Titre: Title:	An analysis of sequential decoding with retransmission procedures
Auteurs: Authors:	Pierre-Yves Pau, & David Haccoun
Date:	1985
Type:	Rapport / Report
Référence: Citation:	Pau, P.-Y., & Haccoun, D. (1985). An analysis of sequential decoding with retransmission procedures. (Rapport technique n° EPM-RT-85-19). https://publications.polymtl.ca/9509/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/9509/
Version:	Version officielle de l'éditeur / Published version
Conditions d'utilisation: Terms of Use:	Tous droits réservés / All rights reserved

Institution:	École Polytechnique de Montréal
Numéro de rapport: Report number:	EPM-RT-85-19
URL officiel: Official URL:	
Mention légale: Legal notice:	

EPM/RT-85-19

AN ANALYSIS OF SEQUENTIAL DECODING WITH RETRANSMISSION PROCEDURES*

by

Pierre-Yves Pau M.Sc.A.

and

David Haccoun Ing., Ph.D.

Department of Electrical Engineering
Ecole Polytechnique de Montréal
May 1985

* This report summarizes the results contained in the thesis "Décodage séquentiel avec retransmissions", submitted by Pierre-Yves Pau under the direction of David Haccoun, in partial fulfilment for the requirements of the Master's Degree in Applied Sciences (May 1984). This research was supported in part by the national Science and Engineering Research Council of Canada.

ABSTRACT

This report describes an analyses techniques that allow a sequential decoder to work as an effective hybrid FEC-ARQ system. These techniques use the computational decoding effort as an indicator as to wether decoding of the current data block should be stopped, and its retransmission requested using a selective ARQ protocol. Two different approaches are implemented on the Zigangirov-Jelinek (stack) algorithm of sequential decoding. Computer simulation results show that both approaches yield excellent results, even on very noisy channels, in terms of throughput efficiency and decoder memory requirements. Furthermore the main sequential decoding drawback, namely the computational variability, is shown to be significantly alleviated. The overflow probability of the input buffer of the decoder is also reduced. Implementation of these techniques results in making sequential decoding a high performance hybrid FEC-ARQ system.

I. INTRODUCTION

For discrete memoryless channels (such as the space channel), systems employing convolutional encoding at the input and probabilistic decoding at the output are among the most attractive means of approaching the reliability of communication promised by the theory. One of the principal probabilistic decoding techniques in Foward Error Correction (FEC) with convolutional codes is sequential decoding. There are two main sequential decoding algoritms: the Fano Algorithm [1] and the Zigangirov-Jelinek (Z-J) Algorithm [2]. Sequential decoding involves a random motion in the coded tree, leading to a variable number of computations to decode a given bit of information. This number of computations C has an asymptotic Pareto distribution, that is a distribution whose tail decreases only algebraically, and given by:

$$P(C > x) = \beta x^{-\alpha} \qquad x \gg 1 \qquad (1)$$

The exponent α is called the Pareto exponent and depends only on the code rate and channel transition probabilities. As for β, it is a constant which depends among other factors on the type of decoder that is used. This computational variability is one of the main drawbacks of sequential decoding. However provided the coding rate R is smaller than some rate Rcomp, typically the average number of computations per decoded information bit, Cav, is very small. At rates beyond Rcomp, Cav becomes unbounded; Rcomp is called the computational <u>cut-off rate</u> and constitutes the limiting rate of operation for sequential decoders. Nevertheless there is a nonzero probability for the number of computations to decode one bit to become infinite [3]. Since a decoder has limited physical resources, this computational variability will occasionally lead to buffer overflows and information erasures and several techniques have been developed to reduce it [4],[5].

In order to accomodate the consequences of erasures, data is usually transmitted in blocks a few hundred to a few thousand bits long. Because of the variability of the computational effort a buffer is required at the input of the decoder in order to store the incoming data waiting to be decoded. Should the decoding of a block require an excessive amount of computations then an overflow would occur with its ensuing erasures. However this difficult block may be discarded from the input buffer and decoding of the following block may be safely resumed.

In this paper we consider sequential decoding (Z-J Stack Algorithm) used in conjunction with retransmission procedures (ARQ). In the Stack Algorithm, the decoder memory consists of a <u>list</u> or "stack" of previously searched paths, ordered with respect to their likelihood or "metric" values. In the Stack Algorithm computational variability may result in an overflow either in the stack or the input buffer. Clearly an overflow of the input buffer is a catastrophic event, leading to loss of synchronization and possible erasure of many information blocks. On the other hand whenever an overflow occurs in the stack, decoding of the faulty block stops, its remaining part is flushed out of the input buffer, and decoding can be resumed for the next block. Stack overflows then appear to be much less troublesome than buffer overflows, and the use of a small stack will prevent a large accumulation of data in the input buffer. For a given input-buffer size, the probability of buffer overflow will decrease, or equivalently, the buffer size may be reduced for a given overflow probability. Still better, further improvement may be obtained if those blocks which are likely to require too large a decoding effort are detected <u>before</u> a stack overflow occurs [6],[7]. Decoding of these blocks would then stop and a retransmission be requested. Furthermore, since difficult blocks are very often decoded in error, such a procedure will lead to both an improvement of the overall error probability and a reduction of the required decoder memory storage. However, like for any ARQ system, the system throughput may be slightly degraded, due to the retransmissions. Therefore, in conjunction with a proper retransmission procedure, a sequential decoder may be used for both error detection and error correction.

A significant amount of work has been done on hybrid FEC-ARQ schemes using convolutional coding and sequential decoding. Recently, Sacham [8] analyzed the performance of a Stop and Wait ARQ protocol associated with a decoder using the Fano Algorithm. The decoder used a time-out condition to determine whether the decoding of the current block had to be stopped and its retransmission requested. This so-called Time-out Algorithm" was also studied by Drukarev & Costello [6] over the Stack Algorithm, assuming a selective-repeat ARQ protocol. In both cases, results concerning the system throughput and optimum time-out duration were derived. An asymptotic upper-bound on the attainable overall error probability was also given in [6]. Finally, they proposed a method referred to as the Slope Control Algorithm, that monitors the metric behaviour in order to decide whether to abort the decoding of a block. However, the input-buffer problem mentionned above, that is crucial in sequential decoding, was not examined.

In this paper we present techniques that allow a sequential decoder to work as an effective hybrid FEC-ARQ system. A model of the system is described in Section II, and the parameters that may be considered as relevant indicators of its performance are be presented. These indicators are shown to be closely related to the computational effort of the decoder; their experimental values have been determined using computer simulations. The results obtained concerning the input buffer overflow probability are discussed in some detail. Several algorithms that allow the implementation of these techniques have been developed, and are then described. Stack Overflow Algorithms which attempt to limit the decoding effort by limiting the memory size of the stack are described in Section III. The influence of the stack size on the decoding effort is examined. Memory savings offered by the Stack Algorithm that uses a flag to store only the negative branch extension on the paths of the coded tree is also analyzed. An Anticipation Algorithm, that predicts that an overflow will occur if decoding proceeds further is also presented [7]. Section IV presents the so-called Slope Control Algorithm, that requests a retransmission whenever the cumulative metric drops below some predetermined threshold over a small path length. For each algorithm, we focus our attention on maximizing the throughput for given memory requirements. Extensive computer simulation results are given, and compared to the analytical results. Finally, the respective advantages of each algorithm are discussed in Section V, in terms of memory requirements and complexity toward possible practical implementations.

II. SYSTEM MODEL AND ANALYSIS

2.1 MODEL AND PARAMETERS OF INTEREST

The system model consists of a forward binary symmetric channel (BSC) perturbed by zero-mean additive white gaussian noise (AWGN). Antipodal signalling is assumed, as well as a perfect selective-repeat ARQ protocol over a noiseless feedback channel. The sequential decoder uses the quantized Z-J Algorithm with the optimum Fano metric.

The system performance has been evaluated using a set of 3 main parameters, namely: transmission efficiency, complexity and memory requirements of the decoder. As for the error performance, it is well known that very small error probabilities can be obtained with sequential decoding using codes of moderate to long constraint lengths ($k > 20$). For hybrid FEC-ARQ using sequential decoding the error probability will be further decreased since those blocks that are discarded often yield decoding errors. Consequently the system error probability could not be efficiently measured by reasonable simulations using the available computer resources. Therefore it has not been investigated in any detail.

Two different parameters can be used in order to evaluate the system efficiency: the Transfer Rate of Information Bits θ (TRIB), and the throughput efficiency η [9]. TRIB can be defined as the inverse of the overall time required to decode one information bit; here this time may be measured in number of decoder operations. However this does not yield accurate evaluation, since each operation may require a highly variable time. Further discrepancy is likely to be introduced by those waiting periods that inevitably occur when a block is discarded before the next one becomes available in the input buffer. Because of all these inaccuracies, it is difficult to rely on TRIB measurements that may be obtained from simulations, although theoretically this parameter may give good indications about the decoder behaviour.

The throughput efficiency is defined as the number of decoded information bits per transmitted channel symbol. Every symbol that has been transmitted, demodulated, and stored in the input buffer, is taken into account, even if it is finally not decoded due to a retransmission request. In an ARQ system ,using simple counting procedures, this latter parameter is thus likely to yield easier and reliable performance evaluation for the overall transmission link. Throughput efficiency has thus finally been preferred over TRIB. However, TRIB will be retained as a criterion of the decoding efficiency.

As for decoder memory requirements, it would be unrealistic not to consider them. For a stack decoder the relatively large memory requirement is the counterpart of the algorithm moderate complexity. Furthermore, the input buffer overflow probability, and hence the buffersize, depends critically on the other system parameters, as will be seen below.

Computer simulation was used in order to evaluate the decoder parameters discussed above. As we were interested in examining critical situations, the channel utilization rate was chosen to be R/Rcomp=0.99, corresponding to a signal to noise ratio Eb/No=4.65 dB. A sufficient number of retransmissions could thus be obtained with simulations of reasonable duration. The code was a robust rate 1/2 systematic code of constraint length 24, defined by its octal generator 67115143 [10]. The data is organized in blocks of length 500 bits to which a tail of 23 known bits is appended. As for the duration of simulations, a rule of thumb is that they should go on until the collected statistics satisfy some maximum standard deviation criterion. However, this difficult to apply when many statistics of heterogenous nature have to be gathered from the same simulation. It has therefore been decided to keep a comparative approach, except for the cases where precise absolute values were needed. This led to simulations of moderate but identical duration, run using the same pseudo-random noise sequences.

2.2 LIMITATION OF THE DECODING EFFORT

As mentionned in Section I, the principle of ARQ strategy in sequential decoding is to detect and retransmit those data blocks that are likely to require a very large decoding effort. Such a strategy can be easily implemented by limiting the total number of computations allowed for the decoding of a block. This will however have a significant influence on the observed computational distributions. Likewise the system throughput will be affected by the retransmissions. The purpose of this Section is to address those problems.

In the absence of limiting constraints on the decoding effort, the computational distribution is given by (1). As for the Pareto exponent α, it can be evaluated in the vicinity of R/Rcomp=1 using the approximation:

$$\alpha = \frac{Rcomp}{R} \qquad (2)$$

which for R/Rcomp =0.99 yields the value α=1.01. The constant β can be readily obtained by matching a straight line of slope α (logarithmic scale) over the experimental computational distribution obtained from simulation. Figure 1 depicts the operation. In our case this yields α=0.14. It is usually admitted [11] that the total number of computations for an L bits block (tail included), here referred to as C_L, also follows a Pareto distribution with exponent α, namely:

$$P(C_L > x) = L\, P(C > x) \qquad (3)$$

Figure 2 shows a distribution of C_L as obtained from simulation, and the corresponding approximation as extrapolated from (1) and Figure 1 Now let $C_L max$ be the maximum number of computations allocated for the decoding of a single block. Those blocks that require more computations will be discarded and thereafter retransmitted. From (1) and (3), the probability Pb of this event is:

$$Pb = L\beta C_L max^{-\alpha} \qquad (4)$$

For the case of selective-repeat ARQ, the system throughput η is then given by:

$$\eta = R\,(1 - L\beta C_L max^{-\alpha}) \text{ bits/symbol} \qquad (5)$$

where R is the code rate [9]. The system throughput is thus an increasing function of $C_L max$.

Clearly limiting C_L will influence the computational distribution $P(C>x)$. Actually, it can be assumed that the number of computations devoted to the decoding of a single bit will be limited to some value $Cmax \leq C_L max$, the exact relation between Cmax and $C_L$max depending, among other factors, on the particular algorithm that will be used to implement the retransmission strategy. From (1), the distribution function $P(C<x)$ is:

$$F(x) = P(C \leq x) = \begin{cases} 1-\beta x^{-\alpha} & x \gg 1 \\ 0 & \text{elsewhere} \end{cases} \quad (6)$$

Conditionning on $C \leq Cmax$ yields the conditional distribution function $Fc(x)$:

$$Fc(x) = \frac{P(C \leq x \ , \ C \leq Cmax)}{P(C \leq Cmax)} \quad (7)$$

that is

$$Fc(x) = P(C \leq x | C \leq Cmax) = \begin{cases} \beta \dfrac{1-\beta x^{-\alpha}}{1-\beta Cmax^{-\alpha}} & 1 \leq x \leq Cmax \\ 1 & Cmax < x < \infty \\ 0 & \text{elsewhere} \end{cases} \quad (8)$$

Differentiating with respect to x we obtain the conditional density function:

$$fc(x) = \begin{cases} \dfrac{1}{1-\beta Cmax^{-\alpha}} \left[\alpha\beta x^{-\alpha-1} + (1-\beta) Uo(x-1) \right] & 1 \leq x \leq Cmax \\ 0 & \text{elsewhere} \end{cases} \quad (9)$$

where Uo is the dirac delta function. The ith moment of the conditional distribution is then readily obtained as:

$$\overline{C^{i}} = \begin{cases} \dfrac{1}{1-\beta Cmax^{-\alpha}} \left[\dfrac{\alpha\beta}{i-} \left(Cmax^{1-\alpha} - 1 \right) + (1-\beta) \right] & \alpha > i \\ \text{Log } Cmax + (1-\beta) & \alpha = i \end{cases} \quad (10)$$

For $Cmax = \infty$, the ith moment will be bounded iff $\alpha > i$, which is typical of sequential decoding. On the other hand, we notice that for any $Cmax < C_L max < \infty$, the moments will be artificially prevented from diverging, at the price of decoding failures. Thus, the distribution can not be asymptotically Pareto. By convention the first moment will be referred to as Cav. Finally from (8) we have:

$$P(C>x|C<Cmax) = 1-Fc(x) = \begin{cases} \beta \dfrac{x^{-\alpha} - Cmax^{-\alpha}}{1 - \beta Cmax^{-\alpha}} & 1 \leq x \leq Cmax \\ 0 & Cmax < x < \infty \\ 1 & \text{elsewhere} \end{cases} \qquad (11)$$

Figure 3 illustrates the theoretical conditional distributions obtained by choosing for Cmax some arbitrary values ranging from 100 to 10,000. It can be seen that the tail of the distributions are no longer Pareto, and decrease asymptotically to a vertical straight line of equation x=Cmax.

2.3 THE INPUT BUFFER PROBLEM

The symbols arriving from the channel induce in the input buffer a queuing process. For a rate 1/2 code, there are 2 symbols per branch of the coded tree. While the interarrival time between 2 consecutive branches is constant, the service (i.e. decoding) time for each branch is randomly distributed. In our case a single decoder is operating. Thus, in the terminology of queuing theory the input buffer waiting line is a D/G/1 queue, i.e. a particular case of the G/G/1 queue [12]. Defining the speed factor U of the decoder as the number of computations that it can perform in one interarrival time, the condition for waiting line stability can be written as:

$$U > Cav \qquad (12)$$

where Cav is proportional to the mean required service time, while U is proportional to the offered service time. Let W be the waiting time for one branch in the input buffer. If (12) is verified, then a stationary distribution P(W>y) exists for the random variable W [12]. Note that, if we choose the time unit to be the interarrival time, the random variable W will also represent the number of branches stored in the input buffer.

It has been shown [13] that, if B is the input buffer storage in number of branches, and provided the speed factor U is large enough, the probability that the buffer will overflow during the decoding of the first bit of a block is given by:

$$Pov = P(C > BU) \qquad (13)$$

Using (11) and assuming U > Cav we obtain:

$$Pov = \begin{cases} \beta \dfrac{(BU)^{-\alpha} - Cmax^{-\alpha}}{1-\beta Cmax^{-\alpha}} & 1 \leq BU \leq Cmax \\ \emptyset & BU > Cmax \end{cases} \qquad (14)$$

This is consistent with the results obtained in [13] using the Fano Algorithm, since for that algorithm we have Cmax=oo. Thus Pov is increasing with Cmax , the allowed maximum number of computations per bit. On the other hand, the system throughput is an increasing function of $C_L$max, the maximum number of computations per block. Thus, a "good" algorithm for limiting the decoding effort will tend to minimize Cmax and maximize $C_L$max, so as to get the best trade off between the throughput and the. waiting line.

Although (14) leads to that interesting result, the argument cannot be generalized to the other bits of the block unless U is very large, which is not very practical. From an engineering point of view, it may be actually preferable to operate with a reduced speed factor. In the case of the Stack Algorithm, where $C_L$max is determined by the memory-size of the stack, it has been shown [14] that an input buffer of size B > 2L will never overflow provided that:

$$U > \frac{C_L max}{L} \qquad (15)$$

Therefore, it would be interesting to examine the waiting line behaviour in the interval:

$$Cav < U < \frac{C_L max}{L} \qquad (16)$$

As U -> Cav, the mean service time tends to match the interarrival time, although always from below in order to insure stationarity. It is then shown [12] that, for any G/G/1 queue, the distribution of the waiting time W can be robustly approximated by an exponential function, that is:

$$P(W>y) = Fw(y) = 1 - e^{-So\ y} \qquad (17)$$

This is called the Heavy-Traffic approximation. The mean value of the random variable W is then:

$$\bar{W} = -\frac{1}{So} \qquad (18)$$

An interesting feature of this approximation is that its domain of validity can be easily verified by simulation, through (18). The observed average waiting-line will be used to compute the exponent in the right-hand side of (17), and the resulting theoretical distribution will then be matched with the observed waiting-line distribution.

On the other hand, as long as condition (12) is verified and $C_L$max finite, the worst possible waiting-line distribution will be exponential in the vicinity of U=Cav. As U increases, the empirical distribution should tend progressively toward the theoretical distribution given by (14), with its tail bounded by the corresponding exponential distribution (such as So=$1/\bar{W}$). In a strict sense, (14) holds true for the first bit of a block, since the input buffer is empty of all preceding bits of that block. Expression (14) will therefore hold also for any other bit provided the same condition concerning the buffer is satisfied for that bit. However regardless of the state of the input buffer (17) may be considered as an asymptotic upper bound for any bit in the block with the exponent So computed by measuring the average waiting-line.

III. STACK OVERFLOW ALGORITHMS

The strategies analyzed above can be implemented through a variety of techniques. This section describes a method that uses the decoder stack memory to limit the decoding effort. For a rate 1/2 code, each computation performed by the decoder results in the storing of 2 extensions in the stack. The stack memory-size can thus be used as a limiting constraint. We successively describe 3 variants of this method, in an approach that aims to improve the throughput vs. waiting-line trade-off.

3.1 VARIANT 1: SIMPLE STACK OVERFLOW ALGORITHM.

In this variant, the algorithm checks for the availability of stack storage at every path extension. A stack overflow triggers a request for a retransmission of the block. The algorithm is:

1. Extend highest metric node of the stack and check for available stack storage. If storage is available enter extensions in stack and go to 2. Otherwise go to 3.
2. Search and extract from stack new highest metric node. If this node is a terminal node, end of decoding. Otherwise go to 1.
3. Stop decoding. Flush out of the input buffer all branches belonging to current block, and request retransmission of the block. Start decoding next block if present.

The throughput of this variant is readily calculated. Let "S" be the size of the stack. Assuming S to be an even integer number, not more than $S/2 = C_L\max$ computations may be devoted to the decoding of one block. Substituting in (5) the system throughput is then:

$$\eta = R\ (1 - L\beta\, 2\, S^{-\alpha}) \qquad (19)$$

As for the maximum number of computations possible for the decoding of the ith bit of the block, it will depend on the number of nodes that have already been stored in stack. As each decoded bit requires a minimum of 1 computation, Cmaxi, the maximum number of computations for the ith bit of the block is given by:

$$\mathrm{Cmax}_i = \frac{S}{2} - i + 1 \qquad (2\emptyset)$$

which is obviously upper-bounded by Cmax1. Let Ps(C>x) be the resulting distribution. Since Cmaxi < Cmax1 then Ps(C>x) will be upper-bounded by the distribution obtained by conditionning (1) on C≤Cmax1, i.e.:

$$\mathrm{Ps}(C>x) < P(C>x \mid C\leq \mathrm{Cmax1}) = \begin{cases} \beta\dfrac{x^{-\alpha} - (S/2)^{-\alpha}}{1-\beta(S/2)^{-\alpha}} & 1 \leq x \leq S/2 \\ \emptyset & S/2 < x < \infty \\ 1 & \text{elsewhere} \end{cases} \qquad (21)$$

Simulations have been performed using various stacksizes, so as to investigate the influence of this latter parameter. Figure 4 displays a set of the observed distributions. Comparison with Figure 3 corroborates the analysis that led to expressions (11) and (21). Figure 5 shows the influence of the stacksize on Cav. The monotone increasing behaviour of this curve is in good agreement with expression (1∅), since in our case α is very near to 1.

Figure 6 illustrates the waiting-line behaviour for the cases represented by Figure 4 The throughput values that have been obtained confirm the fact that discarding a small proportion of noisy blocks yields considerable improvement on the input buffer overflow probability. For each curve, we indicate the corresponding Cav/U ratio, as well as the observed average waiting-line $\overline{W}$. The latter was used for computing the So exponent in (17) using (18). The resulting exponential distributions are indicated in dotted lines. It turns out that the Heavy-Traffic approximation (17) is still roughly holding for Cav/U as low as ∅.57. For smaller values of this ratio, note however that the observed distributions fall almost vertically towards zero, which suggests they are likely to be asymptotically upper-bounded by their associated exponential distribution. Indeed, this is even a certitude in Figure 6 for Cav/U=0.53 and Cav/U=0.48: it results from (15) that the waiting line cannot exceed 2L branches.

3.2 VARIANT 2: STACK OVERFLOW ALGORITHM WITH FLAG.

In variant 2 we add to variant 1 a procedure usually meant to save memory storage in the stack [15]. As the best path gets extended in the coded tree, 2 extensions stem out of its terminal node, and their branch metric is computed. Whenever a branch metric is positive, this branch is not stored in in the stack since it will necessarily be extended at the next cycle. the negative branch extension is stored, and a flag is used to indicate that only that branch extension has been stored. This procedure is usually incorporated in stack decoders and is called the "flag" procedure. As a result there is a virtual increase of the stack memory size.

As for the implementation we followed [3],[16]. Let Np_L be the number of positive extensions encountered throughout the decoding of a particular block. Averaging over the ensemble of transmitted blocKs, we have:

$$\overline{Cmax}_L = \frac{S+\overline{Np}}{2}L \qquad (22)$$

where $\overline{Np}_L$ is the average value of Np_L. From (19), the average system throughput is obtained as:

$$\eta = R\ (1-L\beta 2^{\alpha}(S+\overline{Np}_L)^{-\alpha}) \qquad (23)$$

Higher throughput efficiency is thus available using the same stack size. Now let Npi be the number of positive extensions generated as decoding the ith bit of a block, i=1,L. This number is a random variable which depends on the the current noise level on the channel. As this noise is white gaussian, stationarity may be assumed and hence all Npi have the same distribution. Consequently:

$$\overline{Np}_L = L\ \overline{Np}i \qquad i=1,L \qquad (24)$$

Now the maximum number of computations that can be performed for the decoding of the ith bit of a block is also a random variable, the average value of which is given by:

$$\overline{Cmax}_i = \frac{S + \overline{\left\{\sum_{j=1}^{j=i} Npj\right\}}}{2} - i + 1 = \frac{S + i\,\frac{\overline{Np}}{L}L}{2} - i + 1 \qquad i=1,L \quad (25)$$

Comparison with expression (22) indicates that, for any bit of a block except the last one, $\overline{Cmaxi}$ is smaller than $\overline{C_L max}$. For the last bit however, we have i=L and $\overline{CmaxL} = \overline{C_L max}$. Now substituting these values respectively in (14) and (5) shows that, for a given system throughput, the buffer overflow probability will be in no case greater than for variant 1. Figure 7 compares the throughput vs. stacksize relations for variants 2 and 1.

3.3 VARIANT 3: ANTICIPATION ALGORITHM.

Whenever a noise burst occurs on the channel, it might cause the decoder to increase the number of computations up to a point where there is just not enough stack memory available to complete decoding of the block. Beyond this point, clearly every further computation is a waste, since the block will have to be retransmitted anyway. It is thus useless to wait until the stack actually overflows. Consequently in Variant 3 we imbed in Variant 2 a test performed to anticipate the overflows. However the anticipation test should in no case prematurely discard a block that could have been eventually decoded. Thus, when performing this test we must assume that each of the remaining branches of the block will require the strict minimum amount of computations, i.e. one. The coded block consists of L branches. If k is the constraint length of the code, then the last (k-1) branches, referred to as the "tail" of the block, are known to represent Ø's. Hence in terms of entries in the stack, each computation implies a minimum of:

- A single entry before the tail is reached.
- No entry in the tail.

The test is then described as follows:

1. Compute Sr, the remaining number of entries available in the stack (This is readily obtained from the address of the last node entered in the stack).

2. Compute Sn, the minimum number of entries needed in order to complete decoding. Let Dmax be the maximum depth currently reached in the coded tree. We then have:

$$Sn = (L-k+1)-Dmax \qquad (26)$$

3. If Sr is smaller than Sn, then decoding of the block cannot be completed; it must be retransmitted. Otherwise proceed with decoding.

As no block that would have been decoded using variant 2 will be discarded with variant 3, the system throughput remains unaffected and is given by (23). On the other hand it is easily understood that, except in the tail of the block where overflow cannot be anticipated, the average maximum number of computations for the ith bit of the block (tail excluded) is reduced, as compared to (25). Actually if the ith bit causes the stack to overflow, then Sn= (L-k+1)-Dmax= (L-k+1)-i, which means there are still (L-k-i+1) entries left in the stack when decoding stops. Consequently and from (25):

$$\overline{Cmax}_i = \frac{S + i\,\frac{\overline{Np}}{L}L - L+k+i-1}{2} - i+1 \qquad i=1,L-k+1 \qquad (28)$$

Comparison with (25) indicates that $\overline{Cmaxi}$ is on the average smaller by a quantity (L-k-i+1)/2. It stems from (14) that the probability of buffer overflow is then decreased. Figure 8 shows an example of the observed waiting line distributions, obtained with the same system throughput with and without anticipation. It is worth mentionning that the observed improvement is obtained at a minimal cost, since we only make use of a physical constraint that is inherent to the decoder, namely the limited stack memory size.

3.4 PRACTICAL CONSIDERATIONS

In order to compare the performances of the 3 variants, Figure 9 plots the throughput vs. average waiting-line relationship for each of the 3 variants. It becomes clear then how each new variant has improved over the preceding one. It should be however pointed out that all 3 variants asymptotically yield the same performance, as the system throughput gets larger by using a very large stack. The reason is that improvement is always obtained from those blocks that cause overflows - consequently, the fewer the overflows, the smaller the improvement. Furthermore, the number of computations that can be anticipated before an overflow occurs is upper-bounded by (L-k), and becomes small compared with $C_L$max as this latter increases. from that point of view, the methods described above offer some similarities with the so-called Time-out Algorithms described in [17], [6], [8].

Further to the above remarks, one may wonder whether some optimization criterion may exist in order to determine what is the best stack memory size. A possible criterion [6] is to maximize the efficiency of decoding in terms of delivered bits per decoder computation, that is maximizing:

$$\theta = \frac{1}{Ctot} \quad \text{bits/comp.} \qquad (29)$$

Where Ctot represents the total number of computations required for the decoding of one information bit, taking into account any retransmission. This parameter will be referred to as the system trib, because of its similarity with the TRIB mentionned in section I. Figure 10 plots the system trib vs. stacksize relation for each of the 3 variants. We observe that higher trib values are obtained with variants 2 and 3. The explanation lies in the dependency of Cav over Cmaxi (10), which is smaller for variants 2 and 3. It should also be pointed out that for variants 2 and 3 these maximum trib values are obtained with smaller stack memory size.

Regardless of the variant used, the main advantage of the methods described above resides in their simplicity of implementation. Noteworthy enough, the implementation of the algorithms would only require minimal modifications of a conventional FEC stack decoder. Additional required memory would be negligible, with memory savings even possible in some cases. Furthermore no complexity is added to the Stack Algorithm. Variants 1 and 2 simply make use of test which is in all cases standard on a stack decoder. Even for variant 3, no further complexity is added since the information needed for performing the anticipation test is already present in the conventional Stack Algorithm. Furthermore this test can be performed simultaneously with the test for stack-overflow, and requires no additional processing of information [18].

The final choice of such parameters as the stack memory size or decoder speed factor, and the trade off between them, becomes an engineering problem. Figure 11 illustrates the relationships between speed factor, stack memory size and waiting line for variant 3 when used at Eb/No= 4.65 dB. Family of curves similar to those given in Figures 10, 7, and 5 are helpful for making intelligent trade offs leading to an efficient system design based on a given technology.

Another kind of practical problem concerns the length of the data blocks. In our case the length was chosen to be 500 bits, somewhat arbitrarily. Since the request for retransmission of a block may occur when a substantial part of this block has already been decoded, it may appear attractive to retransmit only the undecoded part of the block. This could be achieved using Path Memory Truncation techniques [19], [6]. However these techniques are not further investigated in this paper. Beyond the necessity for reinitializing the decoder itself, the main drawback of this scheme lies in the necessity for resynchronizing the transmitter and the receiver on time-slots of variable duration. The length of the blocks have an obvious impact on the decoder parameters. However it may be useful to optimize the block length, with respect to the throughput efficiency of the retransmission protocol. The throughput efficiency of a selective repeat ARQ protocol is theoretically insensitive to the block length when the coding rate is constant [20]. However since each block of length λ is appended with a tail of (k-1) known bits, the actual coding rate is:

$$\mathrm{Reff} = R\,\frac{\lambda}{\lambda+k-1} \qquad (30)$$

Hence for a given code, reducing the block length reduces the actual coding rate and hence the throughput. For a rate 1/2 code, expression (5) can thus be rewritten as:

$$\eta = \frac{\lambda}{2(\lambda+k-1)}\left(1-(\lambda+k-1)\beta C_L \max^{-\alpha}\right) \qquad (31)$$

Differentiating with respect to λ yields:

$$\frac{1}{2}\left[\frac{k-1}{(\lambda+k-1)^2} - \beta C_L \max^{-\alpha}\right] \qquad (32)$$

and hence:

$$\lambda\mathrm{opt} = \sqrt{\frac{k-1}{\beta} C_L \max^{\alpha}} - k+1 \qquad (33)$$

The condition for a maximum:

$$\left.\frac{d^2\eta}{d\lambda^2}\right|_{\lambda=\lambda\mathrm{opt}} = -\frac{k-1}{(\lambda\mathrm{opt}+k-1)^3} < 0 \qquad (34)$$

Which is verified for any $\lambda\mathrm{opt}>0$. Our choice for λ is in general slightly larger than the optimal value, but had very little consequence of the throughput.

4. IV. SLOPE CONTROL ALGORITHM

A common feature in the above methods is that the decoding of a block proceeds until it is realized that its successful completion is impossible. Waiting until the successful decoding is impossible often entails a waste in the decoding effort, since the block will have to be retransmitted anyway. It may be interesting to make such a decision earlier. Naturally such a procedure may lead to unnecessary retransmissions for some of the blocks. Reliable indicators can be found to predict that computational difficulties are to be expected. One such indicator is the correct path metric behaviour. Since a severe span of noise on the channel causes the metric to drop very rapidly, the rate of drop may be used as an indicator that the block is likely to require an excessive amount of computations. Such a method called Slope Control Algorithm is described next.

4.1 DESCRIPTION AND IMPLEMENTATION OF THE ALGORITHM

The principle of the Slope Control Algorithm is to monitor the metric behaviour with regard to a predetermined criterion, in order to decide whether the current block should be discarded and its retransmission requested. Let M(i) to be the cumulated metric of a given path at depth i, and let δM (i), the change in metric over the last "F" branches, be the "metric slope" at depth i. "F" being an integer number,

$$\delta M_F(i) = M(i) - M(i-F) \qquad (35)$$

"F" will be referred to as the "window length" [6]. The algorithm is then described as follows:

1. Extend current node. Store only these extensions for which:

   $$\delta M_F(i) > T \qquad i=1,L \qquad (36)$$

   Where "T" is some predetermined threshold value. Retain in a proper register the cumulated metric of the best discarded extension, M*.

2. Get the new top node from the stack. If its metric is not greater than M*, then go to 3. Else, go to 4.

3. Stop decoding. Flush out of the input buffer the remaining branches of the current block and request retransmission for it.

4. If maximum depth is reached, then end of decoding. Else, go to 1.

Clearly the implementation of this algorithm is more involved than for the Stack Overflow Algorithms. Two parameters must be set, namely, the window length F and the metric threshold T. Results obtained using asymptotic approximations such as (1) show that both have to be optimized in order to get the algorithm working properly [6]. Based on computer simulations, the observed system tribθ vs. threshold relation for F= 30 branches, is given in Figure 12 The simulations were repeated in order to optimize the threshold values for different values of F, allowing the determination of the system tribθ vs. window-length F relation, which is plotted on Figure 13 The obtained broad maximum for θ led to choose F between 24 and 30 branches.

As for the stack memory size, it should theoretically be chosen as large as possible, to minimize its influence on the retransmission events. However it should be noted that the Slope Algorithm automatically limits the number of nodes that will be stored in the stack, in a very efficient way: as a noise burst on the channel causes the number of computations to grow and fill the stack, the number of extensions that are discarded by the threshold T also grows significantly resulting in a regulating effect on the filling of the stack. In principle a procedure such as the flag could also be implemented, in order to reduce the required stacksize. Two problems however arise, and have to be thoroughly examinated:

1. As a positive extension gets discarded from the stack due to the flag procedure, its cumulated metric has to be retained in some way, since it may be needed F branches further in order to compute δM_F.
2. If a negative extension is discarded by the threshold, then the positive extension must be entered in stack in order to recover the decoded sequence.

As these problems would seriously increase the algorithm complexity, the flag was not used with the Slope Control Algorithm. However a simple counting procedure allows an evaluation of the memory savings that would be obtained if the flag was used. Hence the influence of the flag on the stack was only simulated.

4.2 SIMULATION RESULTS

The same model as described in section II was used in order to evaluate the performance of the algorithm. Figures 14 gives the computational distributions obtained with a stacksize of 10,000 nodes and several threshold values, varying from -181 to -41 and a window length F of 30 branches. The possible branch metrics being +6, -29, -64 the thresholds were thus set to detect correct path metric drops corresponding to 7, 8, 9, 10 and 11 code symbols received in error over the window length F. As the threshold is raised, the metric drops are detected sooner causing the decoding to stop and hence preventing the number of computations to build up. As shown in Figure 14 the tails of the distribution drop very rapidly with increasing theshold. Naturally raising the threshold also increases the probability of retransmission as illustrated in Figure 15

The memory utilization of the SCA has also been examined with and without the flag procedure, and compared to the usual stack algorithm. On Figure 16 the observed distributions of the total number of nodes S stored in the stack when decoding of a block stops is given. Three cases are examined:

1. Conventional Stack Algorithm, stacksize= 10,000 nodes: in spite of this significant stack memory storage, overflows did occur.
2. Slope Algorithm, without flag. A stack with 2,500 nodes storage would have accomodated all the decoded blocks.
3. Slope Algorithm with simulated flag: a stack with only 1,700 nodes memory storage would have been sufficient.

In the SCA the threshold was chosen to be T= -111 to maximize the system trib as shown in Figure 12. As the threshold is lowered fewer retransmissions may be due to an actual threshold violation. However as in any Stack Algorithm, the stack overflow phenomenon is always present. Hence depending on the stacksize a greater number of retransmissions may be due to a stack overflow rather tham a threshold violation. For example Figure 17 shows that with a stacksize of 2000 nodes, and a threshold of T= -111, approximately 25% of the SCA retransmissions are due to a stack overflow.

The waiting line distributions for several threshold values are given in Figure 18 As expected a lowering of the threshold results in an increased buffer overflow probability. In order to be able to draw comparisons with the Anticipation Algorithm, the throughput vs. average waiting line relations are plotted on Figure 19 for both algorithms.

From this point of view, the advantage clearly goes to the Slope Control Algorithm.

4.3 PRACTICAL CONSIDERATIONS

When implementing the Slope Control Algorithm the main source of added complexity arises from the necessity of computing δM_F any time a node is extended. This operation can be easily implemented using a set of pointers that link any stored node to its predecessor in the tree. These pointers called "Pathpointers" are always incorporated in the data structure of the stack [3]. Although quite simple in principle, this procedure quickly becomes very much time consuming as the window length F increases, since the backtracking in the tree can be performed only one node at a time.

In the software implementation of the algorithm, we departed from this somewhat simplistic approach by using a cyclic memory structure, where the metric values of extended nodes are retained for the eventual purpose of computing δM_F. This memory may be represented by a shift register, as illustrated on Figure 20 The cumulated metric of the newly extended node is entered while, at the other end, the data concerning its Fth predecessor on the path is lost. Note that the address of the node in the stack may be used in place of its metric value. This may present significant advantages if the same hardware structure is to be used with different metric tables. The stack structure remains roughly unchanged. However it should be noted that the shift register has to be reinitialized with the proper metric values or addresses each time the decoder leaves the currently extended path and starts its search on a new one.,

In order to overcome this difficulty, one could retain for each stored node the relevant information concerning its F predecessors, which is the state of the shift register at the time the node was stored. This principle is illustrated on Figure 21 This approach, albeit certainly faster, would significantly increase the memory requirements. However when considering Figure 21 observe that:

1. 2 nodes stemming from the same predecessor can share the same memory positions, and
2. the number of required memory positions is bound to be limited since the decoder will never backsearch in the tree beyond a certain depth [21].

Thus, although very attractive in principle, the SCA faces severe implementation problems.

5. V. CONCLUSIONS

In this paper Sequential Decoding used in conjunction with retransmission procedures has been examined. The Stack Algorithm has been used with a rate 1/2 code over a hard quantized discrete memoryless channel operating at R/Rcomp equal to Ø.99. The retransmissions are all initiated by the computational effort of the decoder.

Two types of selective repeat retransmission procedures have been developed and their performance studied. The impact of these algorithms on the decoder stack storage and the overall system performance (system throughput) has been analyzed and compared to computer simulation results. The trade offs between the system throughput, the waiting line at the input buffer and the decoder speed factor have been examined.

For the very noisy channel considered, the Stack Overflow Algorithm offer very good throughput and a reduction of the required memory storage. The Anticipation Algorithm is especially attractive for its excellent performance and ease of implementation. The Slope Control Algorithm allows a smaller waiting line as a function of the throughput than the Stack Overflow Algorithms, but these advantages are obtained at a cost of substantial increase of complexity. All the results show that sequential decoding with retransmission procedures allow very robust performances and hence may become a very attractive alternative for Hybrid FEC/ARQ over very noisy channels.

REFERENCES

1. Fano,R.M., "A Heuristic Discussion on Probabilistic Decoding", IEEE Transactions on Information Theory,Vol.IT-9,pp.64-73,avril 1963.

2. Jelinek,F., "Fast Sequential Decoding Using a Stack", IBM Journal of Research and Development,Vol.13,pp.675-685,novembre 1969.

3. Bhargava,V.K,Haccoun,D.,Matyas,R.,Nuspl,P., "Digital communication by Satellite", John Wiley & Sons,Inc.,1981.

4. Haccoun,D., Ferguson,M.J., "Generalized Stack Algorithms for decoding Convolutional Codes", IEEE trans. on Information Theory, Vol. IT-21, Nov. 1975.

5. Chevillat, P., Costello, D., "A Multiple Stack Decoder for erasure-free decoding of Convolutional Codes", IEEE Trans. on Com. Vol. COM-25, Dec. 1977.

6. Drukarev,A.,et Costello,D.J.,(1983) "Hybrid ARQ Error Control using Sequential Decoding",IEEE Transactions on Information Theory, Vol.IT-29,pp.521-535,juillet 1983.

7. P.Y. Pau and D. Haccoun. "Sequential decoding with ARQ". IEEE Symposium on Information Theory. St Jovite, Quebec, September 1983.

8. Sacham,N., "Performance of ARQ with Sequential Decoding over One-Hop and Two-Hop Radio Links",IEEE Transactions on Communications,Vol.COM-31,pp.1172-1180,octobre 1983.

9. Dixon R. Doll "Data Communications" John Wiley, New York, 1978.

10. Johannesson,R., "Robustly Optimal Rate One-Half Binary Convolutionnal Codes",IEEE Transactions on Information Theory,juillet 1979.

11. Jacobs,I.M.,et Berlekamp,E.R., "A Lower-Bound to the Distribution of Computations for Sequential Decoding",IEEE Transactions on Information Theory,Vol.IT-13,pp.167-174,avril 1967.

12. Kleinrock,L., "Queuing Systems", John Wiley & Sons,Inc.,1975.

13. Savage,J.E., "The Distribution of the Sequential Decoding Computation Time",IEEE Transactions on Information Theory, Vol.IT-12,pp.143-147,avril 1966.

14. Haccoun,D.,Dufour,M., "Stack and input buffers overflow of Stack decoding algorithms",IEEE International Symposium on Information Theory,Los Angeles,p. 55,f{vrier 1981.

15. Jelinek,F., "An Upper-Bound on Moments of Sequential Decoding Effort",IEEE Transactions on Information Theory,Vol.IT-15,pp.140-149, janvier 1969.

16. Geist, J., "Algorithmic aspects of sequential decoding", Ph.D dissertation, University of Notre-Dame, August 1970.

17. R.E. Kahn, S.A. Gronemeyer, J. Burchfield, and R.C. Kunzelman, "Advances in packet radio technology", Proceedings of the IEEE, Vol. 66, pp. 1468-1496, Nov 1978.

18. P.Y. Pau, "Décodage séquentiel avec retransmissions", Master's Thesis, École Polytechnique de Montréal, May 1984.

19. Viterbi,A.J.,et Omura,J.K., "Principles of Digital Communication and Coding", McGraw-Hill,Inc.,1979.

20. Morris,J.M., "Optimal Block-Length for ARQ Error Control Schemes",IEEE Transactions on Communications,Vol.COM-27, pp.488-493,f{vrier 1979.

21. D. Haccoun, "Multiple-Path Stack-Algorithms for decoding Convolutional Codes", Ph.D. Dissertation, McGill University, Oct. 1974.

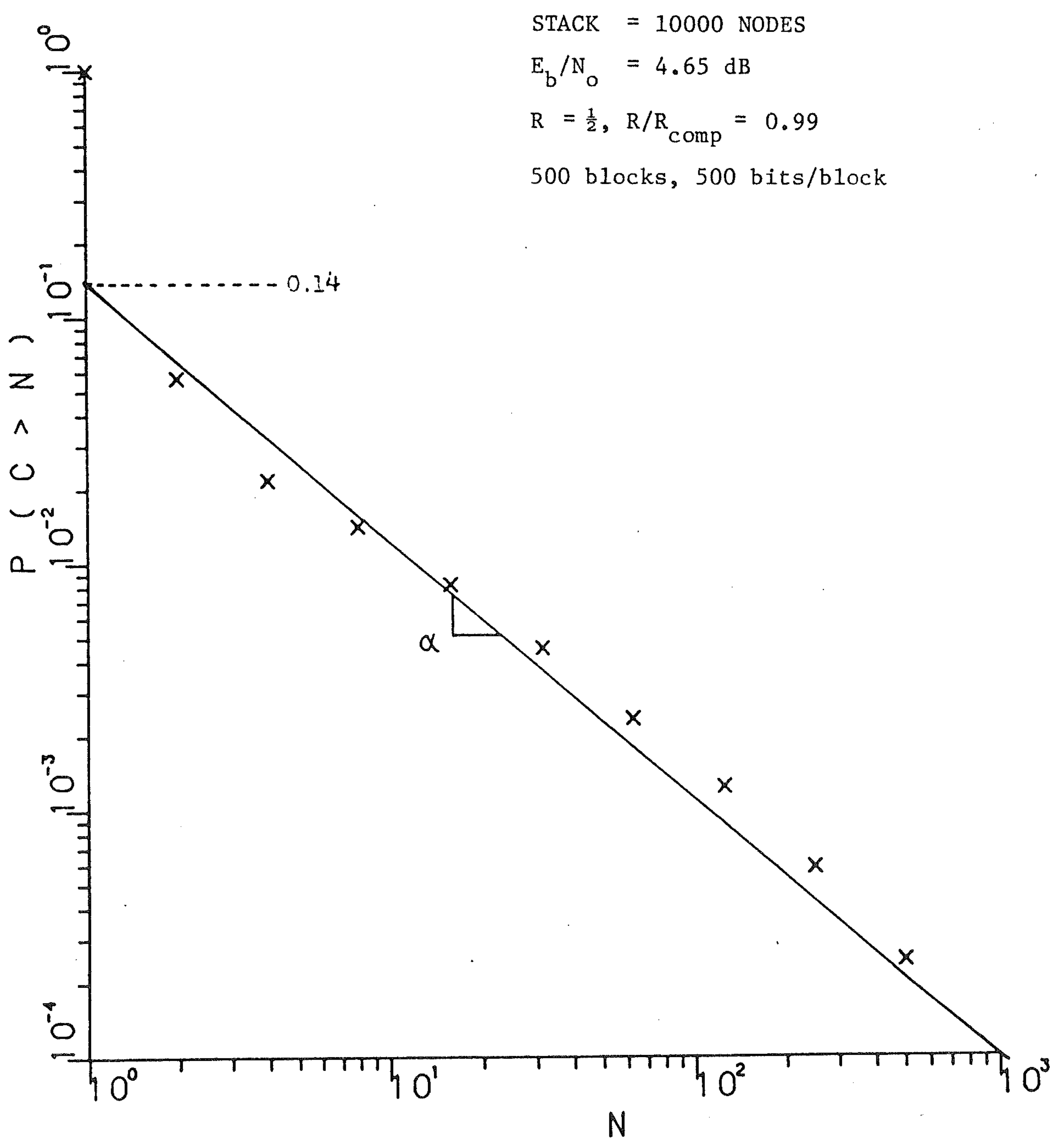


Figure 1: Observed distribution of number of computations to recode one bit and Pareto approximation

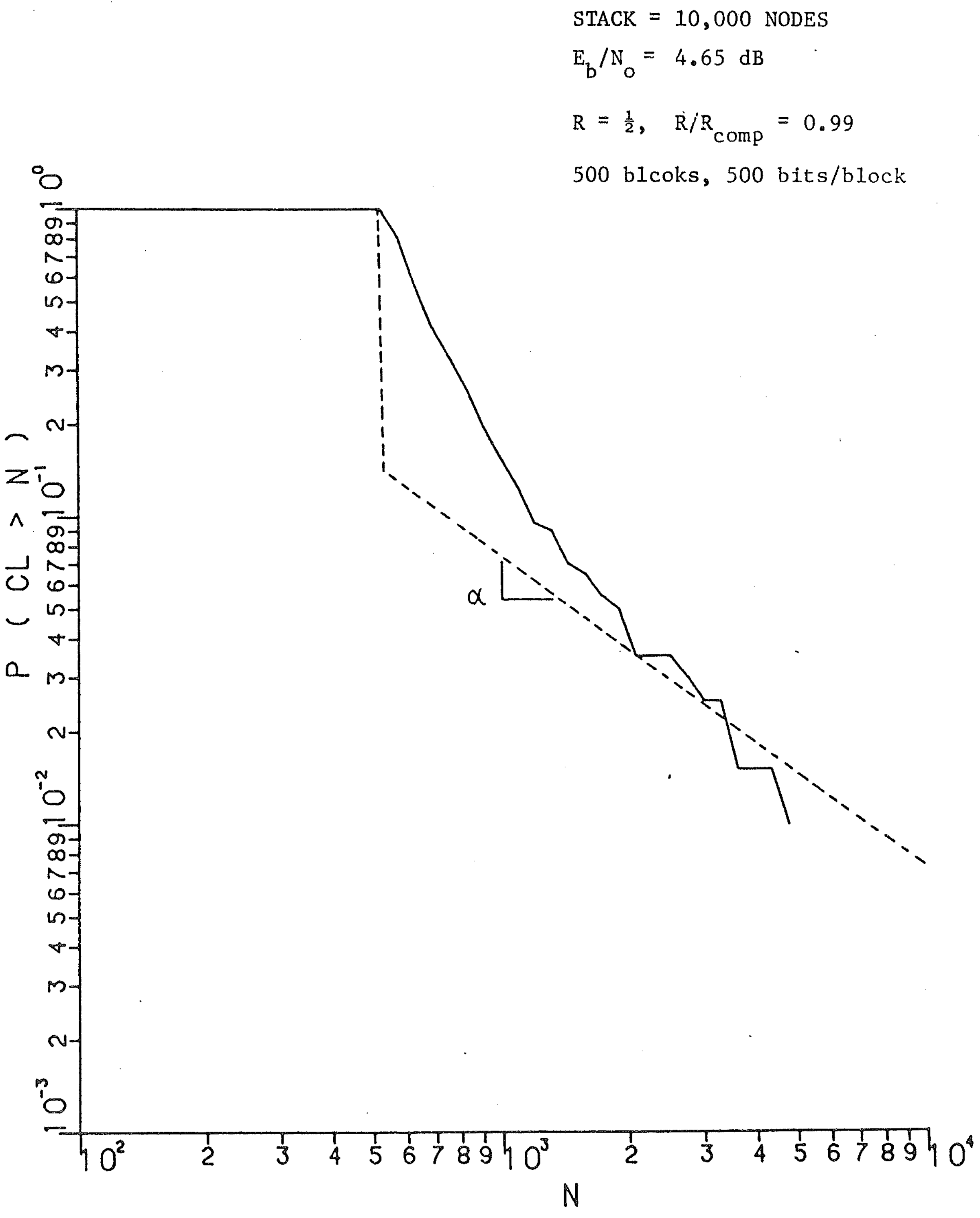


Figure 2: Observed distribution of the number of computations to decode one block and Pareto approximation

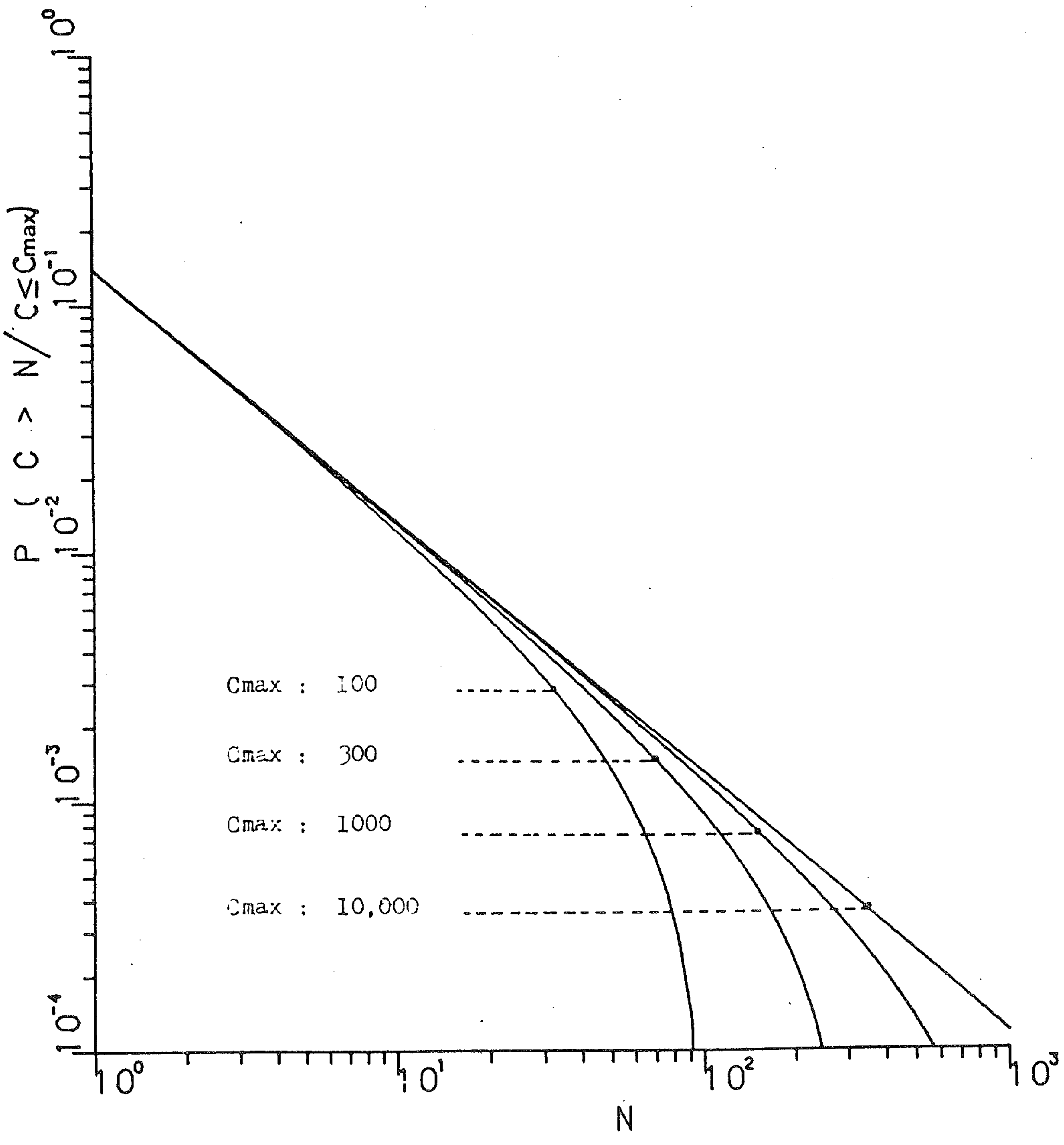


Figure 3: Theoretical distribution of the number of computations to decode one bit. Effect of C_{max}.

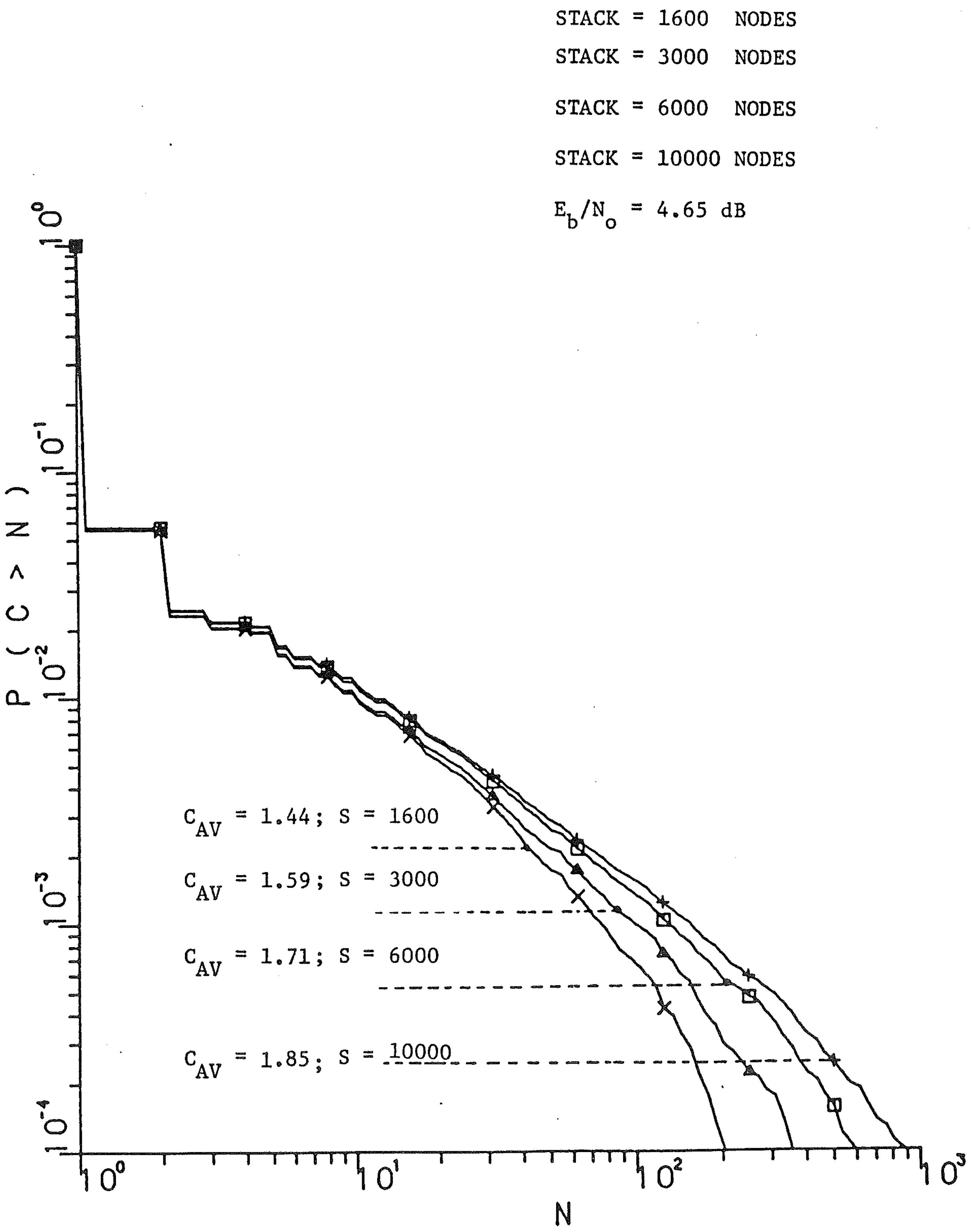


Figure 4: Distribution of the number of computations to decode one bit. Influence of the stack size for the Simple Stack Overflow Algorithm.

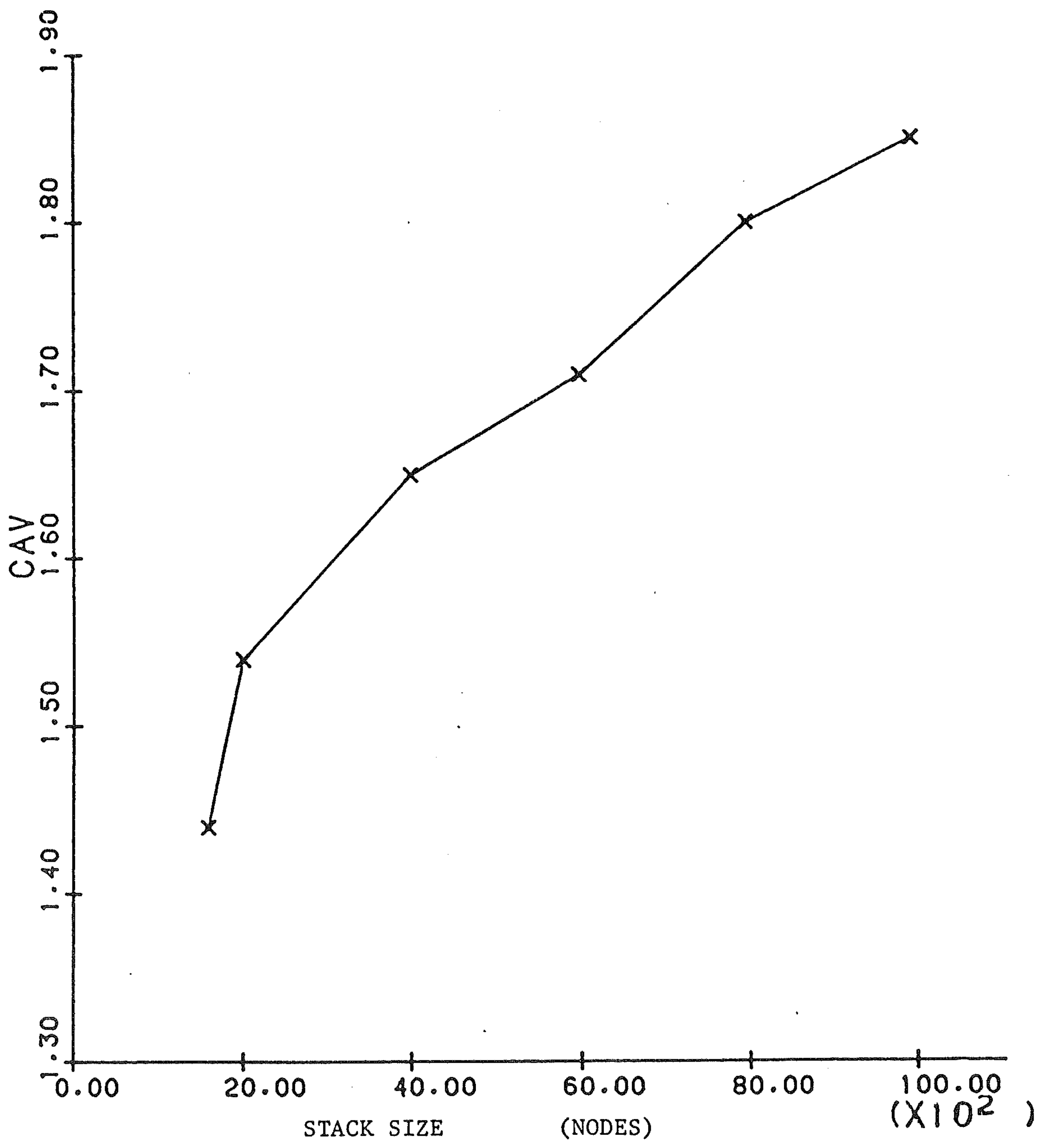


Figure 5: Average decoding effort as a function of the stack size, Simple Stack Overflow Algorithm (VARIANT 1)

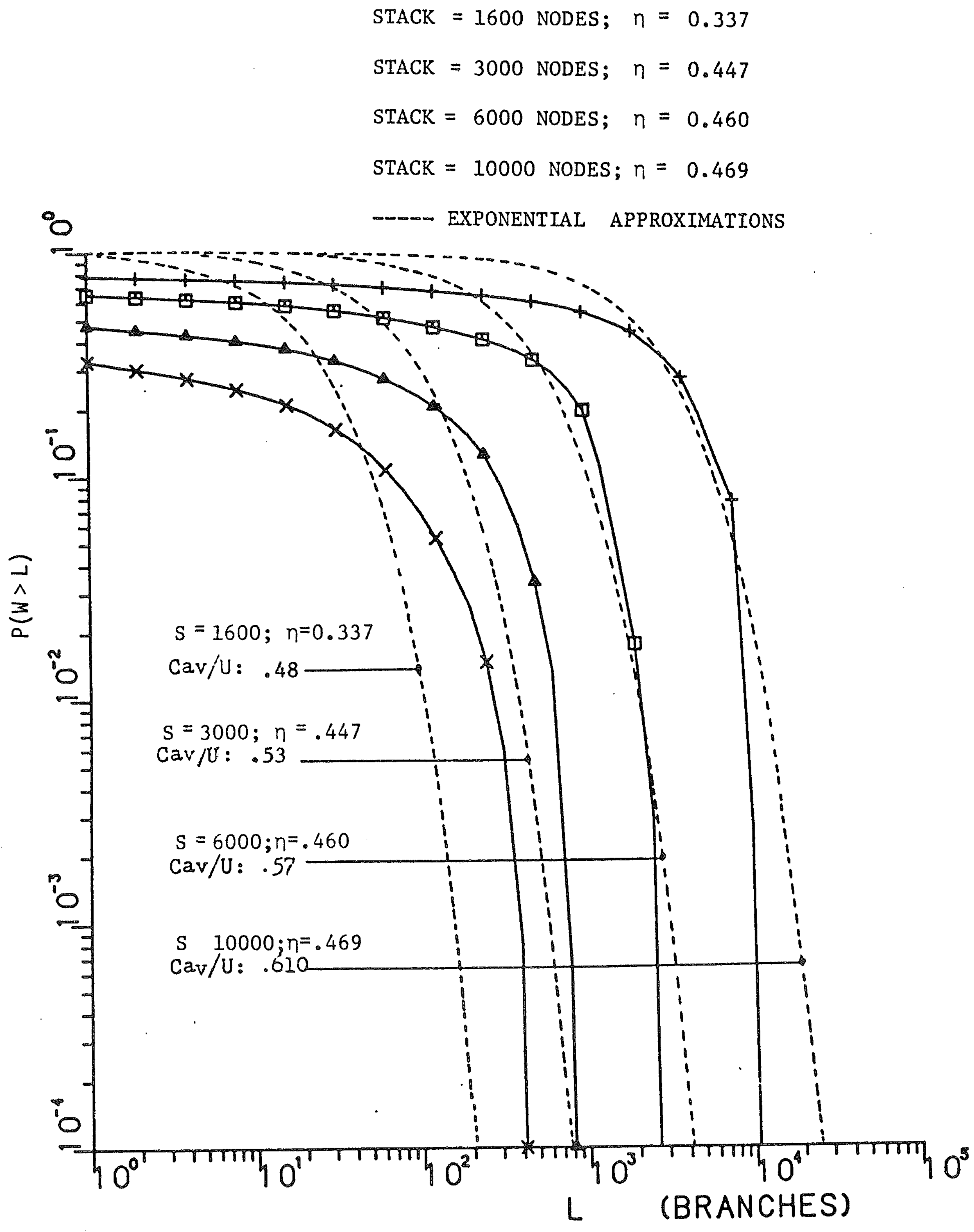


Figure 6: Waiting line distribution for VARIANT 1 with decoder speed factor U = 3.

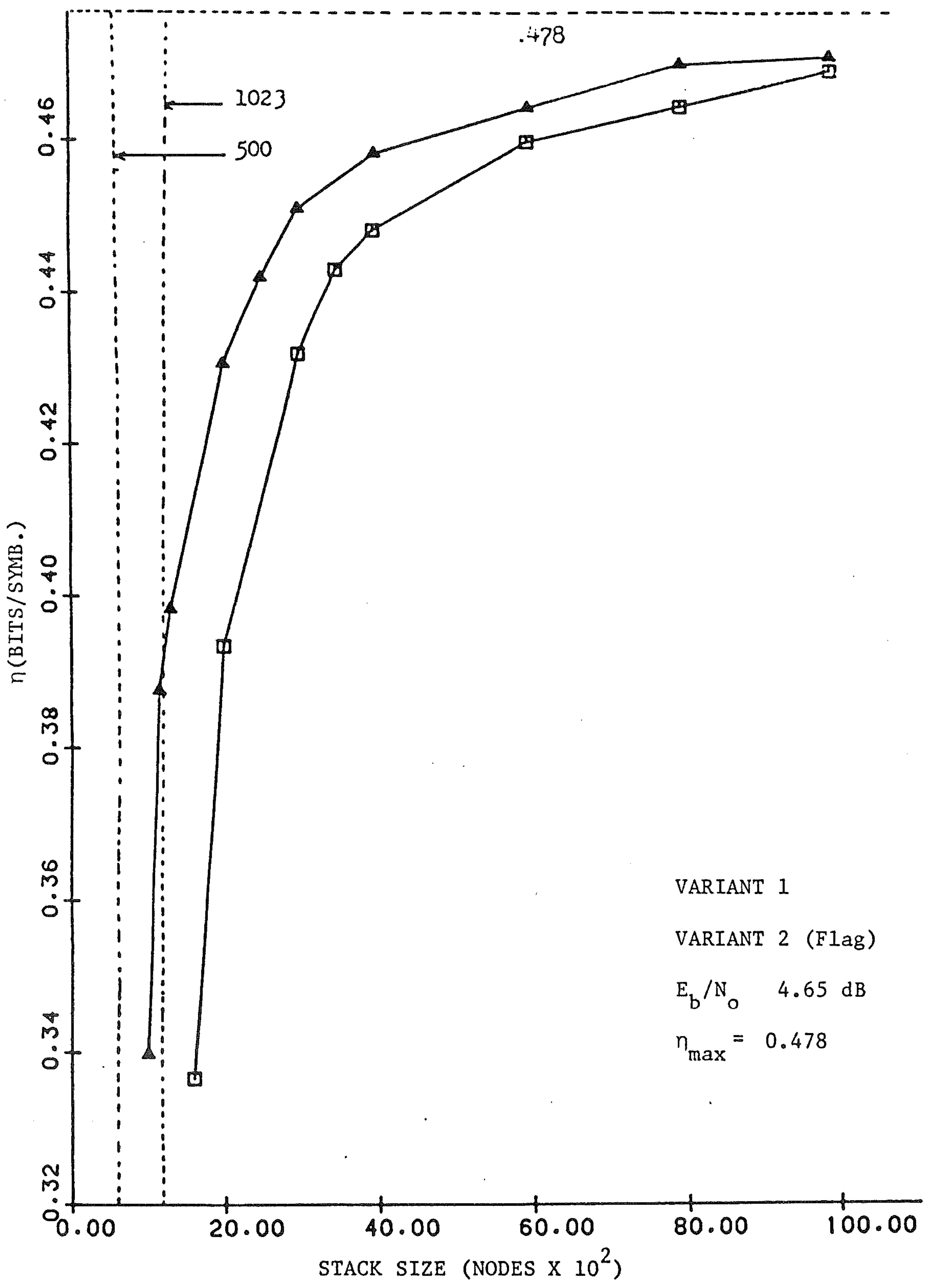


Figure 7: Throughput as a function of the stack size for Variant 1 and Variant 2 (Flag)

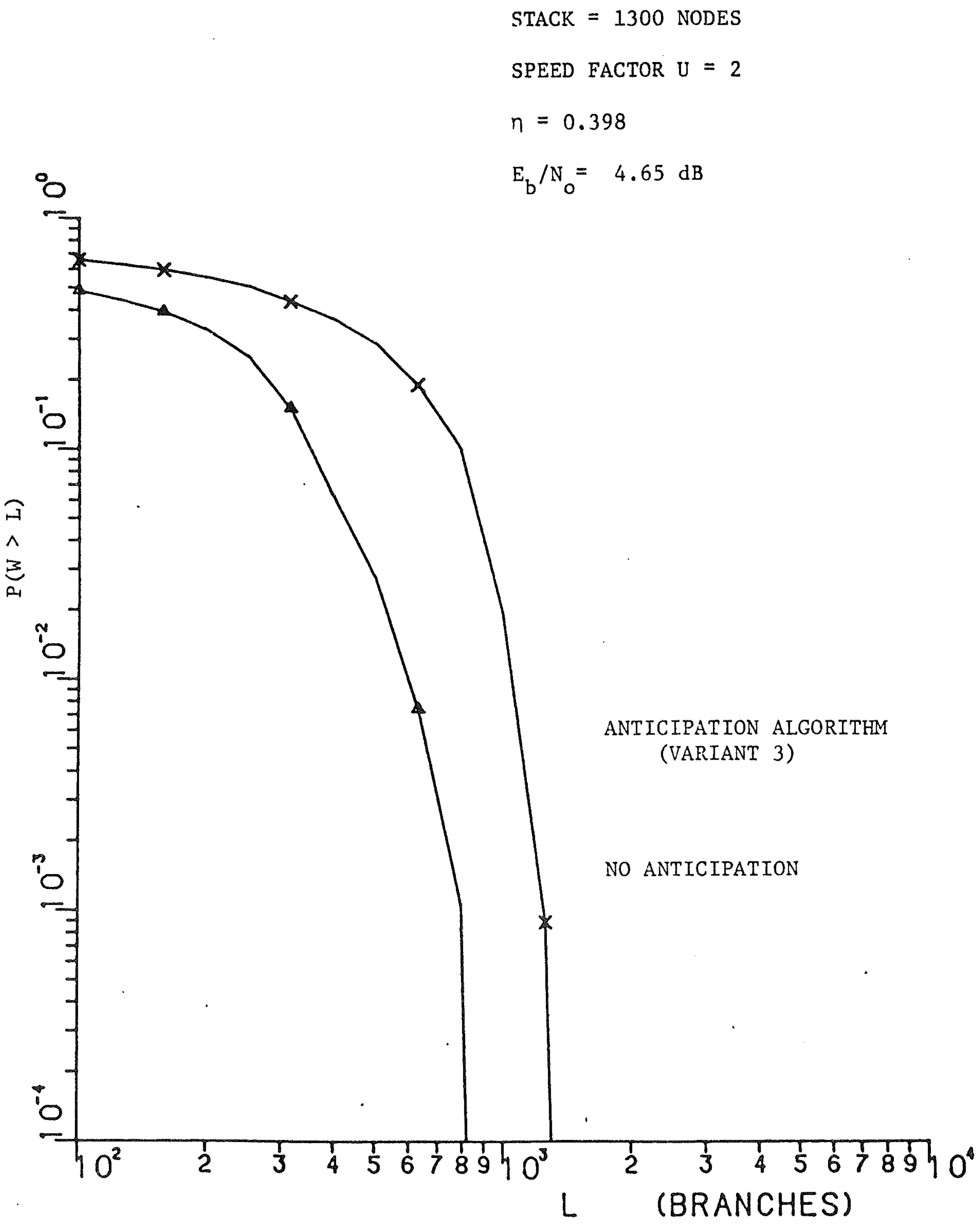


Figure 8: Waiting line distribution with and without anticipation at constant η.

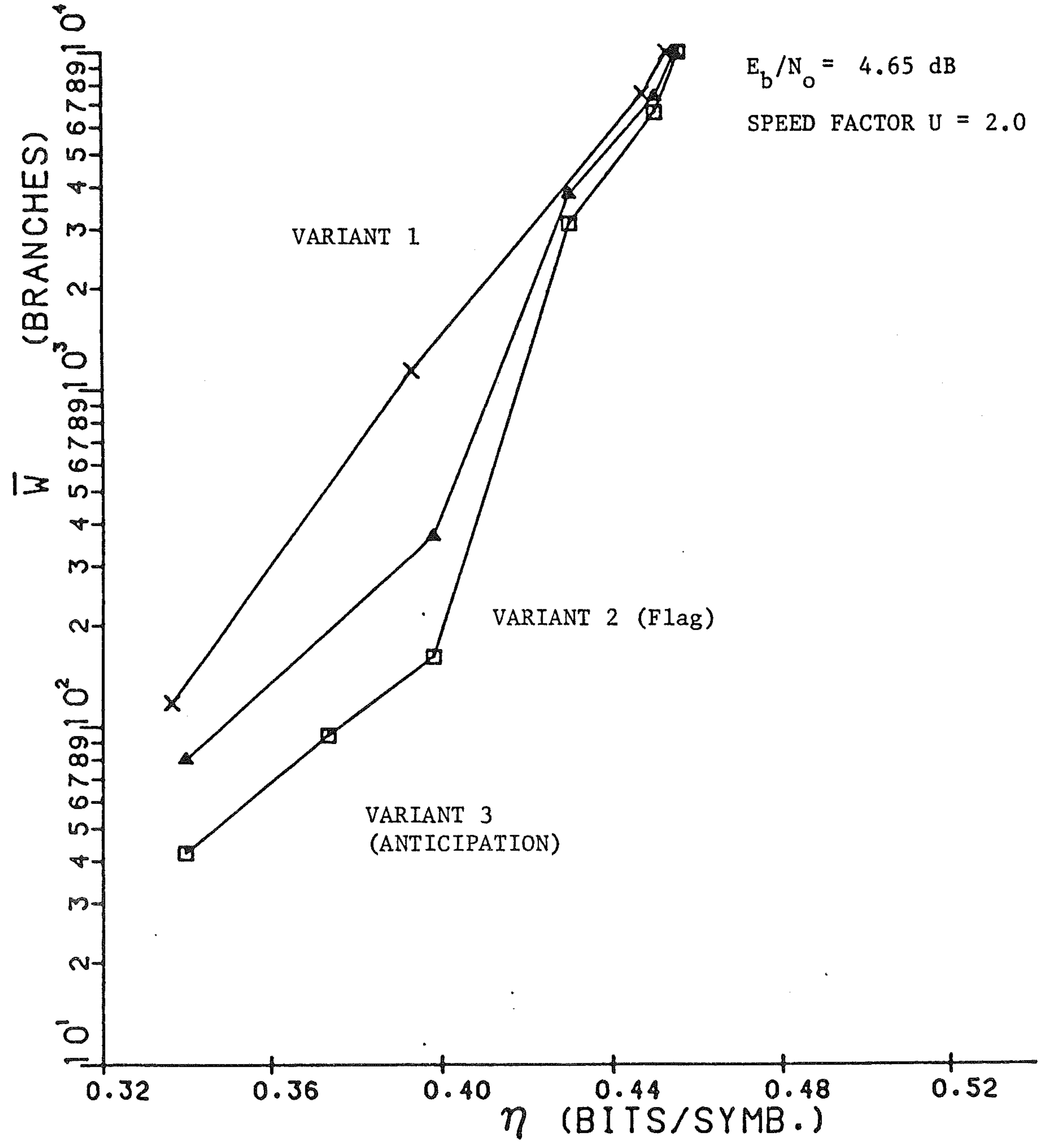


Figure 9: Average waiting line $\overline{W}$ as a function of the throughput for variants 1,2, and 3.

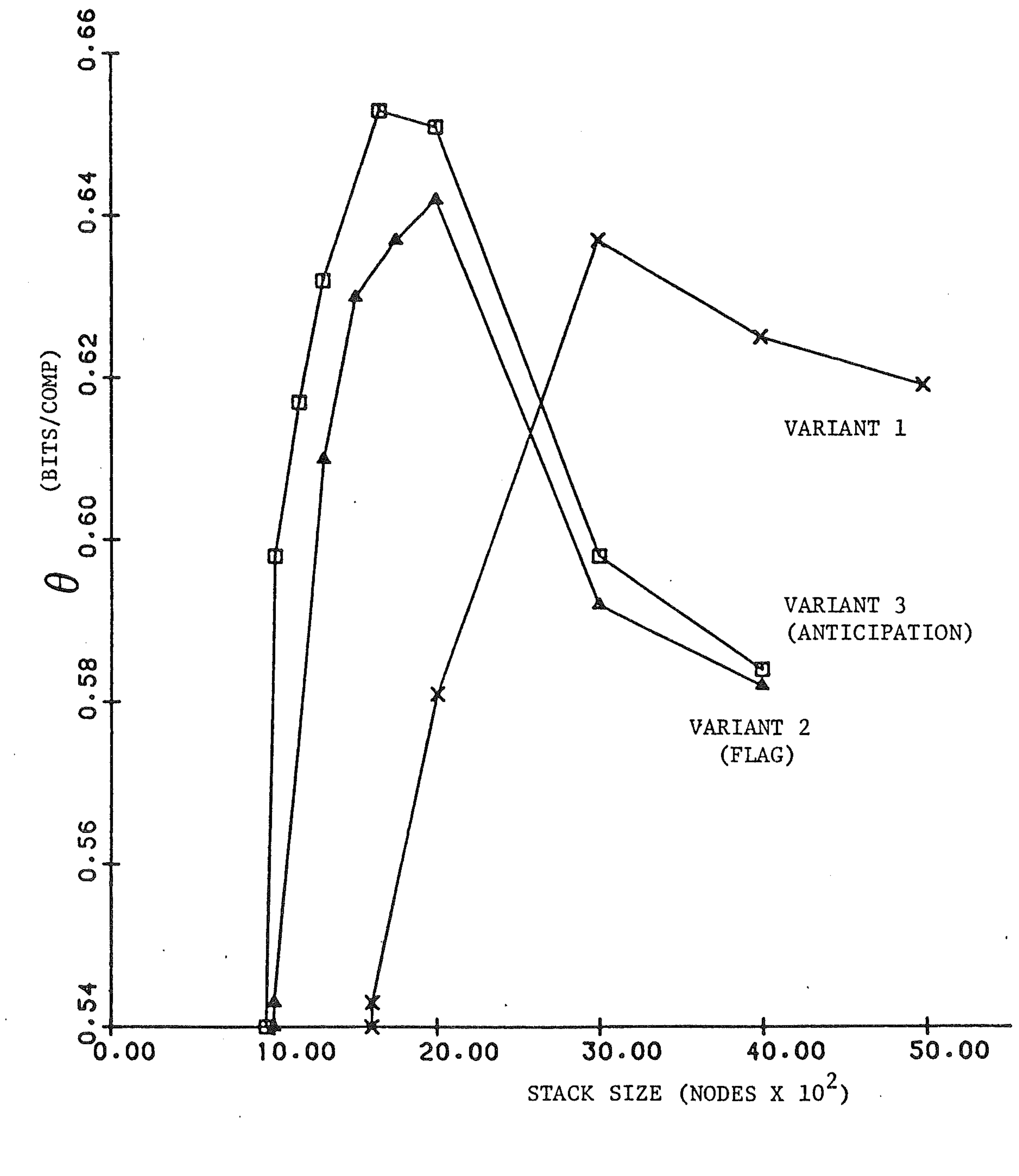


Figure 10: Stack size optimization for Variants 1,2, and 3.

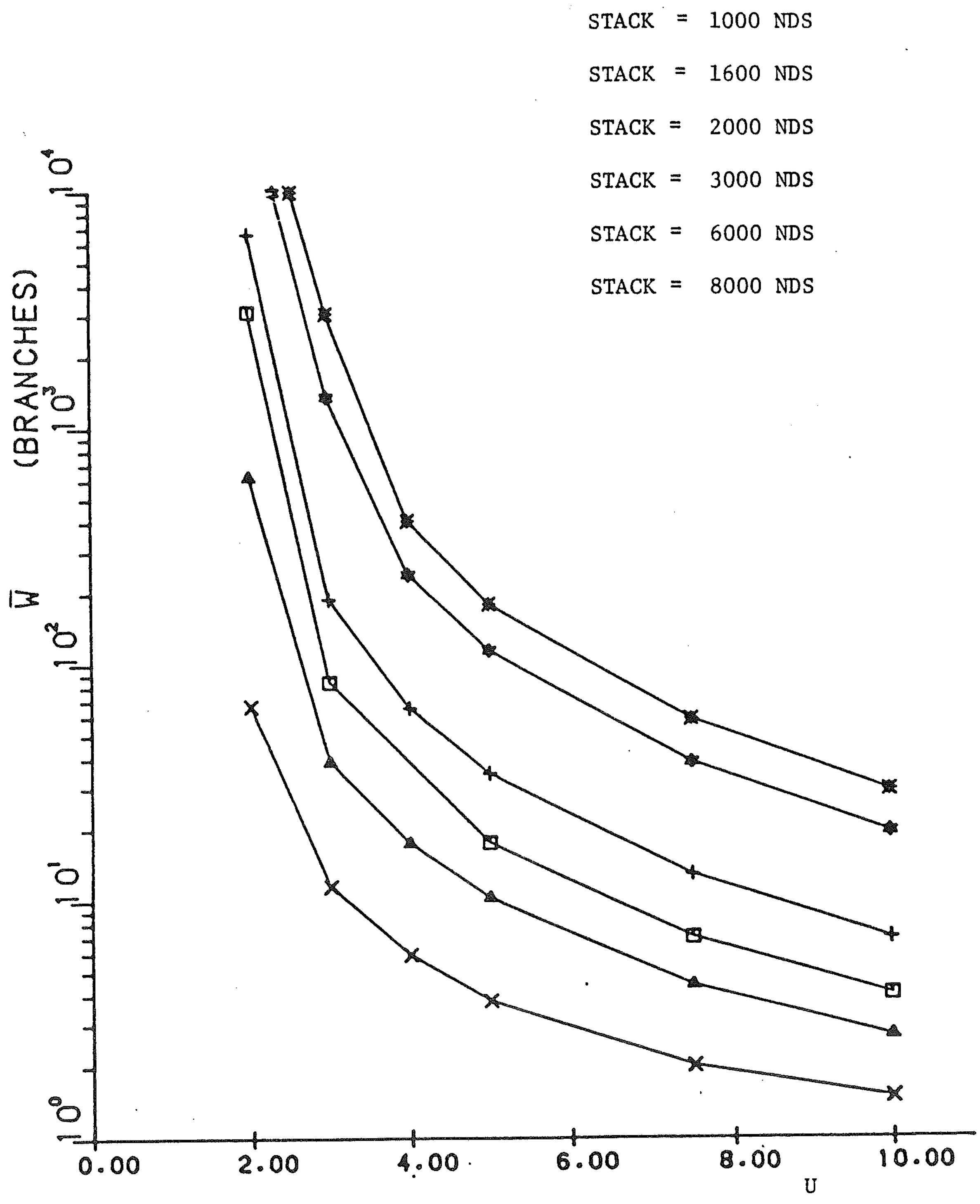


Figure 11: Waiting line as a function of the speed factor U for different stack sizes. VARIANT 3.

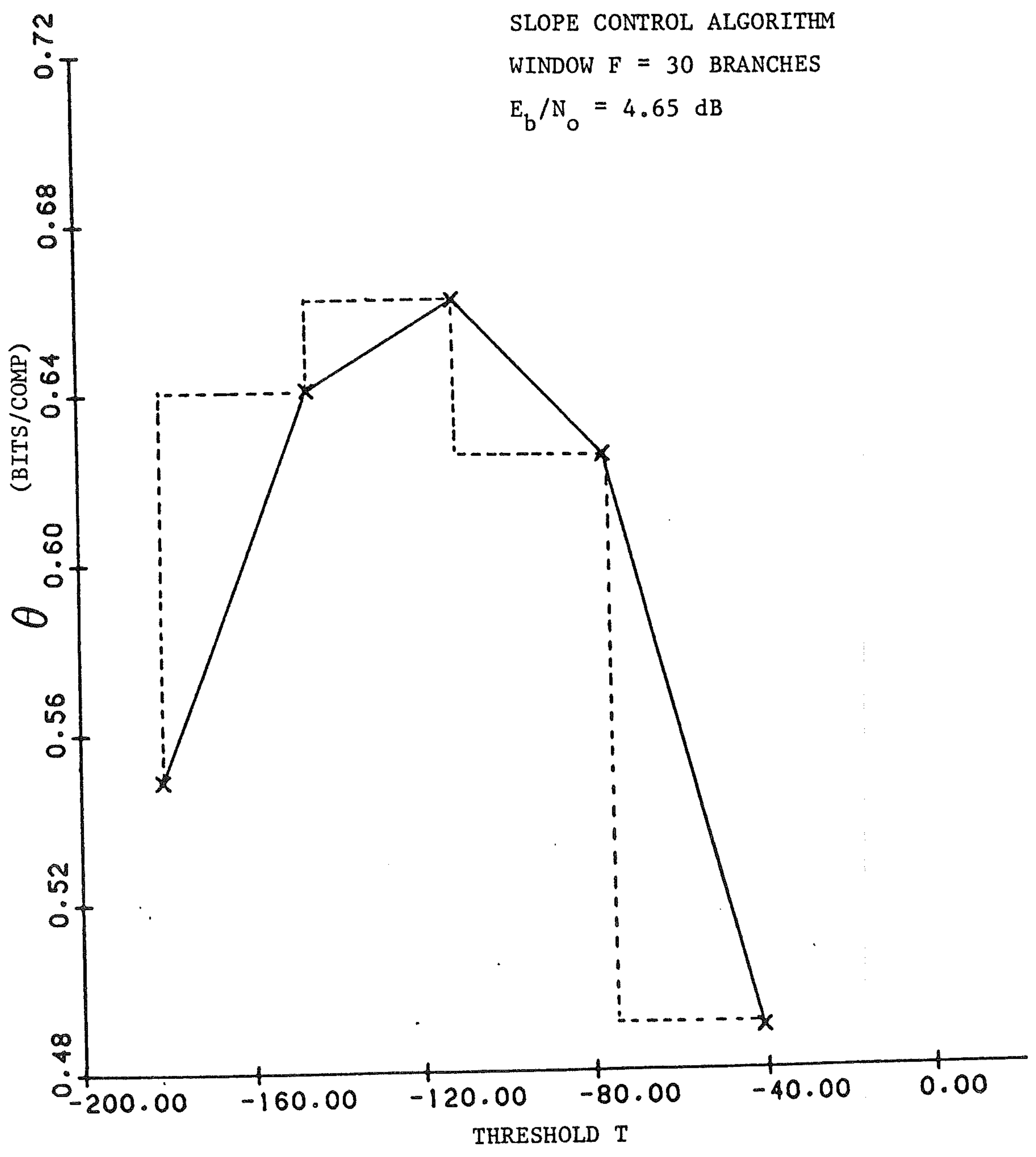


Figure 12: Observed system TRIB as a function of the threshold. Slope Control Algorithm with 30-branch window.

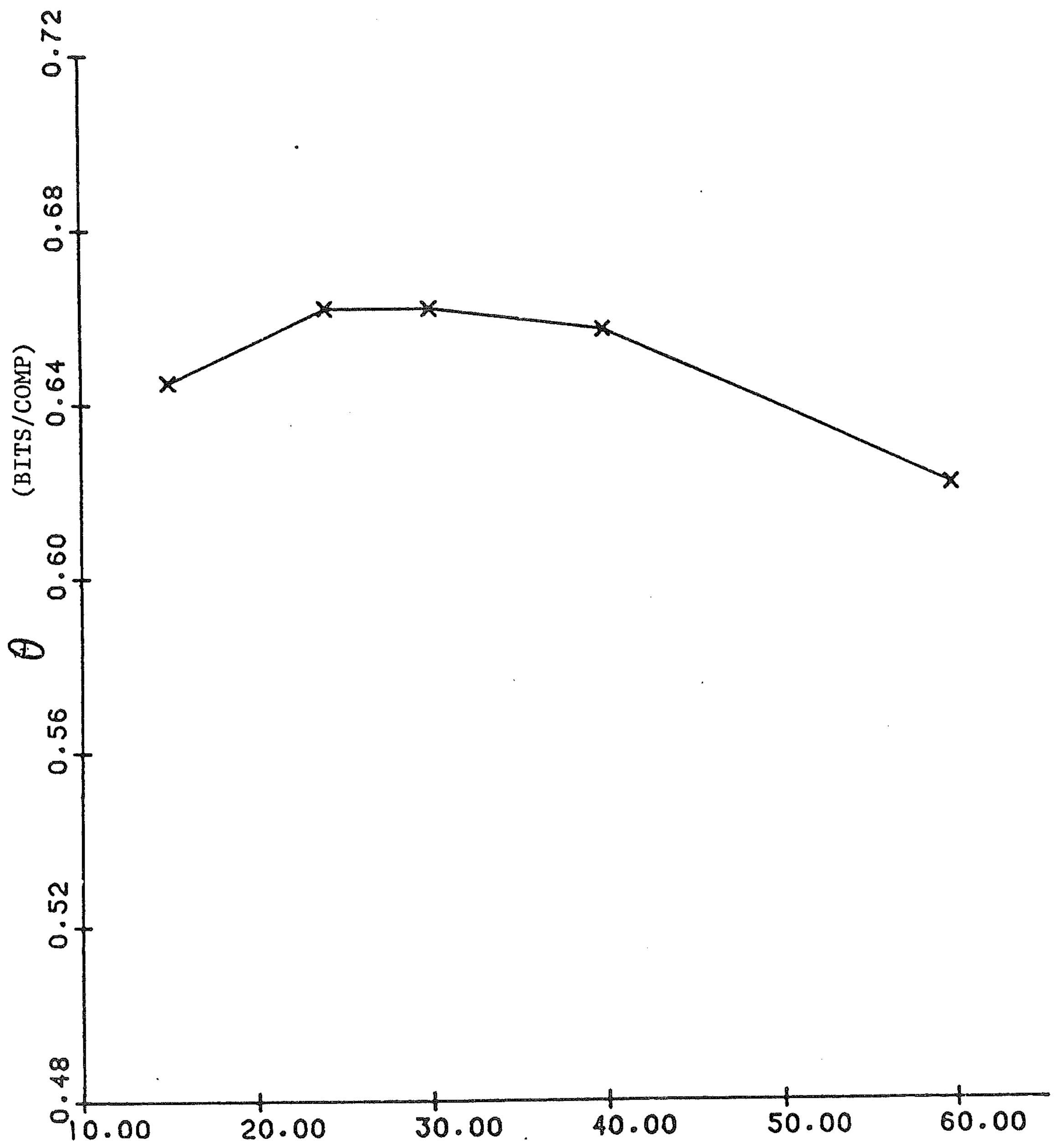


Figure 13: Window optimization. Slope Control Algorithms, E_b/N_o = 4.65 dB.

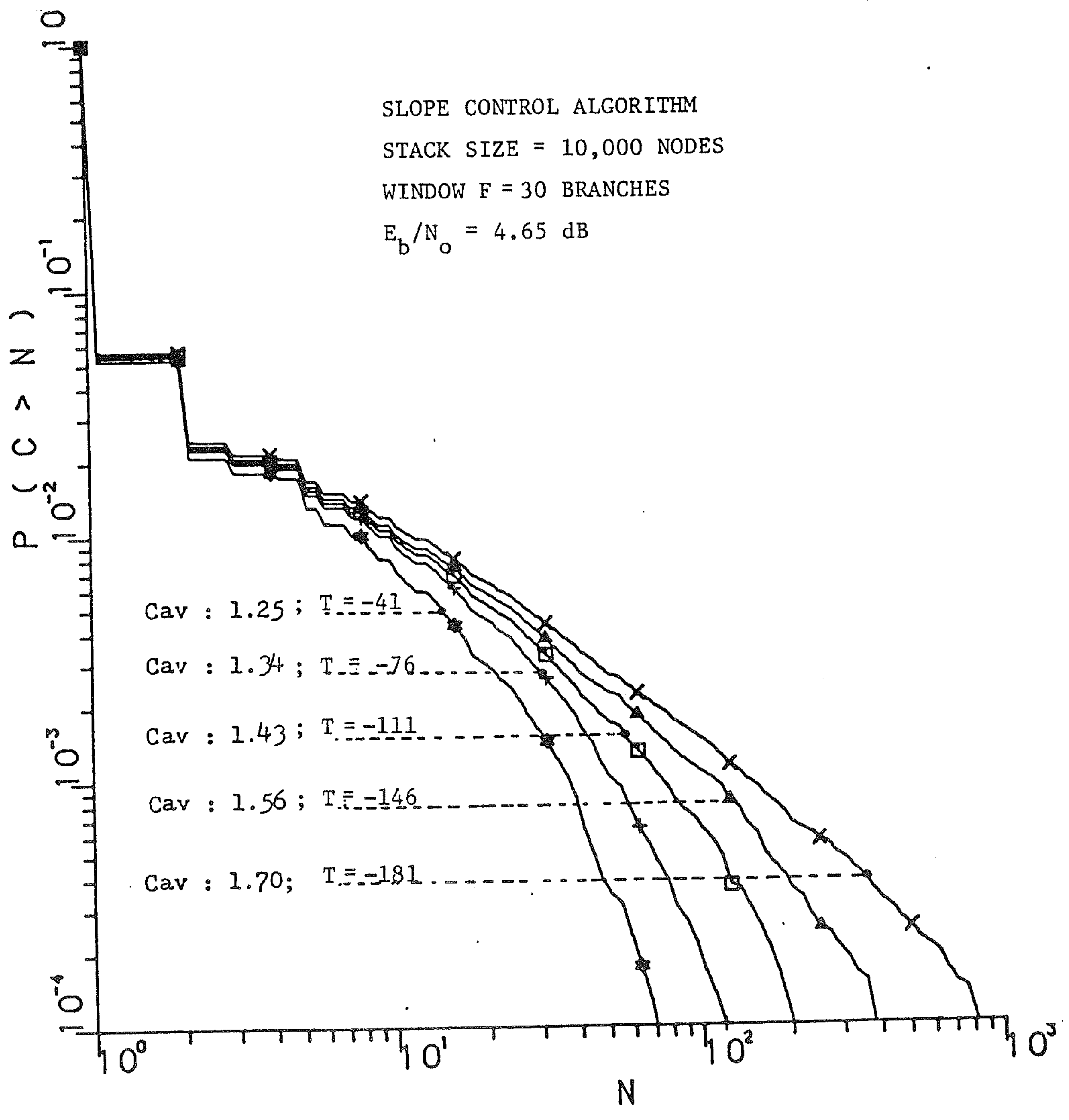


Figure 14: Observed computational distributions for Slope Control Algorithm with different threshold values.

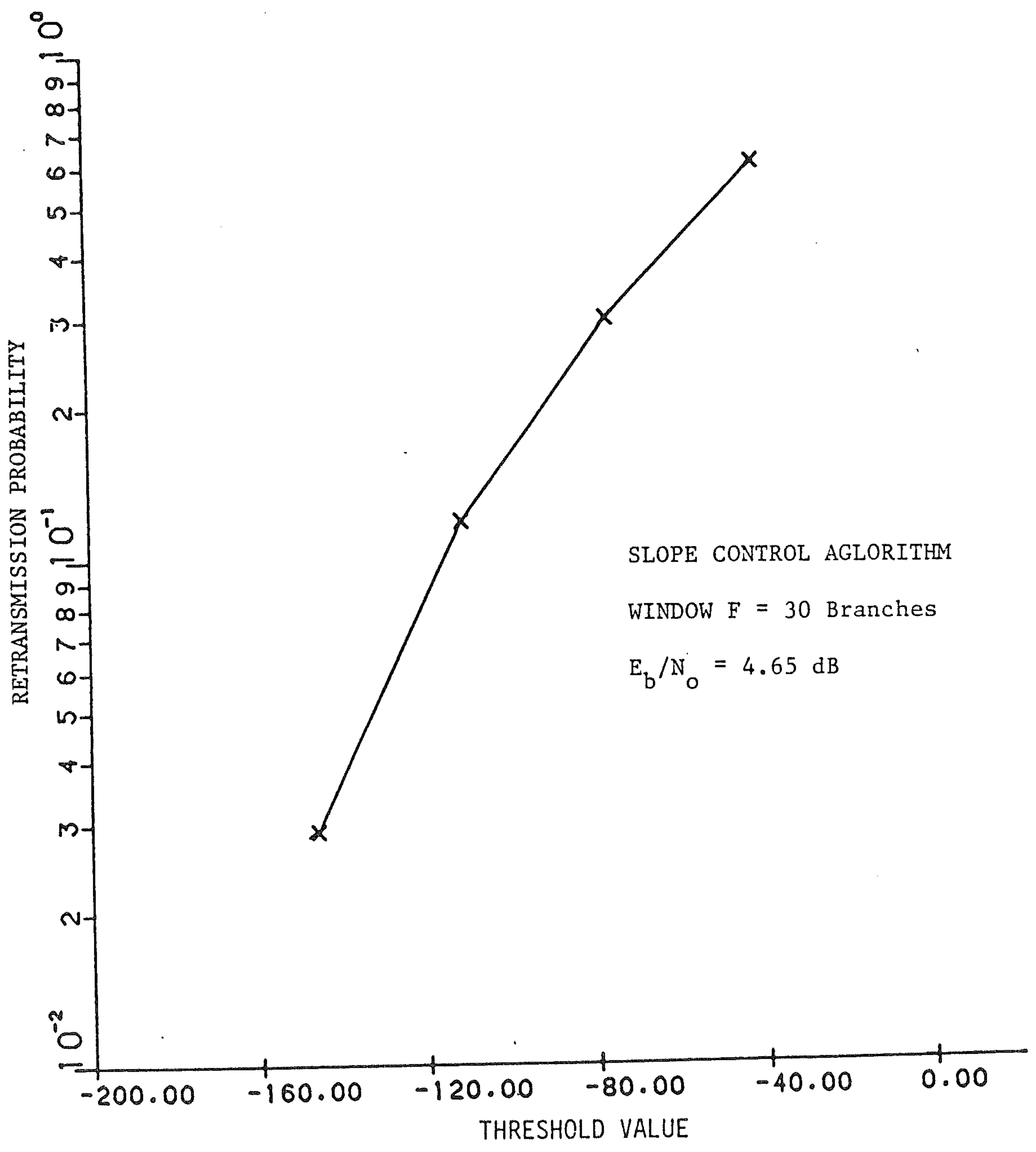


Figure 15: Retransmission probability as a function of the threshold value.

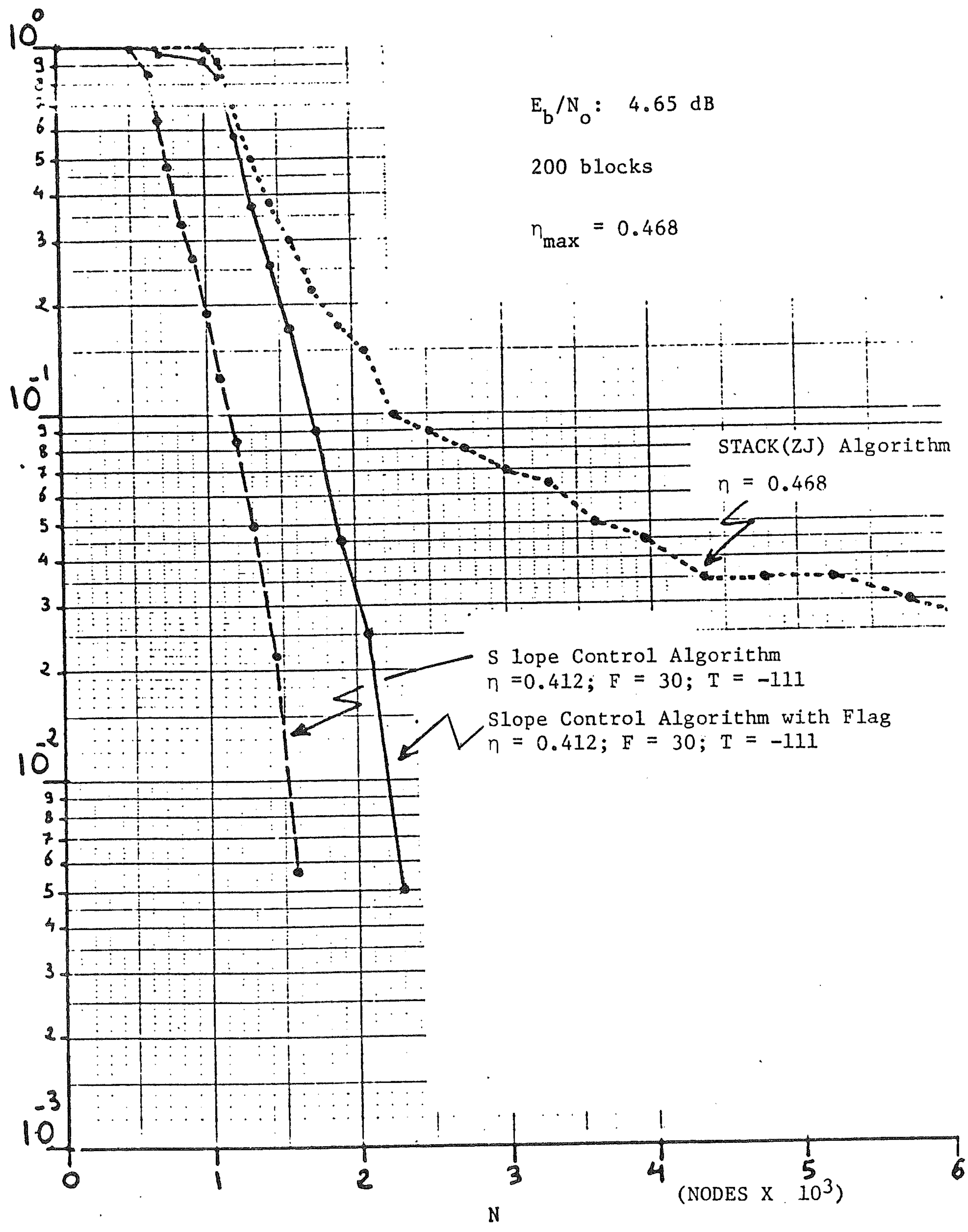


Figure 16: Distributions of number of entries in stack per decodent block.

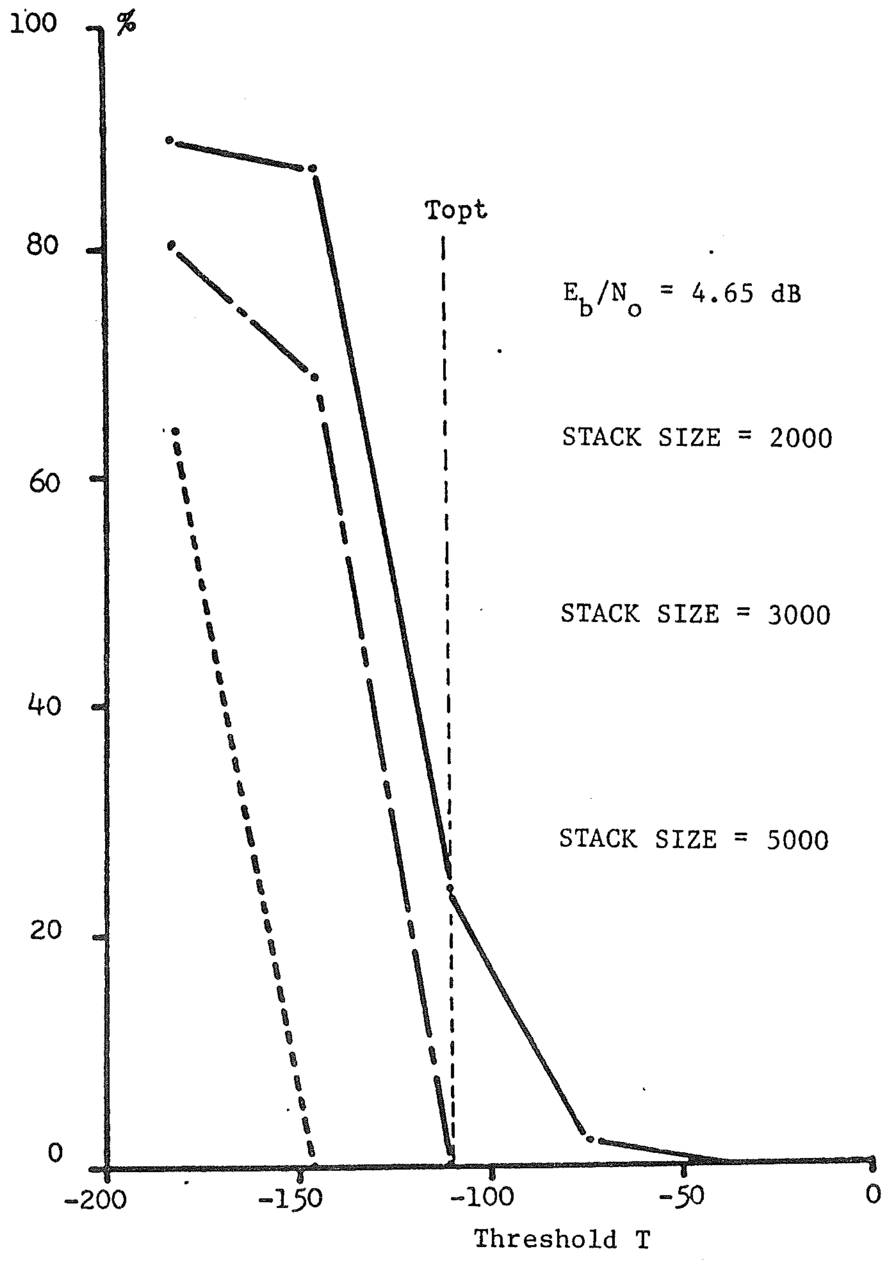


Figure 17: Percentage of ARQ's due to stack overflows. Slope Control Algorithm with window F = 30 Br.

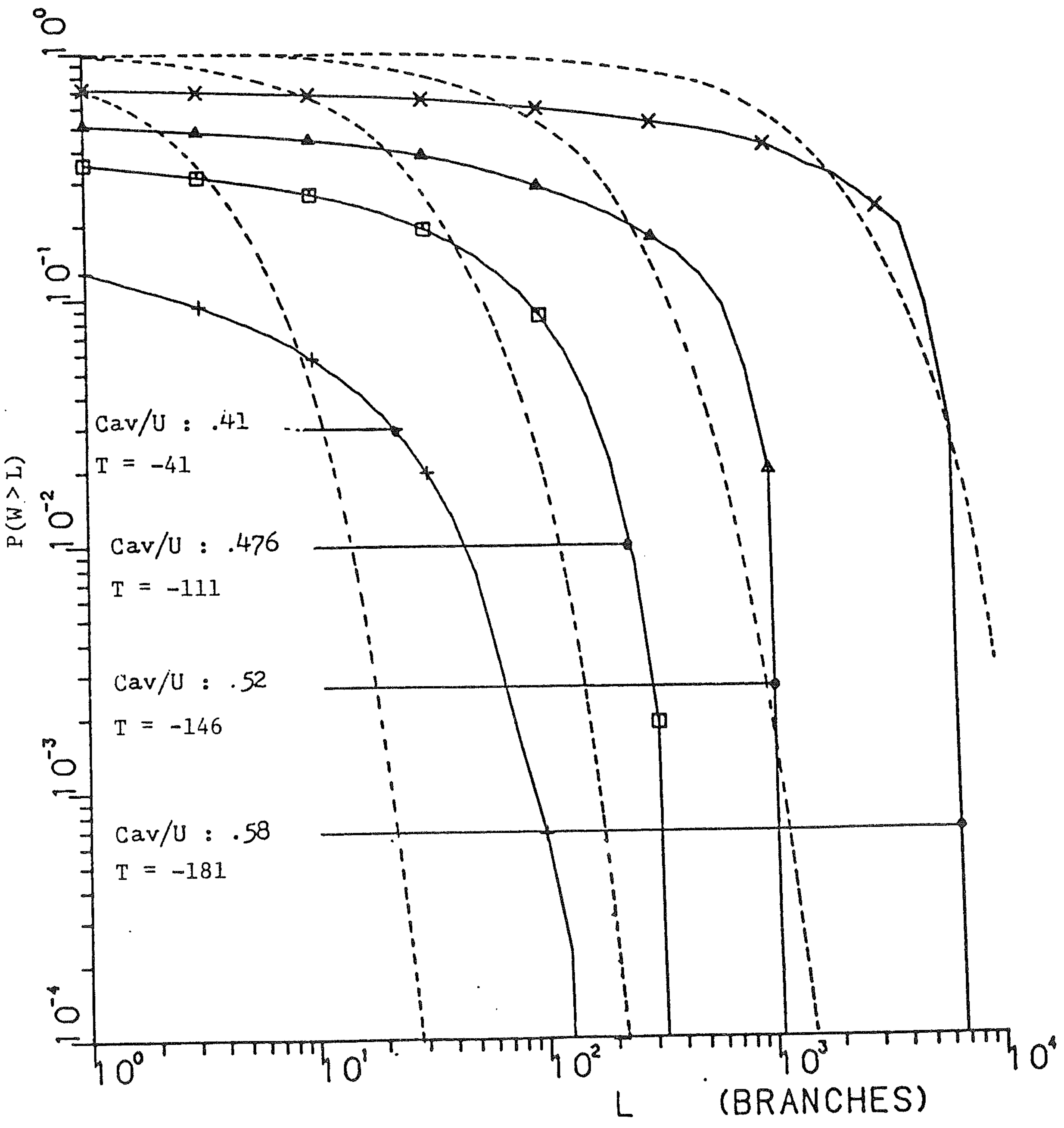


Figure 18: Waiting Line Distributions for several threshold values. Slope Control Algorithm with F = 30, S = 10,000; U = 3.

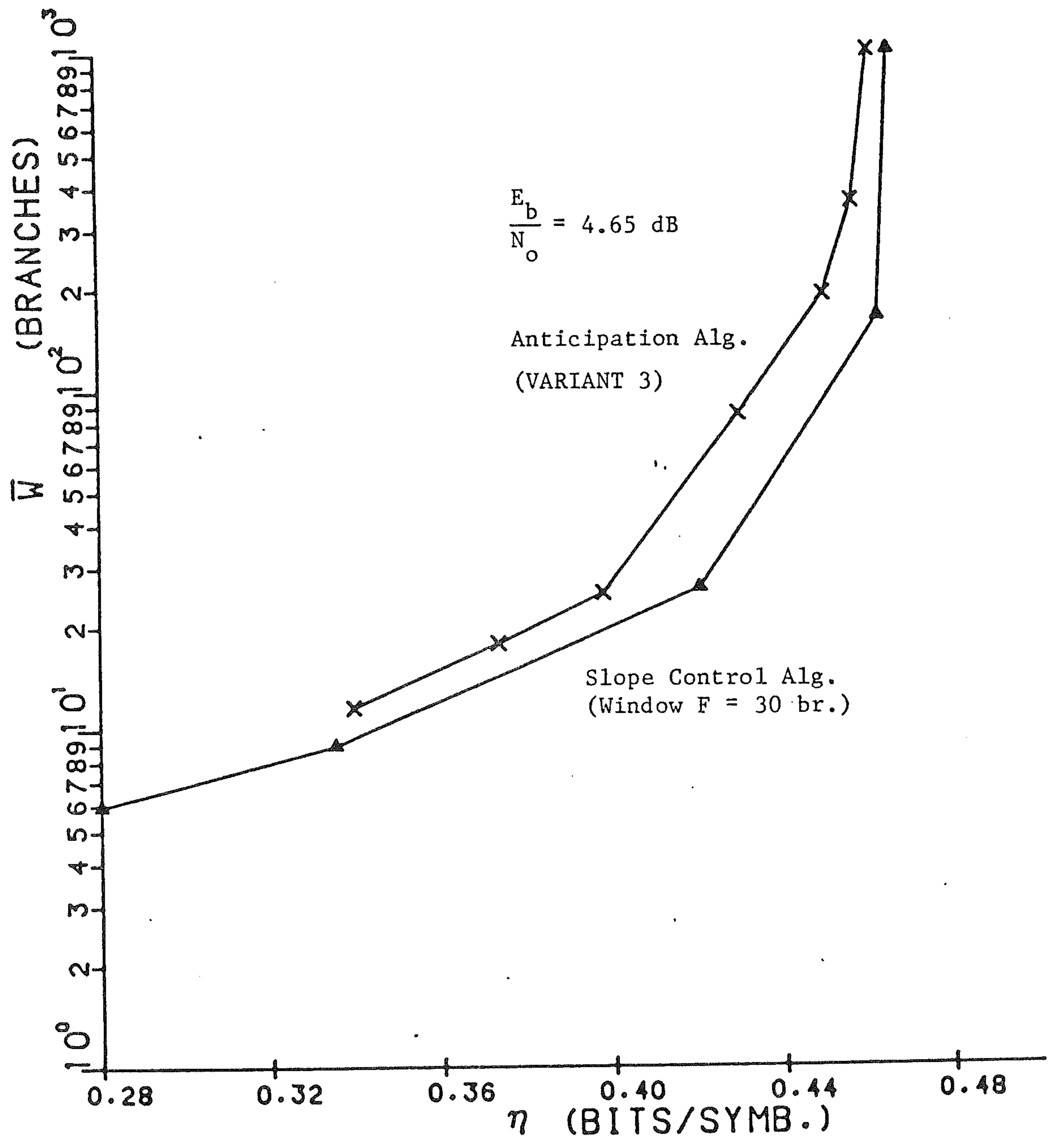


Figure 19: Performance comparison between Slope Control and Anticipation Algorithms for Speed Factor U = 3.

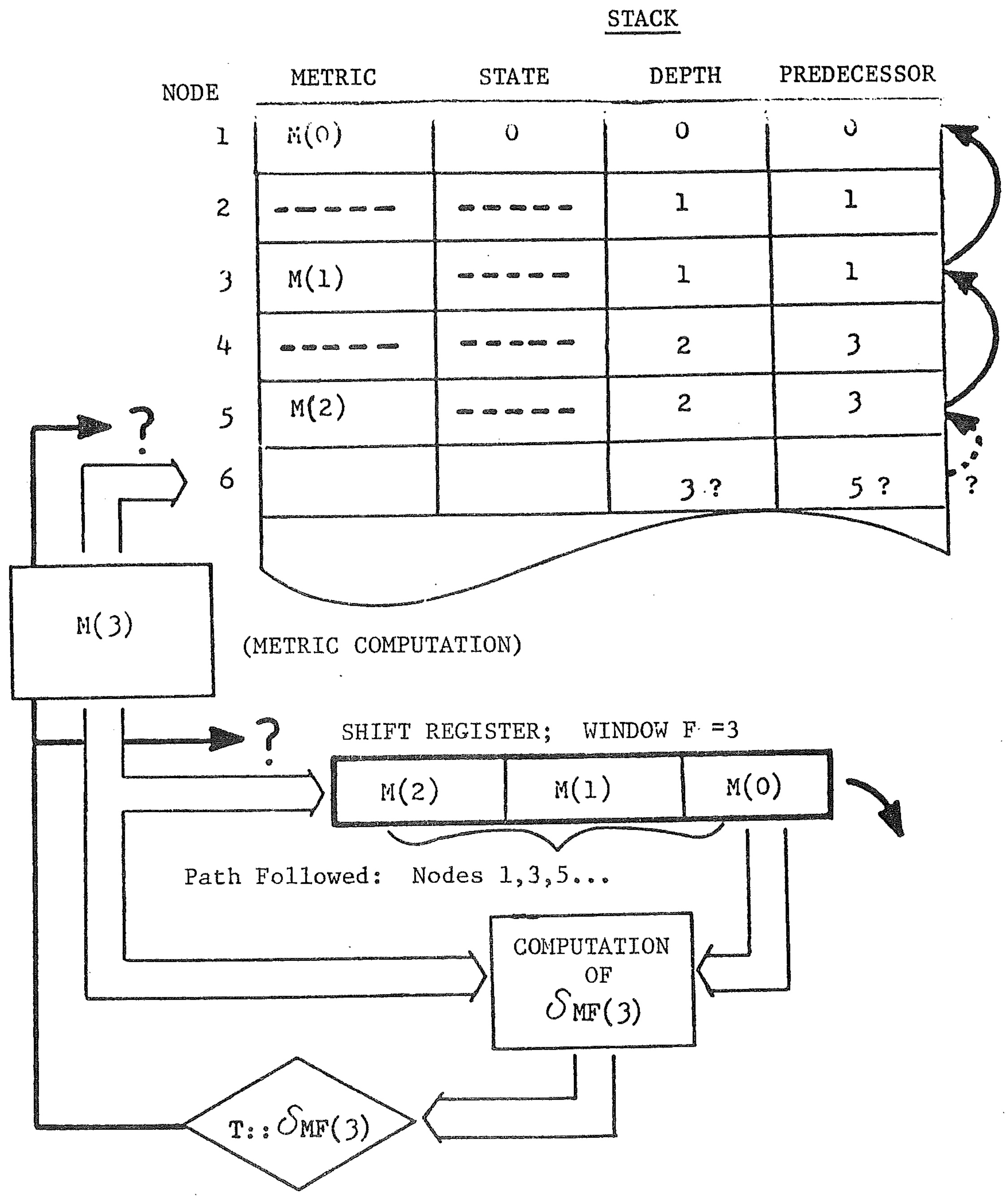
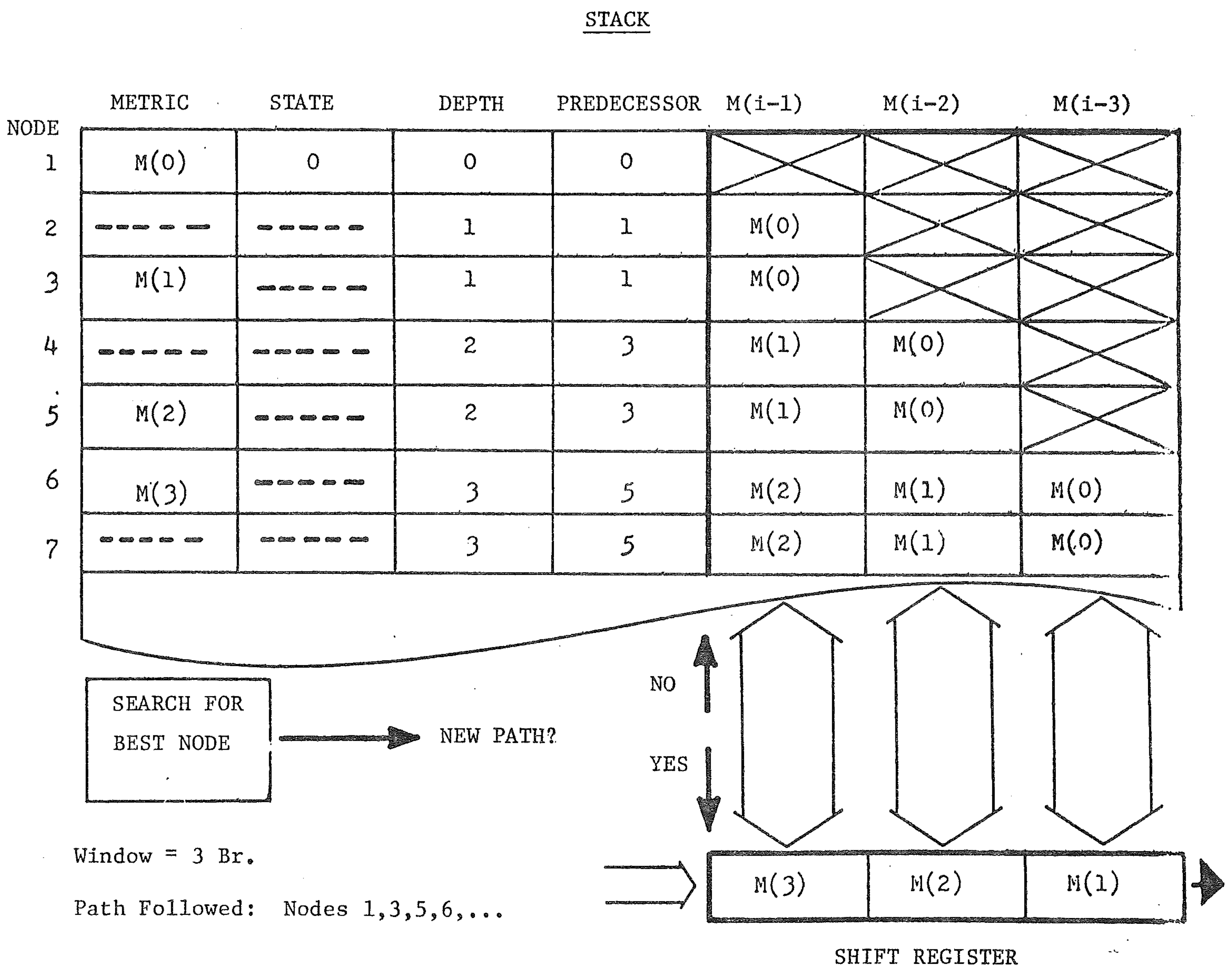


Figure 20: Implementation of Slope Control Algorithm; utilization of an F-shift register



NODE	METRIC	STATE	DEPTH	PREDECESSOR	M(i-1)	M(i-2)	M(i-3)
1	M(0)	0	0	0			
2	------	------	1	1	M(0)		
3	M(1)	------	1	1	M(0)		
4	------	------	2	3	M(1)	M(0)	
5	M(2)	------	2	3	M(1)	M(0)	
6	M(3)	------	3	5	M(2)	M(1)	M(0)
7	------	------	3	5	M(2)	M(1)	M(0)

M(3)	M(2)	M(1)

Window = 3 Br.

Path Followed: Nodes 1,3,5,6,...

Figure 21: Implementation of slope control algorithm; storage of explored paths.