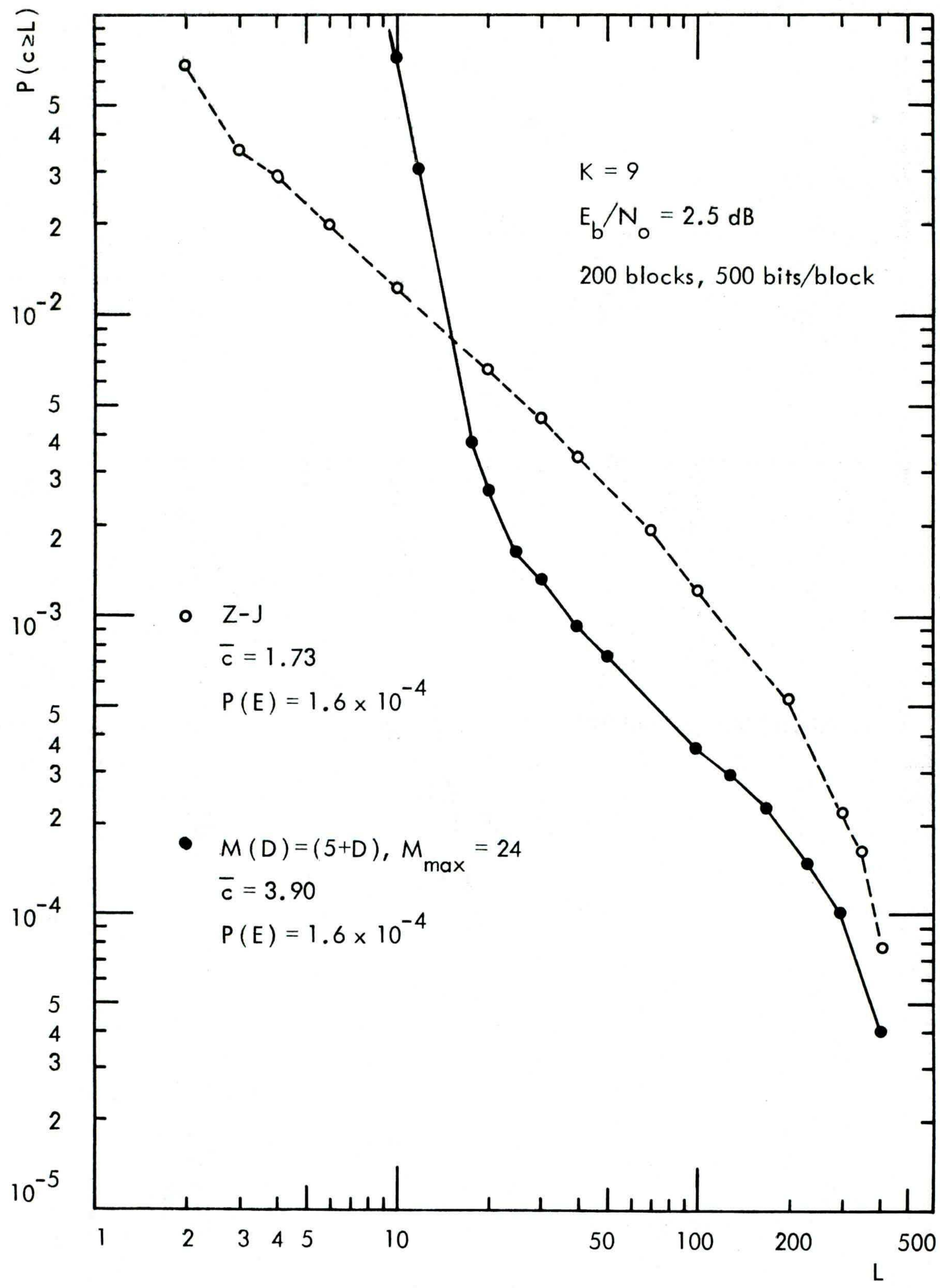


Figure (5.12) Empirical distribution of computations per search for the adaptive algorithm.

TABLE 5.4

	K	E_b/N_o (dB)	$P(C>M_{max})_{M(D)}$	$P(C>M_{max})_{Z-J}$	Improvement
$M(D) = (D+5), M_{max} = 24$	6	3.5	4.93×10^{-5}	7.71×10^{4}	16.7
	7	3.0	3.48×10^{-4}	2.25×10^{-3}	6.45
	8	3.0	3.50×10^{-4}	2.4×10^{-3}	6.80
	9	2.5	1.66×10^{-3}	5.5×10^{-3}	3.32
	6*	3.5	3.6×10^{-5}	7.71×10^{-4}	21.6
$M(D) = (D+2), M_{max} = 20$	6	3.5	1.38×10^{-4}	1.08×10^{-3}	7.85
	7	3.0	8.26×10^{-4}	2.74×10^{-3}	3.42
	8	3.0	8.74×10^{-4}	3.01×10^{-3}	3.45

* $M(D) = (D+6)$, $M_{max} = 24$

(E_b/N_o), (see Fig. (5.12)). However, in view of the improvement of the distribution over sequential decoding, the increase in $\overline{C}$ is very modest indeed.

The error probability P (E) is shown to decrease with K and (E_b/N_o), but when the constraint length is small many error events of sequential decoding are corrected by the adaptive algorithm. As expected, for the same K and (E_b/N_o), more errors are corrected as more paths are extended (see Fig. (5.6) to (5.9)). For K = 6, (E_b/N_o) = 3.5 dB, the minimum value obtained P (E) = 2.6×10^{-4}, appears to be the limit for this code and this noise sequence, whereas for K = 8 and K = 9, no improvement of the P (E) over sequential decoding was observed.

Table 5.5 gives a limited comparison of the improvement achieved by the 4-path and adaptive algorithm over sequential decoding. As expected the adaptive algorithm is clearly shown to be superior to the 4-path algorithm, and this superiority is obtained at a smaller computational cost.

All these results support our assertion that by using more information about past stack behaviour , an adaptive procedure is bound to be computationally superior to a fixed one. Of course the superiority of the adaptive algorithm increases with the amount of information used in the decision rule, the limit being the ideal algorithm where all the information for an optimum no-search operation is utilized. By basing its decision rule on the current breakout node value, the ideal algorithm not only includes the correct path in each path extension cycle, but guarantees that no future decoding cycle will require more than M_v

TABLE 5.5

	Algorithm	$\frac{P(C>M_{max})_{Z-J}}{P(C>M_{max})_{M(D)}}$	$\overline{C}$	P (E)
K = 6, E_b/N_o = 3.5 dB	Z-J		1.13	5.6×10^{-4}
	4-path	5.05	4.05	4.5×10^{-4}
	M(D) = 1+D, M_{max} = 20	5.06	1.73	4.5×10^{-4}
K = 6, E_b/N_o = 3.5 dB	Z-J		1.13	5.6×10^{-4}
	4-path	4.67	4.05	4.5×10^{-5}
	M(D) = (6+D), M_{max} = 24	21.6	3.11	2.6×10^{-4}
K = 7, E_b/N_o = 3.0 dB	Z-J		1.27	8×10^{-4}
	4-path	3.04	4.11	7.7×10^{-4}
	M(D) = 5+D, M_{max} = 24	6.45	3.19	7.7×10^{-4}

computations, regardless of the size of the correct path metric dip. Such is not the case when the decision rule attempts to only include the correct path at each path extension cycle; the maximum number of computations to decode one branch becomes unbounded. Consequently, any practical adaptive stack algorithm that uses only the information available in the stack together with a finite value for M_{max}, cannot be prevented from entering search modes of operation, and therefore an asymptotic Pareto behaviour appears to be inescapable. In our adaptive algorithm, the only direct information used in the decision rule was the detection of a dip of the top node of the stack and the determination of its size. Getting this information involved hardly any added complexity to the Z-J algorithm. All our results were obtained with a linear function of the dip sizes and no attempt was made to optimize $M(D)$. However, it may be conjectured that some other function $M(D)$, which estimates better the number of incorrect paths will further improve these results.

CHAPTER VI

CONCLUSIONS AND SUGGESTIONS FOR FURTHER RESEARCH

The results reported in this thesis show that the principal drawback of sequential decoding can be alleviated by using some of the concepts of Viterbi decoding: exploitation of the trellis structure of the convolutional code and simultaneous extension of a subset of the most likely paths. The broad class of generalized stack decoders was shown to unify the Viterbi and sequential (Z-J) decoding algorithms. We have proposed, analyzed and simulated several algorithms in this class. Compared to ordinary sequential decoding, the distribution of computations was substantially improved, and some of the undetected errors corrected, at a cost of a moderate increase of the average decoding effort. The added complexity of these algorithms over the Z-J algorithm is very modest and consists mostly in a larger memory requirement.

The M-path algorithm closes the gap between the single-path Z-J algorithm and the all-paths Viterbi algorithm. Depending on the value of M relative to the number M_v of distinct states, the M-path algorithm is "closer" to either of these two algorithms, but unless $M = M_v$, the computational behaviour remains asymptotically that of a sequential decoder. As the decision rule is fixed, the average number of computations per decoded node varied linearly with M. However because of the purging cycle the stack storage is reduced and all computations are not equally complex. Compared to the Z-J algorithm, we observe that (1) A new entry in the stack involves a slightly more complex operation since it requires the setting of the LINK pointer. (2) The elimination of a converging branch represents a far less complex operation. (3) A stored converging branch may correspond to a more or less complex operation depending on whether or not the original substack value is modified. Hence, as the number of convergences increases, the computational complexity of the M-path algorithm will effectively increase slower than linearly with M.

The subclass of adaptive algorithms departs from either the Z-J, M-path or Viterbi decoding algorithms. It was shown that by modifying the path extension decision rule according to the information contained in the stack, some useless computations can be eliminated. The computational variability could thus be further reduced without unduly increasing the average decoding effort. As more information about the correct path is included in the decision rule, the overall performance improves, up to a limit set by the ideal algorithm. Of course, in evaluating the real performance of the adaptive algorithm, the cost of getting this information must be properly accounted for.

The algorithms presented here could be used with any constraint length convolutional code. They may be particularly attractive in those applications where the reduction of the computational variability, hence the reduction of the size of the input data buffer, is well worth the modest increase of the average decoding effort. In some sense, these algorithms extend the range of application of sequential decoding to relatively short constraint length codes, and may be an attractive alternative to Viterbi decoding for short to moderate constraint length codes $6 \leq K \leq 15$.

Among the class of generalized stack decoders, the adaptive algorithms appears to be the most promising. An interesting area for further research is finding more efficient methods to adapt the decoding effort to the top node metric dips. For instance the metric behaviour of the top node of the stack could be determined a few branches ahead, and this "prediction" could be used in the decision rule for the running decoding cycle. Efficient functions $M(D)$ for the adaptive algorithm may be determined by a combination of a theoretical analysis of the population of the incorrect paths and computer simulation.

Most of the increased decoder complexity comes from the exploitation of the trellis structure of the code in the purging cycle. This particular purging rule may thus be dropped when the number of remergers does not not warrant the required additional memory. (In our simulation the effect of the remergers was negligible for $K \geq 8$). Other purging rules and their consequences on the error probability may thus be investigated. For example, eliminating from the stack all nodes whose metric is below some threshold value below the current maximum metric value.

Some other interesting problems for future research are:

(a) Metric bias. We have used the Fano metric with a bias value equal to the rate of the code since it minimizes the extension of unlikely paths. However with an unbiased metric, no path can converge with a larger metric on a node stored in the stack. Hence, on the trellis structure of the code no computations can be repeated but the number of explored incorrect paths is expected to increase. Therefore, with a purging rule exploiting the reconvergence of the paths, the determination of the optimum bias value that would minimize the average decoding effort of the particular M-path and adaptive algorithms would be interesting.

(b) Waiting line behaviour. The effect of the reduction of the computational variability on the buffer storage reduction and buffer overflow probability was not examined and warrants future research.

(c) Dynamic behaviour of the incorrect paths. The search mode probabilities P_H were determined under the assumption the decoder can ignore all dips whose value is smaller than H. The problems of relating M or $M(D)$ to H, and the bounding of $\overline{C}$ for the adaptive algorithm lie with the dynamic behaviour of the population of the incorrect paths. In analyzing this behaviour the branching process model of Appendix III appears to be most promising.

APPENDIX I

THE GENERALIZED STACK DECODER

In this appendix we briefly describe the operation of the generalized stack algorithm as we have programmed it in Fortran. It follows in essence the Z-J algorithm described by Jelinek (1969a) and Geist (1970).

As mentioned in Section 4.2.2, an entry in the stack consists of the 3 items (metric value, node depth, encoder state) necessary for a possible further extension of the node, and of 3 pointers necessary for the stack ordering, path retrieval and convergence test. This information is stored in six contiguous words of memory labelled respectively VALUE, DEPTH, STATE, STAKPT, PATHP and LINK. In addition 2 auxilliary arrays called AUXPT and NODEPT are necessary to scan the stack as explained below.

Initially all arrays are cleared, but the origin node (the first top node) is assigned an arbitrary positive metric value to avoid encountering negative total metric values during decoding. The information about new nodes is entered in the stack in consecutive stack addresses, and once entered, this information is never destroyed. All new entries whose accumulated metric, say Γ, yields the same integer part Q of Γ/H (see Eq. 3.16), belong to the same substack Q. Moreover the algorithm always keeps track of the highest substack Q_{Top} ever reached.

In a given substack say Q_m, all the nodes are linked in a continuous fashion by their pointers STAKPT. The STAKPT of the first node in Q_m contains 0, the STAKPT of the second entry contains the address of the first entry, and so on. The address of the last new entry in Q_m (i.e. the beginning of the chain) is contained in the Q_m th word of the auxilliary array AUXPT. Suppose a new extension yielding the metric value Γ and substack Q_m is about to be stored in the stack at address N. The metric value, state, and depth are first properly stored in their respective registers in the stack at address N. Then the address contained in AUXPT (Q_m) is entered in STAKPT (N) and N is entered in AUXPT (Q_m). For any empty substack Q_i, the corresponding contents of AUXPT (Q_i) is zero.

When a node is extended, its "elimination" from the stack consists simply in bypassing it in the chain of STAKPT pointers. Since the highest encountered substack Q_{Top} is always known, to determine the top node of the stack, starting from Q_{Top}, the first non-empty substack is found and then scanned for the maximum metric value. The chain of STAKPT pointers and the AUXPT array are then properly modified to exclude this top node. In the quantized Jelinek algorithm no scanning is performed within the substack and the top node is considered to be the last entry in the highest non-empty substack.

If the path extension cycle requires the extension of a number, say M nodes, these M nodes are extracted in sequence from the top node to the Mth, using the same technique. For the adaptive stack algorithm, the dip value D must be first determined. Using the current value of Q_{Top}, after the top node is extracted the dip value D and the number M (D) are readily obtained.

However only after all these nodes have been "deleted" from the stack will the extensions be performed.

When the top node reaches the end of the tree, the information symbols on the finally decoded path must be recovered. This is accomplished by a chain of PATHP pointers. For every new entry in the stack, the stack address of the parent node is stored in the PATHP register. Hence the chain of PATHP pointers specify the decoded path from the final node back to the origin.

The purging cycle exploits the reconvergence of the explored paths. As described in Section 4.2, all nodes in the stack having the same depth are linked together by the chain of LINK pointers. The beginning of the chain of depth, say N, (i.e. the last unmerged entry at depth N) is contained in the Nth word of the array NODEPT. For the first entry at a given depth, the LINK contains 0, and hence the test for convergence is easily performed by following the LINK pointers, and comparing the state of the new extension with the states of the nodes specified by the LINK pointers. If no convergence is detected the new extension is properly entered in the stack. If there is a convergence the metrics are then compared. Suppose the new extension has the larger metric of the pair. Then the metric value and predecessor address of the new extension replace the corresponding contents of the VALUE and PATHP registers of the older node. Moreover to keep the stack properly ordered, if the new substack value is increased, the chain of STAKPT and AUXPT pointers must be modified accordingly. Finally, if the new extension does not increase the metric value of the older node, the new extension is simply discarded. Therefore the redundancy in the stack is always eliminated and no storage is wasted.

APPENDIX II

MARKOV CHAIN MODEL OF THE CORRECT PATH METRIC

In this appendix we present the Markov chain model of the correct path metric, and some of its properties related to sequential decoding.

Consider some node $\underline{U}_K$ on the correct or decoded path and let its metric be Γ_K. Then a fundamental quantity closely related to its decoding is the metric difference Δ_K given by

$$\Delta_K = \Gamma_K - \min_{i \geq k} \Gamma_i \tag{1}$$

with

$$\Gamma_K = \sum_{i=0}^{K} \gamma_i \tag{2}$$

where $\{\gamma_i\}$ are the correct branch metric values. From Eq. (1) we have

$$\Delta_K \geq 0 \quad K = 1, 2, \ldots \tag{3}$$

where the equality holds only for breakout nodes.

Breakout nodes on the correct path are decoded by a single computation, but as Δ_K increases, the number of computations necessary for decoding $\underline{U}_K$ increases exponentially. The distribution of Δ_K is thus an important factor in determining the computational behaviour of sequential decoding.

A recursive relation between consecutive values of Δ_K may be obtained as follows. From Eq. (1) and (2) and using $\Gamma_{K+1} = \Gamma_K + \gamma_{K+1}$, we have

$$\Delta_K = - \text{Min} \, [0, \gamma_{K+1}, (\gamma_{K+1} + \gamma_{K+2}), (\gamma_{K+1} + \gamma_{K+2} + \gamma_{K+3}), \ldots] \quad (4)$$

Hence,

$$\Delta_K = \begin{cases} - \text{Min} \, [\gamma_{K+1}, (\gamma_{K+1} + \gamma_{K+2}) \ldots] & \text{if } - \text{Min} \, [\gamma_{K+1}, (\gamma_{K+1} + \gamma_{K+2}) \ldots] > 0 \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Likewise

$$\Delta_{K+1} = - \text{Min} \, [0, \gamma_{K+2}, (\gamma_{K+2} + \gamma_{K+3}), \ldots] \quad (6)$$

and

$$\Delta_{K+1} - \gamma_{K+1} = - \text{Min} \, [\gamma_{K+1}, (\gamma_{K+1} + \gamma_{K+2}), \ldots] \quad (7)$$

Substituting Eq. (7), in Eq. (8), we obtain

$$\Delta_K = \begin{cases} (\Delta_{K+1} - \gamma_{K+1}) & \text{if } (\Delta_{K+1} - \gamma_{K+1}) \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

Equivalently we have

$$\Delta_K = \text{Max} \, [0, (\Delta_{K+1} - \gamma_{K+1})] \quad (9)$$

Since the branch metrics γ's are statistically independent random variables with a common distribution, the sequence $\Delta_{K+2}, \Delta_{K+1}, \Delta_K, \ldots,$ defined by Eq. (8) induces a queuing process. This queuing process is equivalent to a Markov chain where the states are the possible values of Δ_K, and the 1-step transition probabilities are the probabilities associated with the branch metrics γ_K's, that is

$$P_\ell = P(\gamma_K = -\ell) \quad , \quad 0 \le \ell \le J$$
$$q_\ell = P(\gamma_K = +\ell) \quad , \quad 0 < \ell \le Q \tag{10}$$

This set of probabilities is easily obtained from the channel transition probabilities and the definition of the biased metric. The states Δ_K are integers since the branch metric values are always rounded to integers in practical sequential decoders.

If a node $\underline{U}_K$ has some value $\Delta_K = j$, then state j of the chain is occupied. The next state to be occupied, say state ℓ, will be determined by the 1 step transition from state j, according to the set of probabilities of Eq. (10), and $\Delta_{K-1} = \ell$. For example, in Fig. (II.1), the set of the possible branch metrics is $\{+2, +1, 0, -3, -7\}$ and the corresponding probabilities are $\{q_2, q_1, p_o, p_3, p_7\}$. As shown in Fig. (II.1), for this particular example, all states to the right of state 2 exhibit the same transition pattern.

A breakout node of the correct path locates the decoder at state 0 whereas a non breakout node will locate it at the corresponding non-zero state. Clearly decoding the correct path is executing an integer-valued random walk where a visit to any non-zero state corresponds to a search-mode of operation. Since from state zero the decoder could either remain in that state or go into a search, state zero acts as a reflecting barrier at the origin for the random walk. There are Q possible transitions towards the zero-state and J possible transitions away from the zero state, corresponding respectively to the possible Q positive and J negative branch metrics values. The form of the probability transition matrix for such a random walk with $Q = 3$ and $J = 5$ is given below

$$\begin{array}{c|cccccccccc}
\nearrow & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \cdot & \cdot \\
\hline
0 & p_o + \sum_{i=1}^{3} q_i & p_1 & p_2 & p_3 & p_4 & p_5 & 0 & 0 & 0 & 0 \\
1 & \sum_{i=1}^{3} q_i & p_o & p_1 & p_2 & p_3 & p_4 & p_5 & 0 & 0 & 0 \\
2 & \sum_{i=2}^{3} q_i & q_1 & p_o & p_1 & p_2 & p_3 & p_4 & p_5 & 0 & 0 \\
3 & q_3 & q_2 & q_1 & p_o & p_1 & p_2 & p_3 & p_4 & p_5 & 0 \\
4 & 0 & q_3 & q_2 & q_1 & p_o & p_1 & p_2 & p_3 & p_4 & p_5 \\
5 & 0 & 0 & q_3 & q_2 & q_1 & p_o & p_1 & p_2 & p_3 & p_4 \\
6 & \cdot & \cdot & 0 & q_3 & q_2 & q_1 & p_o & p_1 & p_2 & p_3 \\
\cdot & \cdot & \cdot & \cdot & 0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & & & & & & & & & &
\end{array}$$

This Markov chain model for Δ_K was first proposed by Massey and al (1969, 1972) who considered the particular case in which only a single transition of unit length is permitted toward the origin state (i.e., $Q = 1$, $P(\gamma_K = +1) = q$), but any finite number J of transitions away from the origin are permitted.

Regardless of the chain, when a node $\underline{U}_K$ on the correct path has a positive metric difference, $\Delta_K > 0$, there is a search associated with this node, and clearly the correct path metric must go through a dip at least equal to Δ_K before reaching the next breakout node. We can therefore associate the Δ_K's with the correct path metric dips. Of particular importance with this representation is the stationary probability distribution of the different states or dip values, and the average distance between breakout nodes.

Properties of the Markov Chain

We now interpret several well known properties of the Markov chain (see for example Feller 1966) with respect to sequential decoding.

Property 1: The average distance between breakout nodes of the correct path is equal to the mean recurrent time of the origin state.

Suppose state j can be reached from state i in n steps. Let T_{ij} be the waiting time (in number of transitions required) for the first entrance to state j from the initial state i. Then

$$f_{ij}^{(n)} = P (T_{ij} = n) \tag{11}$$

is the probability to reach state j for the first time from state i in exactly n steps. The probability of eventually reaching state j in a finite number of steps from state i is

$$f_{ij} = P (T_{ij} < \infty) = \sum_{n=1}^{\infty} f_{ij}^{(n)} \quad , \quad \forall \; i,j \tag{12}$$

In particular T_{ii} is the return time to state i or recurrence time of i, and if $f_{ii} = 1$, then state i is said to be recurrent or persistent. If $f_{ii} < 1$, state i is said to be transient.

For a recurrent state i, the mean recurrence time (or mean return time) is defined as

$$\mu_i = \sum_{n=1}^{\infty} n \, f_{ii}^{(n)} \tag{13}$$

If $\mu_i = \infty$, state i is called a null state, and if $\mu_i < \infty$ it is called a nonnull state.

For sequential decoding the probability that the first return to state 0 (a breakout node on the correct path) takes n steps is

$$P (T_{oo} = n) = f_{oo}^{(n)} \tag{14}$$

Hence the average number of steps to return to state 0, or equivalently the average distance d_o (in branches) between consecutive breakout nodes of the correct path is the mean recurrence time of state 0,

$$d_o = \mu_o = \sum_{n=1}^{\infty} n \, f_{oo}^{(n)} \tag{15}$$

The average distance d_o is finite only if state 0 is recurrent nonnull. We know that for sequential decoding, the only incorrect paths of interest emerge from non breakout node on the correct path. Therefore in general both the average value and the variability of the computational effort increase with the average distance between consecutive breakout nodes. A decoding algorithm which effectively reduces the average distance d_o will also reduce the variability of the decoding effort, and if $d_o = 1$ this algorithm will never be in a search mode of operation.

<u>Property 2:</u> If $R < R_{comp}$ for the sequential decoder, then state 0 is recurrent.

If $R < R_{comp}$, then the average number of computations to decode one branch on the correct path is finite. Therefore the decoder always return to a breakout node or zero state of the chain. Hence $f_{oo} = 1$ and state 0 is recurrent.

<u>Property 3:</u> The Markov chain is irreducible.

Ther is no absorbing state in our Markov Chain, every state can be reached from every other state and therefore the chain is irreducible. Since in any irreducible Markov chain all states are of the same class (recurrent or transient), if state 0 is recurrent, then all other states are recurrent and the chain is said to be ergodic.

Stationary Distribution of the Chain

A probability distribution $\{v_j\}$ is called stationary if

$$v_j = \sum_{i=0}^{\infty} v_i P_{ij} \quad , \quad j = 0, 1, 2, \ldots \tag{16}$$

If a chain has a stationary probability distribution, then the unconditional distribution of occupancy of the states becomes independent of time: the process is in statistical equilibrium. When it exists, the stationary distribution is unique and the v_j are given by

$$v_j = \frac{1}{\mu_j} \tag{17}$$

where μ_j is the mean recurrence time of state j.

If such a distribution exists for the states of our chain, then

$$P(\Delta_K = \ell) = v_\ell \, , \quad \ell \geq 0 \tag{18}$$

Of special interest is the stationary probability v_o of occupying state 0, since it represents the relative frequency of breakout nodes on the correct path which a sequential decoder will decode in a single computation. Furthermore from

Eq. (15) and (17) the average distance between breakout nodes would be given by

$$d_o = \frac{1}{v_o} \tag{19}$$

It is well known that an irreducible Markov chain has a stationary distribution if and only if all states are recurrent nonnull. Hence from Property 2 and 3, if $R < R_{comp}$ the stationary distribution for our chain exists. Let

$$z = \sum_{i=1}^{Q} i\, q_i - \sum_{\ell=0}^{J} \ell\, p_\ell \tag{20}$$

be the average length or "drift" of a single step. Then Feller (1966, Vol. 1) shows that the chain has a unique stationary distribution if and only if $z > 0$.

The determination of the $\{v_i\}$ according to Eq. (16) is in general quite cumbersome. For the simple case where there are any finite number J of transitions away from the origin state, but where only a single transition of unit length towards the origin is permitted, Massey et al (1972) give a simple recursive technique to determine the stationary probabilities $\{v_i\}$, given they exist. For such a chain

$$z = q - \sum_{\ell=0}^{J} \ell\, p_\ell \tag{21}$$

and assuming $z > 0$, v_o is given by

$$v_o = \frac{z}{q} \tag{22}$$

Given v_o and using Eq. (16) the stationary probabilities for all the other states are determined recursively.

$$v_o = v_o(q + p_o) + v_1 q$$
$$v_1 = v_o p_1 + v_1 p_o + v_2 a$$
$$\vdots$$
$$v_J = v_o p_J + v_1 p_{J-1} + \dots + v_J p_o + v_{J+1} q \tag{23}$$

and

$$v_{n+1} = \frac{1}{q} [v_n - \sum_{i=0}^{J} p_i v_{n-i}], \quad n > J \tag{24}$$

For this chain the average distance between breakout nodes is then

$$d_o = \frac{1}{v_o} = \frac{q}{z} \tag{25}$$

Unfortunately for the general case where there are Q, $Q > 1$ positive branch metric values, this "drift balancing" technique cannot be used (Massey 1972). However it can be applied for the binary symmetric channel when all metric values are normalized so that the positive branch metric has unit value.

The stationary probability v_o represents the proportion of the nodes on the correct path that will be decoded by a single computation. Hence a proportion $(1-v_o)$ of the nodes on the correct path are non breakout nodes and are thus responsible for the varying of the computational effort of sequential decoding. Eq. (19) is intuitively agreeable since as v_o increases, d_o and the proportion of non breakout nodes both decrease, decreasing also the variability of the decoding effort. Clearly a procedure which reduces the effective d_o will consequently reduce the variability of the computation.

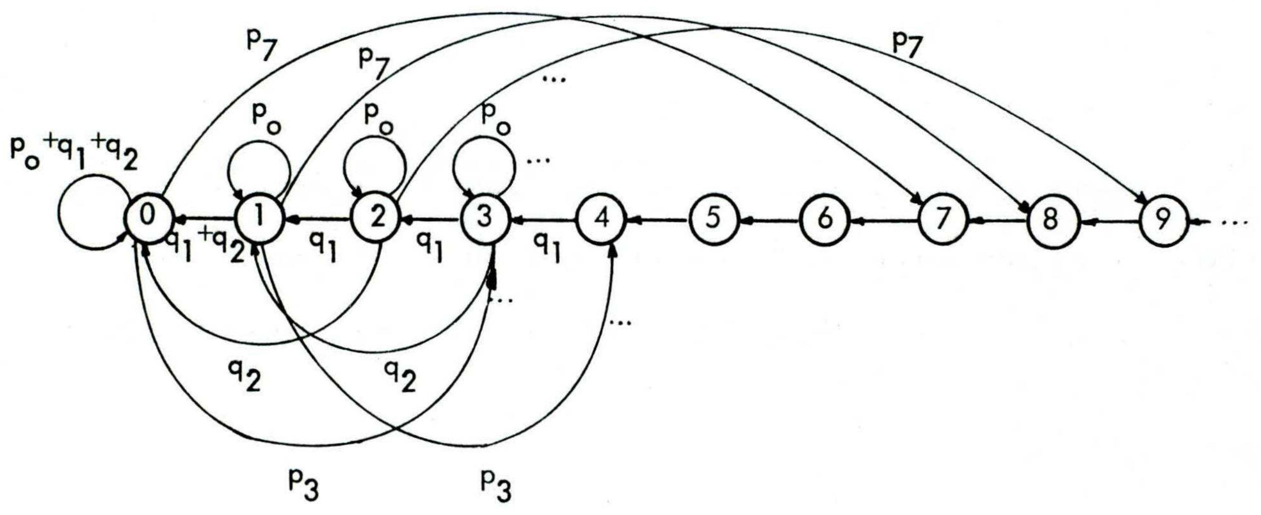


Figure (II.1) Example of a Markov chain with $Q = 2$, $J = 7$.

APPENDIX III

BRANCHING PROCESS MODEL OF THE INCORRECT PATHS

In an ordinary discrete time branching process (see for example Feller (1966), Vol. 1), a single particle (called the zeroth generation) produces 0, 1, 2, ..., other particles with probability $p_o, p_1, p_2, \ldots$. These particles form the 1st generation and produce further particles independently, and distributed as $\{p_m\}$, $m = 0, 1, 2, \ldots$ In general each of the particles of the nth generation can produce m descendants with probability p_m, $m = 0, 1, 2, \ldots$. By considering the particles as nodes and generations as tree depths, we see that a branching process expands in a tree-like structure.

Consider now a special branching process where each particle gives birth to the same number, say N, of descendants for the next generation. A parent particle being at some amplitude level ℓ, $\ell = 1, 2, \ldots H$ above a ground level of amplitude zero, each of the N descendants moves away from the parent and goes independently to an amplitude level $(\ell + j)$ with a fixed probability assignment $\{p_j\}$. If a particle moves to either an amplitude level greater than H or smaller or equal to zero, it disappears. That is, the amplitude levels 0 and $(H+1)$ act as absorbing barriers for the process. Such a process is called a branching random walk with absorbing barriers (BRWAB). (See Gallager 1974).

In the Markov chain representation of the metric differences $\{\Delta_k\}$ of the correct path, breakout nodes $(\Delta_k = 0)$ may be associated with a zero or ground level. Non-breakout nodes would then correspond to nodes whose amplitude levels are the corresponding Δ_K above the ground level. Each of these non-breakout nodes is a root node of a subtree of incorrect paths that must be extended by a sequential decoder as long as they lie above the ground level. Hence the ground level acts as an absorbing barrier for the incorrect paths.

Since the number of descendants of each node is constant, and assuming the branch metrics to be statistically independent, we observe a close connection between BRWAB and the subtrees of incorrect paths that must be extended by a sequential decoder if we add another absorbing barrier of amplitude level (H+1) above the ground level. Then, any incorrect path that crosses (or touches) either barrier will be absorbed. Although absorbing the incorrect paths that cross the upper barrier is not very realistic, these crossings will be very rare occurences if H is high enough. The main reason of its introduction is analytical convenience (Gallager, 1974). With this model of the incorrect paths emerging from non-breakout nodes on the correct path, each incorrect node is a particle and its metric above the ground level is the particle amplitude level. If a particle moves from amplitude level i to j, the metric increment for the corresponding incorrect branch is $(j-i)$, and this increment occurs with probability P_{j-i}. These probability assignments are easily obtained from the channel transition probabilities.

As in any branching process, our BRWAB may grow without limit or become extinct. Since it represents incorrect paths that must be explored by a sequential decoder, we want our process to be eventually extinguished with certainty. Gallager (1974) has shown that the probability of extinction is 1 if

$$N \phi(t) < 1 \tag{1}$$

Where N is the number of descendants per particle and where t minimizes the moment generating function:

$$\phi(s) = \sum_i P_i e^{si} \tag{2}$$

If the process were to grow without limit, then the exploration of an incorrect subtree may take an infinite amount of computation, leading to an unbounded decoding effort with probability 1. Hence inequality (1) may be taken as an alternative to R_{comp} of sequential decoding.

We now find a bound on the expected number of computations performed by a Z-J algorithm to decode one branch on the correct path. We recall that a computation is defined as the extension of a node into its N immediate descendants. The subtree of incorrect paths emerging from a non-breakout node being modelled as a BRWAB, we assume the existence of an upper absorbing barrier. By using the branching process model of the incorrect paths and the Markov Chain model of the correct path, the following analysis departs from the traditional union or Chernoff bounding techniques. This analysis was inspired by a technique used by Gallager (1974) to determine the average number of nodes explored by a tree encoding algorithm of source sequences.

Let $c_i(L)$ be the expected number of computations performed by the algorithm in extending up to a distance L branches away, all incorrect paths issued from a root node of amplitude level i, $1 \le i \le H$. For $L = 1$ only the root node is counted and $c_i(1) = 1$.

For $L \ge 2$, as shown in Fig. (III.1) one of the branches of the first generation has a probability P_{j-i} to be at amplitude level j. If this amplitude level j is between the two barriers, the algorithm will extend it further. Hence this node can be considered as the root node of amplitude j, $1 \le j \le H$, of a new subtree of incorrect paths whose termination length is $(L-1)$. Given the amplitude level of the root node is j, the conditional average number of computations is $c_j(L-1)$, and the unconditional average is then $\sum_{j=1}^{H} P_{j-i}\, c_j(L-1)$. The same argument applies to the remaining $(N-1)$ other nodes. Assuming statistical independence between the incorrect paths we obtain the recursive relation

$$c_i(L) = 1 + N \sum_{j=1}^{H} P_{j-i}\, c_j(L-1), \quad L \ge 2 \qquad (3)$$

The above expression assumes all nodes to give birth to the same number of descendants. However, the first generation of nodes consists of one node on the correct path and $(N-1)$ incorrect branches, whereas beyond the 1st generation all descendants of incorrect nodes are incorrect nodes (see Fig. III.2).

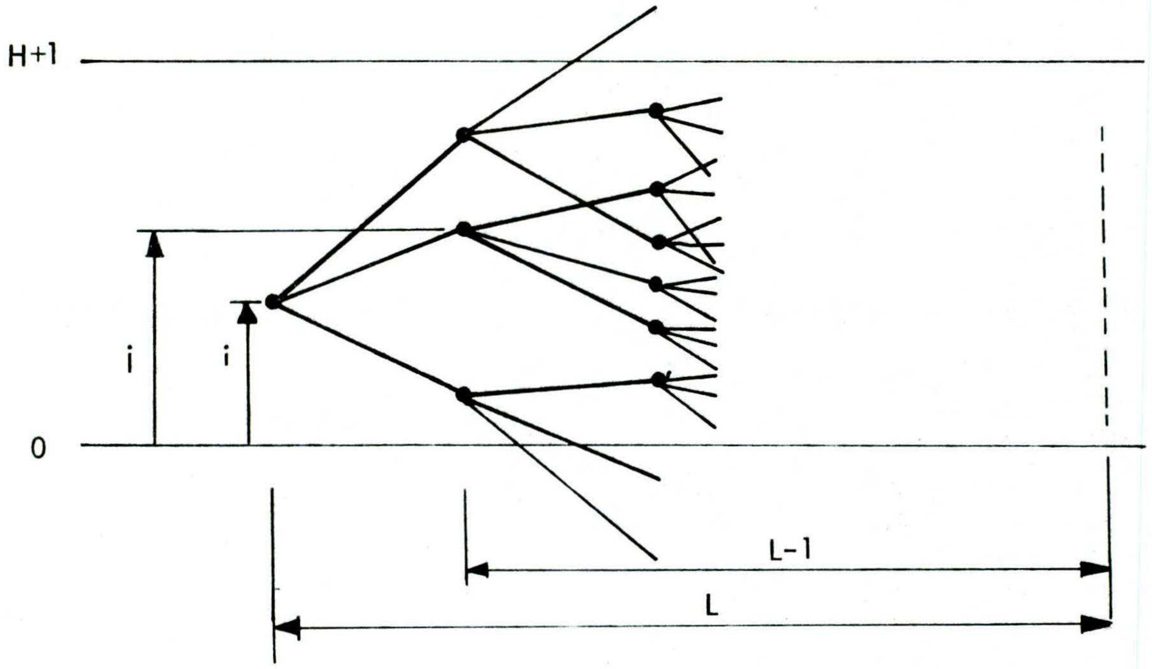


Figure (III.1) BRWAB with N = 3 branches per node.

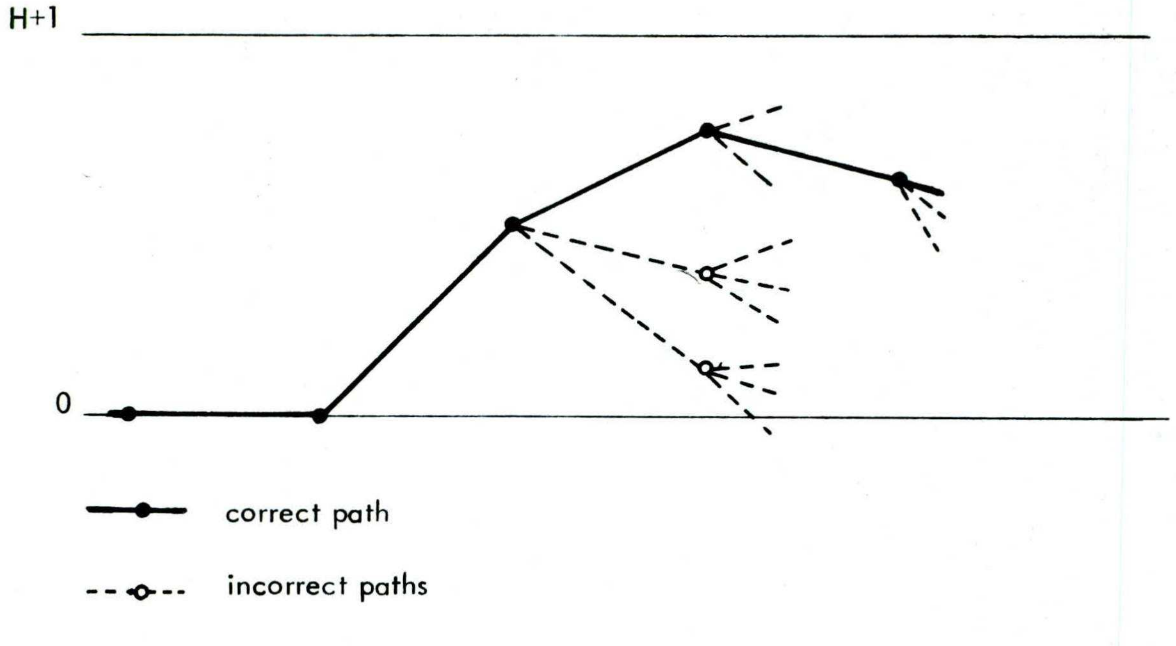


Figure (III.2) BRWAB model of the incorrect paths.

Consequently the average amount of computations performed on the incorrect paths diverging from a non-breakout node of amplitude level i is bounded by the right side of Eq. (3). Hence

$$c_i(L) \leq 1 + N \sum_{j=1}^{H} P_{j-i}\, c_j(L-1) \quad , \quad (L \geq 2)$$

$$c_i(1) = 1 \tag{4}$$

We now proceed to remove the conditioning on the amplitude level i. Define the H-dimensional vectors

$$\underline{c}(L) = \begin{bmatrix} c_1(L) \\ c_2(L) \\ \vdots \\ c_H(L) \end{bmatrix} \tag{5}$$

and

$$\underline{1} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{6}$$

Let the $H \times H$ matrix $[B]$ be defined as

$$[B] = \begin{bmatrix} P_{11} & P_{12} & P_{13} & & P_{1H} \\ P_{21} & P_{22} & P_{23} & & P_{2H} \\ \vdots & \vdots & \vdots & & \vdots \\ P_{H1} & P_{H2} & P_{H3} & \cdots & P_{HH} \end{bmatrix} \tag{7}$$

where $\{ P_{ij} \}$ are the incremental probabilities $\{ P_{j-i} \}$. The vector form of (4) is then

$$\underline{c}(L) \leq \underline{1} + N [B] \underline{c}(L-1), \quad L \leq 2$$
$$\underline{c}(1) = 1 \tag{8}$$

Recursively we then have for $L \geq 2$

$$\underline{c}(L) \leq \underline{1} + N [B] \underline{1} + N^2 [B]^2 \underline{1} + \ldots + N^{L-1} [B]^{L-1} \underline{1} \tag{9}$$

Defining

$$[B_N] = N [B] \tag{10}$$

we obtain

$$\underline{c}(L) \leq (I + [B_N] + [B_N]^2 + \ldots + [B_N]^{L-1}) \underline{1} \tag{11}$$

Assuming the process will be extinguished for L large enough, $\underline{c}(L)$ does not diverge. If the largest eigenvalue of $[B_N]$ is less than 1, than as $L \to \infty$ the right hand side of (11) converges, to give

$$\underline{c}(L) \leq \underline{c}(\infty) \leq [I - [B_N]]^{-1} \underline{1} \tag{12}$$

Using the stationary probability distribution $\{v_i\}$ of the correct path metric differences, the unconditioned average decoding effort $\bar{c}$ on the incorrect paths is

$$\bar{c}' \leq \underline{V}_H^T \cdot \underline{c}(\infty) = \sum_{i=1}^{H} v_i c_i(\infty) \qquad (13)$$

where

$$\underline{V}_H^T = [v_1, v_2, \ldots v_H] \qquad (14)$$

Finally, counting the single computation performed on the breakout node (whose stationary distribution is v_o), the average number of computations $\bar{c}$ to decode one branch on the correct path is upper bounded by

$$\bar{c} \leq v_o + \underline{V}_H^T \cdot \underline{c}(\infty) \qquad (15)$$

For noiseless channels, all nodes on the correct path are breakout, $v_o = 1$ and $\underline{V}_H = \underline{0}$. The computational effort is constant, $\bar{c} = v_o = 1$, and the average distance between the nodes is $d_o = 1$. As the channel becomes noisier, v_o decreases and $\underline{V}_H > \underline{0}$. If $\bar{c}'$ is bounded, then $\bar{c}$ is also bounded by a larger value than 1. Moreover d_o increases, increasing with it the variability of the computational effort. Naturally if $\bar{c}'$ diverges, then obviously $\bar{c}$ also becomes unbounded.

The Average Population of Incorrect Paths

The above technique may also be used to determine the average number of incorrect nodes existing between the barriers at any depth away from the parent node.

Suppose a parent particle has an amplitude level i, $1 \leq i \leq H$, and let $\mu_i(L)$ be the average number of particles existing between the two absorbing barriers, from the zeroth to the Lth generation. Assuming statistical independence between the N immediate descendants of any particle, as in Eq. (3) we have

$$\mu_i(L) = 1 + N \sum_{j=1}^{H} P_{j-i}\, \mu_j(L-1), \qquad L \geq 2$$

$$\mu_i(1) \leq 1 + N \sum_{j=1}^{H} P_{j-i} \tag{16}$$

Here again, for sequential decoding the root node is on the correct path, hence for the first generation there are at most (N-1) incorrect nodes. Therefore the right hand side of Eq. (16) is an upper bound to $\mu_i(L)$

$$\mu_i(L) \leq 1 + N \sum_{j=1}^{H} P_{j-i} \; (L-1), \quad L \geq 2)$$

$$\mu_i(1) \leq 1 + N \sum_{j=1}^{H} P_{j-i} \tag{17}$$

Define the H-dimensional vector

$$\underline{\mu}(L) = \begin{bmatrix} \mu_1(L) \\ \mu_2(L) \\ \vdots \\ \mu_H(L) \end{bmatrix} \tag{18}$$

The vector form of Eq. (16) is then

$$\underline{\mu}(L) \leq \underline{1} + N[B]\,\underline{\mu}(L-1)\quad , \quad L \geq 2$$

$$\underline{\mu}(1) \leq \underline{1} + N[B]\,\underline{1} \tag{19}$$

where $\underline{1}$ and $[B]$ are given by Eq. (6) and (7) respectively. Using $[B_N]=N[B]$ we then obtain

$$\underline{\mu}(L) \leq (I + [B_N] + [B_N]^2 + \ldots + [B_N]^L)\,\underline{1} \tag{20}$$

Using the stationary probabilities $\{v_i\}$ for the nodes on the correct path, the unconditioned average number of incorrect nodes is bounded by

$$\bar{\mu}(L) \leq \underline{V}_H^T \cdot \underline{\mu}(L) = \sum_{i=1}^{H} v_i\,\mu_i(L) \tag{21}$$

where $\underline{V}_H^T$ is given by Eq. (14).

Consequently the average number of incorrect nodes $\overline{M}(D)$ lying between the barriers at a distance D, $D \geq 1$, branches away from the root node is then

$$\overline{M}(D) \leq \bar{\mu}(D) - \bar{\mu}(D-1) \tag{22}$$

$$\leq \underline{V}_H^T \cdot ([B_N]^D\,\underline{1}) \tag{23}$$

where Eq. (20) was used to obtain Eq. (23). The largest value of the average number of incorrect nodes lying at the same tree depth may thus be found by scanning $\overline{M}(D)$ for maximum over D.

APPENDIX IV

SEARCH MODE PROBABILITY

In this appendix we present a direct method of obtaining P_H, the probability of entering a search mode of operation when starting from a breakout node on the correct path. Since all states at or beyond state H correspond to the search mode of operation, they can all be lumped together into an absorbing state at H. The transition probability matrix of the chain becomes

$$\begin{array}{c|cccccc|}
 & 0 & 1 & 2 & \dots & H-1 & H \\
\hline
0 & p_o + \sum_{i=1}^{Q} q_i & p_1 & p_2 & \dots & p_{H-1} & g_o \\
1 & \sum^{Q} q_i & p_o & p_1 & \dots & p_{H-2} & g_1 \\
\vdots & \vdots & & & & & \\
H-1 & \sum_{i=H-1}^{Q} q_i & q_{H-2} & q_{H-3} & \dots & p_o & g_{H-1} \\
H & 0 & 0 & 0 & \dots & 0 & 1
\end{array}$$

where

$$g_i = 1 - \left(\sum_{j=i}^{Q} q_j + \sum_{j=0}^{H-i-1} p_j \right), \quad i = 0, 1, .. H-1 \tag{1}$$

P_H is then the probability of being absorbed <u>before</u> returning to the zero state.

Define

$${}_k f_{iH}^{(n)} = P\,[\Delta_n = H,\ \Delta_i \neq k,\ i = 1, 2, \dots n-1 \mid \Delta_o = i\,], \quad \begin{array}{l} i=0, 1, .. H-1 \\ k=0, 1, .. H-1 \end{array} \tag{2}$$

Then, starting at the origin, ${}_{o}f_{oH}^{(n)}$ is the probability of absorbtion at the nth step without visiting state 0 "en route", and hence

$$P_H = \sum_{n=1}^{\infty} {}_{o}f_{oH}^{(n)} \tag{3}$$

To solve this equation, we have

$${}_{o}f_{oH}^{(n)} = \sum_{j=1}^{H-1} P_{oj} \; {}_{o}f_{jH}^{(n-1)} \tag{4}$$

and

$${}_{o}f_{jH}^{(n)} = \sum_{k=1}^{H-1} P_{jk} \; {}_{o}f_{kH}^{(n-1)} \tag{5}$$

Define the H-dimensional vector $\underset{\multimap}{G}_o$

$$\underset{\multimap}{G}_o = \begin{bmatrix} {}_{o}f_{oH}^{(1)} \\ {}_{o}f_{1H}^{(1)} \\ \vdots \\ {}_{o}f_{H-1,H}^{(1)} \end{bmatrix} = \begin{bmatrix} g_o \\ g_1 \\ \vdots \\ g_{H-1} \end{bmatrix} \tag{6}$$

and let $[A]$ be the $H \times H$ matrix whose first column is zero

$$[A] = \begin{bmatrix} 0 & p_1 & p_2 & \cdots & p_{H-1} \\ 0 & p_o & p_1 & & p_{H-2} \\ \vdots & \vdots & & & \vdots \\ 0 & q_{H-2} & q_{H-3} & \cdots & p_o \end{bmatrix} \tag{7}$$

Then from Eq. (4), (5), (6) and (7) we have

$$\underline{G}_1 = \begin{bmatrix} {}_of^{(2)}_{oH} \\ \\ {}_of^{(2)}_{1H} \\ \vdots \\ {}_of^{(2)}_{H-1,H} \end{bmatrix} = [A]\, \underline{G}_o \tag{8}$$

Continuing in the same way we obtain in general,

$$\underline{G}_{n-1} = \begin{bmatrix} {}_of^{(n)}_{oH} \\ \\ {}_of^{(n)}_{1H} \\ \vdots \\ {}_of^{(n)}_{H-1,H} \end{bmatrix} = [A]^{n-1}\, \underline{G}_o \tag{9}$$

We can arrange each vector $\underline{G}_n$ as the (n+1)th column of a matrix $[B]$

$$\begin{aligned} [B] &= [\underline{G}_o \vdots \underline{G}_1 \vdots \ldots \vdots \underline{G}_n \vdots \ldots] \\ &= [\underline{G}_o \vdots [A]\,\underline{G}_o \vdots [A]^2\,\underline{G}_o \vdots \ldots \vdots [A]^n\,\underline{G}_o \vdots \ldots] \\ &= [I \vdots [A] \vdots [A]^2 \vdots \ldots \vdots [A]^n \vdots \ldots)\,\underline{G}_o] \end{aligned} \tag{10}$$

From the definition of the vectors $\underline{G}$'s and Eq. (3) we see that the sum of the elements of the first row of $[B]$ is P_H. Since $[A]$ is the truncation of a stochastic matrix we have

$$I + [A] + [A]^2 + \ldots = [I - [A]]^{-1} \tag{11}$$

and hence

$$P_H = x_1 \tag{12}$$

where x_1 is the first element of the vector

$$\underline{X} = [I - [A]]^{-1} \underline{G_o} \; . \tag{13}$$

REFERENCES

Berlekamp, E.R. (1968), Algebraic Coding Theory, McGraw Hill, N.Y., 1968.

Bucher, E.A., (1970), "Sequential decoding of systematic and non-systematic convolutional codes with arbitrary decoder bias", IEEE Trans. on Information Theory, Vol. IT-16, pp. 611-624, Sept. 1970.

Falconer, D.D., (1967), "A hybrid sequential and algebraic decoding scheme", Ph.D. Dissertation, Depart. of Elect. Eng., M.I.T., Feb. 1967.

Fano, R.M. (1963), "A heuristic discussion of probabilistic decoding", IEEE Trans. on Information Theory, Vol. IT-19, pp. 64-73, April 1963.

Feller, W., (1966), "An introduction to probability theory and its applications", John Wiley, N.Y. 1966.

Forney, G.D. Jr., (1967), "Coding system design for advanced solar missions", Natl. Aeron. and Space Adm., Final report, Contract NAS2-3637, Codex Corp., Watertown, Mass., December 1967.

Forney, G.D. Jr., (—), "Convolutional codes III, sequential decoding", IEEE Trans. on Information Theory, to be published.

Gallager, R.G., (1968), "Information Theory and Reliable Communication", John Wiley, N.Y. 1968.

Gallager, R.G., (1974), "Tree encoding for symmetric sources with a distortion measure", IEEE Trans. on Information Theory, Vol. IT-20, pp. 65-76, Jan. 1974.

Geist, J., (1970), "Algorithmic aspects of sequential decoding", Ph.D. Dissertation, Depart. of Elect. Eng., University of Notre Dame, Aug. 1970.

Haccoun, D., (1966), "Simulated communication with sequential decoding and phase estimation", S.M. thesis, Dept. of Elect. Eng., M.I.T., Sept. 1966.

Haccoun, D., and Ferguson, M.J., (1973), "Adaptive sequential decoding", 1973 IEEE International Symposium on Information Theory, Ashkelon, Israel, June 1973.

Heller, J.A. (1967), "Sequential decoding for channels with time varying phase", Ph.D. dissertation, Dept. of Elect. Eng., M.I.T., Aug. 1967.

Heller, J.A., and Jacobs, I.M., (1971), "Viterbi decoding for satellite and space communication", IEEE Trans. on Communication Technology, Vol. Com-19, pp. 835-848, Oct. 1971.

Jacobs, I.M., (1967), "Sequential decoding for efficient communication from deep space", IEEE Trans. on Communication Technology, Vol. Com-15, pp. 492-501, Aug. 1967.

Jacobs, I.M., and Berlekamp, E.R., (1967), "A lower bound to the distribution of computation for sequential decoding", IEEE Trans. on Information Theory, Vol. IT-13, pp. 167-174, April 1967.

Jelinek, F. (1969a), "A fast sequential decoding algorithm using a stack", IBM Journal of Research and Develop., Vol. 13, pp. 675-685, Nov. 1969.

Jelinek, F. (1969b), "An upper bound on moments of sequential decoding effort", IEEE Trans. on Information Theory, Vol. IT-15, pp. 140-149, Jan. 1969.

Jordan, K., and Bluestein, G., (1963), "An investigation of the Fano sequential decoding algorithm by computer simulation", Group report 62G-3, Lincoln Lab., M.I.T., July 1963.

Larsen, K.J. (1973), "Short Convolutional Codes with Maximal Free Distance from Rates 1/2, 1/3 and 1/4", IEEE Trans. on Information Theory, Vol. IT-19, pp. 371-372, May 1973.

Massey, J.L., (1963), "Threshold decoding", M.I.T. Press, 1963.

Massey, J.L., and Sain, M.K., (1969), "Distribution of the minimum cumulative metric for sequential decoding", 1969, IEEE International Symposium on Information Theory, Ellenville, N.Y., Jan. 1969.

Massey, J.L., Sain, M.K., and Geist, J.M., (1972), "Certain infinite Markov chains and sequential decoding", Discrete Mathematics, Vol. 3, pp. 163-175, Sept. 1972.

Niessen, C.W.,((1965), "An experimental facility for sequential decoding", Sc.D. Thesis, Depart. of Elect. Eng., M.I.T., Feb. 1965.

Odenwalder, J.P., (1970), "Optimal decoding of convolutional codes", Ph.D. Dissertation, Depart. of Elect. Eng., U.C.L.A., Jan. 1970.

Omura, J.K., (1969), "On the Viterbi decoding algorithm", IEEE Trans. on Information Theory, Vol. IT-15, pp. 177-179, Jan. 1969.

Savage, J.E., (1965), "The computation problem with sequential decoding", M.I.T. Lincoln Lab. Technical Report No. 371, Feb. 1965.

Shannon, C. E., (1948), "A mathematical theory of communication", Bell System Technical Journal, Vol. 27, July and Oct. 1948.

Viterbi, A.J., (1967), "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm", IEEE Trans. on Information Theory, Vol. IT-13, pp. 260-269, April 1967.

Viterbi, A.J., (1971), "Convolutional codes and their performance in communication systems", IEEE Trans. on Communication Technology, Vol. Com-19, pp. 751-772, Oct. 1971.

Wozencraft, J.M., (1957), "Sequential decoding for reliable communication", Sc. D. Thesis, Depart. of Elect. Eng., M.I.T., June 1957.

Wozencraft, J.M., and Jacobs, I.M., (1965), "Principles of Communication Engineering", John Wiley, N.Y., 1965.

Wright, D., (1967), "Sequential decoding with a fading dispersive channel", S.M. Thesis, Depart. of Elect. Eng., M.I.T., April 1967.

Zigangirov, K., (1966), "Some sequential decoding procedures", Problemii Peredachi Informatsii, Vol. 2, No. 4, pp. 13-15, 1966.