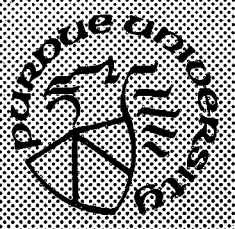
5-1-1990

Importance Sampling Simulation of the Stack Algorithm with Application to Sequential Decoding

Khaled Ben Letaief
Purdue University



Importance Sampling Simulation of the Stack Algorithm with Application to Sequential Decoding

Khaled Ben Letaief

TR-EE 90-37
May 1990

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

IMPORTANCE SAMPLING SIMULATION OF THE STACK ALGORITHM WITH APPLICATION TO SEQUENTIAL DECODING

A Thesis

Submitted to the Faculty

of

Purdue University

by

Khaled Ben Letaief

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 1990

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor Professor John Sadowsky for providing the guidance that made this work possible. I thank him for all his support, financial and otherwise. I would also like to thank Professors Edward Delp, Saul Gelfand, and Steve Lalley for serving on my advisory committee. I acknowledge their help and suggestions. Special thanks goes to Professor Delp for all the help he provided me during graduate school.

My wife, Selma, has been a constant support to me throughout my undergraduate and graduate schools. I cannot sufficiently thank her for her support and understanding.

Special thanks to all the members of my family for their moral support.

Finally, I would like to thank all the friends that I have had the opportunity to meet at Purdue University.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

Table Page

Table Page

LIST OF FIGURES

Figure Page

Figure Page

ABSTRACT

Ben Letaief, Khaled. Ph.D., Purdue University. May 1990. Importance Sampling Simulation of the Stack Algorithm with Application to Sequential Decoding. Major Professor: J. S. Sadowsky.

Importance sampling is a Monte Carlo variance reduction technique which in many applications has resulted in a significant reduction in computational cost required to obtain accurate Monte Carlo estimates. The basic idea is to generate the random inputs using a biased simulation distribution. That is, one that differs from the true underlying probability model. Simulation data is then weighted by an appropriate likelihood ratio in order to obtain an unbiased estimate of the desired parameter.

This thesis presents new importance sampling techniques for the simulation of systems that employ the *stack algorithm*. The stack algorithm is primarily used in digital communications to decode convolutional codes, but there are also other applications. For example, sequential edge linking is a method of finding edges in images that employs the stack algorithm. In brief, the stack algorithm is an algorithm that attempts to find the maximum metric path through a large decision tree. There are two quantities that characterize its performance. First there is the probability of a branching error. The second quantity is the *distribution of computation*. It turns out that the number of tree nodes examined in order to make a specific branching decision is a random variable. The distribution of computation is the distribution of this random variable. The estimation of the distribution of computation, and parameters derived from this distribution, is the main goal of this work.

We present two new importance sampling schemes (including some variations) for estimating the distribution of computation of the stack algorithm. The first general method is called the *reference path method.* This method biases noise inputs using the weight distribution of the associated convolutional code. The second method is the *partitioning method.* This method uses a stationary biasing of noise inputs that alters the drift of the node metric process in an ensemble average sense. The biasing is applied only up to a certain point in time; the point where the correct path node metric minimum occurs. This method is inspired by both information theory and large deviations theory.

This thesis also presents another two importance sampling techniques. The first is called the *error events simulation method.* This scheme will be used to estimate the error probabilities of stack algorithm decoders. The second method that we shall present is a new importance sampling technique for simulating the sequential edge linking algorithm. The main goal of this presentation will be the development of the basic theory that is relevant to this simulation problem, and to discuss some of the key issues that are related to the sequential edge linking simulation.

CHAPTER 1

INTRODUCTION

It is often the case that the complexity of the problems associated with many of todays communications systems discourages analytical solutions. Furthermore, even though there is a vast body of knowledge about these systems. Nontheless, when it comes down to evaluation of system performance, the present state of the art often requires some idealistic assumptions. As a result, it has long been recognized that *Monte Carlo* simulation is effectively the only practical way to evaluate the performance of many of todays communications systems. A typical example is the calculation of error probabilities for digital communications systems. If the parameters of interest are events of rare probabilities, then simulating such events using a brute force Monte Carlo approach might be intractable, if not impractical. The reason being that one has to generate a very large number L of independent random samples in order to obtain good estimates of the probability of these rare events. To appreciate the magnitude of this problem, suppose that we wish to estimate an error probability P_e. To evaluate P_e based on Monte Carlo simulation techniques, we estimate P_e using the maximum likelihood estimator

$$\hat{P}_e = \frac{1}{L} \sum_{\ell=1}^{L} I_e(X^{(\ell)})$$

where $I_e(.)$ denotes the indicator random variable of the error event and $X^{(1)}, X^{(2)}, ..., X^{(L)}$ are independent random samples which are identically distributed. If

P_e is small, then we would not expect to "hit" the error event very often during the L simulations. This then would require that L must be very large to insure that $\hat{P}_e$ is close to P_e with high probability. In fact, it is easy to show that for any $\varepsilon > 0$ (Chebychev inequality)

$$\mathcal{P}(|\hat{P}_e - P_e| \geq \varepsilon) \leq \frac{\mathrm{var}[\hat{P}_e]}{\varepsilon^2}$$

$$= \frac{\mathrm{var}[I_e(X^{(1)})]}{L\,\varepsilon^2}$$

$$= \frac{P_e\,(1-P_e)}{L\,\varepsilon^2}.$$

For such an estimator to be meaningful, ε must be chosen to be some fraction of P_e. For example, we may choose $\varepsilon = \frac{P_e}{10}$. In this case, we will have

$$\mathcal{P}(|\hat{P}_e - P_e| \geq \varepsilon) \leq \frac{100}{L}\,\frac{1-P_e}{P_e}$$

$$\leq \frac{100}{L}\,\frac{1}{P_e}$$

(recall that $0 \leq P_e \leq 1$.) Consequently, the number of simulation runs L must be greater than $100/P_e$ in order for $\hat{P}_e$ to have any significance. Therefore, for sufficiently small values of P_e (requirements in the range of 10^{-6} to 10^{-9} are not unusual,) the corresponding Monte Carlo estimates can be difficult, if not impossible, even for the most powerful computers. For example, to estimate a probability on the order 10^{-7}, we would need $10^9 \approx 2^{30}$ independent simulation runs. If the system is complex, computer run times may be prohibitive. Furthermore, the period of a typical random number generator is anywhere from 2^{15} to 2^{32} [15]. In order for these random number generators to approximate true "randomness," it is necessary that the total simulation

utilize only a fraction of this period. Thus Monte Carlo estimation of probabilities smaller than 10^{-6} is difficult because of the questionable quality of the computer generated random numbers. Of course, the advantage of Monte Carlo simulation is that it is often the only way to evaluate the performance of complex systems. Consequently, it is highly desirable to develop simulation techniques which retain the ability to simulate complex systems, yet require substantially fewer simulation runs to estimate small probabilities.

Importance sampling (IS) is a modified *Monte Carlo* simulation technique which, in comparison to ordinary Monte Carlo, may reduce by orders of magnitude the number of simulation runs required to obtain a specified estimator accuracy [1-31]. This technique arises from the observations that the events of importance, namely errors, typically occur very rarely by the underlying noise processes. The simulation efficiency can be improved if errors can be made artificially to occur more often in such a way that true error events probabilities can be estimated from the inflated ones. The basic principle behind importance sampling is to simulate noisier conditions than the actual operating conditions, so that the simulation of these events can be made without needing a very large number of samples. This is done by simulating using a distribution that generates the random inputs in the simulation which is different that the true distribution. Simulating a high noise environment, of course, produces more errors. In order to properly estimate the relative frequency of these error events for the actual low noise operating conditions, the simulation outputs are weighted by the ratio of the actual probability distribution to the simulation distribution. The end result is an unbiased estimator with a substantially lower variance in comparison to the ordinary Monte Carlo estimator using the same number of simulation runs, or what is usually sought, the number of importance sampling simulation runs can be made much less

than the ordinary Monte Carlo ones for a given variance or accuracy.

Importance sampling seems to have first appeared at a conference in 1949 [1], and it has found wide spread applications in many diverse fields. Recently, importance sampling has become quite popular in digital communications [5-6], [10-21], [16], [21], [23-24], [30], detection [12], [20], [31], Network simulation and queueing systems [9], [15], [17], [20], [27]. There has been substantial success in applying this technique to nonlinear and/or non-gaussian channels. In particular, optical and satellite communication channels. Most of this work, however, was ad hoc and has mainly been revolved around choosing various stretched and shifted versions of the true distribution in order to achieve improvements in the importance sampling estimator performance. Such methods, have been referred to as *conventional importance sampling* (CIS) techniques in the communications literature [21], [24], [29]. Many of these CIS methods are based on optimized variance scaling schemes operating on the true distribution. It turns out that when CIS was first implemented, good performance has been obtained (for simple channel models). However, when CIS was applied to more realistic channel models and with coding, the computational efficiencies were not appreciably reduced [29]. A more recent importance sampling technique which we shall refer to as the *mean translation biasing method,* has been proposed in the communication literature. This new method was first proposed by D. Lu and K. Yao [21][1], and further refined by Sadowsky and Bucklew [26]. In addition, it was shown that this technique is more powerful and more efficient than the CIS approach [21], [26]. In particular, under certain conditions, Sadowsky and Bucklew [26] showed that the mean translation biasing method is the unique asymptotically optimal scheme.

1. In [21], this method was referred to as the *improved importance sampling* (IIS) technique.

In this thesis, we consider the application of importance sampling to the simulation of systems that employ the *stack algorithm [44], [57].* The stack algorithm or *Z-J algorithm* of Zigangirov and Jelinek is a *sequential tree searching algorithm* whose goal is to find the maximum metric path through a decision tree with *random node metrics*. Our main application is to the *sequential decoding* of a general class of error control codes that are called *convolutional codes [57].* Specifically, we consider the simulation of stack algorithm sequential decoders decision processes.

Sequential decoding was introduced by Wozencraft [35] as the first efficient and practical scheme for decoding convolutional codes. This scheme is independent of the decoder memory, and hence arbitrary low error probabilities can be achieved provided operating under cutoff rate [57]. Its main drawback is its inherent inability to deal effectively with severe bursts of noise. Specifically, severe noisy frames may occasionally take a large amounts of computation, causing information to be lost.

A key characteristic of the stack algorithm is that the number of computations per correct decision is a random variable [38], [45], [57]. we denote this random variable C. Thus in order to assess the performance of stack algorithm sequential decoders, the probability distribution of C must be determined. A great deal of work has gone into the statistical analysis of the distribution of C. The bulk of this work is the classical information theoretic analysis of sequential decoding. This analysis indicates that the distribution of C which we shall refer to as the *distribution of computation* has a *Pareto tail,* a function of the channel, but independent of the code constraint length [41], [45], [54], [57]. The main drawback of this analysis is that it is based on random coding arguments. Such arguments, by their very nature, are not tied directly to the properties of specific codes. As a consequence, the classical analysis cannot predict the distribution of computation for a specific convolutional code. It turns out that there

is abundant experimental evidence that shows that the distribution of computation has indeed a Pareto tail for any code. However, it has been noticed and demonstrated that the distribution of computation depends also on the distance property of the convolutional code [57], [59]. Note that the classical analysis exhibits no such dependence.

One could continue the above discussion of the statistical analysis of the distribution of computation for sequential decoders, but the point should now be clear. There are no analytical results expressing the distribution of computation explicitly in terms of specific code parameters. Hence for performance evaluation of sequential decoders, it is necessary to use computer simulations. It turns out that simulating sequential decoders using a brute force Monte Carlo approach could prove to be extremely difficult , if not impossible, especially for low noise conditions.

This thesis presents new efficient importance sampling techniques for estimating the distribution of computation of stack algorithm decoders. The first method is called the *reference path method.* The reference path method is based on the distance structure of the code being simulated. The second method is the *partitioning method.* This method is motivated by the asymptotics of *large deviations theory [26] and an information theoretic ensemble* averaging argument. Finally the third method is called the *M-method.* This technique is basically a variation of the partitioning method. We note that in all of the above schemes, we do not consider branching errors probabilities. In other words, we assume that the decoder ultimately chooses the correct transmitted path after the search is terminated.

The error probability of sequential decoders is also known as a random coding average only [57], [59]. That is, as in the case of the distribution of computation, it is obtained in the form of averages over the ensemble of random convolutional codes.

Based on this random coding analysis, it is known that the error probability of sequential decoders is a function of only the code constraint length, the transmission channel, and the code rate [57]. It has been noticed, however, that this error probability does depend on the distance structure of the convolutional code. But as in the case of the distribution of computation there are no analytical results that express this probability in terms of the code distance properties, nor in terms of specific code parameters. Hence, in order to evaluate the error probabilities of sequential decoders, it is necessary to use computer simulations. In this thesis, we shall present another importance sampling technique which we will refer to as the *error event simulation method*. The error event simulation method will be used to estimate bit error rates for stack algorithm decoders.

The organization of this thesis is as follows. In Chapter 2, a general background on importance sampling is presented. In particular, we shall derive the expression of the unconstrained optimal importance sampling simulation distribution.

Chapter 3 presents some background on tree searching and the stack algorithm. In addition, it will present a new theorem which we shall refer to as the *fundamental theorem*. This theorem forms the structural foundation for the simulation of the stack algorithm.

In Chapter 4, we shall present our new importance sampling simulation schemes for estimating the distribution of computation.

Chapter 5 demonstrates the power and accuracy of the above simulation techniques by presenting some simulation results for the rate 1/2 and constraint lengths 5, 14, and 21 convolutional codes operating on the binary symmetric channel and the additive white gaussian noise channel.

Chapter 6 presents the error event simulation method and demonstrates its potential by presenting some simulation results.

In Chapter 7, we shift our focus to a different application. In this chapter, we will present a new importance sampling technique for simulating the *Sequential Edge Linking* (SEL) algorithm. The SEL algorithm is a stack algorithm technique for detecting edges in images. Our main objective in this chapter is to develop some of the basic theory relevant to this application, and to discuss the key issues that are related to the SEL simulation via importance sampling.

Finally, Chapter 7 concludes the thesis by commenting on the results of this work and discussing possible future research topics.

CHAPTER 2
IMPORTANCE SAMPLING

2.1 Introduction

Importance sampling is a Monte Carlo simulation technique in which the simulation data is generated using a simulation distribution which is different from the true underlying distribution. The importance sampling estimator then weights the simulation data by an appropriate likelihood ratio in order to form an unbiased estimate of the desired parameter. This method is called importance sampling because the simulation distributions which minimize the estimator variance also tend to increase the relative frequency of the "important events."

The goal of importance sampling is to select a simulation density which tends to minimize the number of simulation runs (and hence, less computations) to obtain a specified accuracy. The unconstrained optimal simulation distribution is well known, and in fact, under certain conditions this distribution yields to a perfect estimator. That is, an estimator with a zero variance. However, the unconstrained optimal solution is not practical because it assumes knowledge of precisely the parameter that we wish to estimate. Thus the practical problem of importance sampling is to obtain the most efficient simulation distribution from a suitably large class of candidate distributions that are determined by implementation constraints.

In this chapter, we shall present a brief overview of the basic theory behind importance sampling. In addition, we will also derive the expression of the unconstrained optimal importance sampling distribution. This distribution will indicate some of the key properties that good importance sampling distributions should have.

2.2 General Theory Behind Importance Sampling

2.2.1 Importance Sampling

Let $(\Omega, \mathcal{F}, P)$ be a probability space, X be an Ω–valued random element and g(.) be a real valued function of X. We shall consider the problem of estimating

$$\alpha = E[\, g(X)\,] \tag{2.1}$$
$$= \int g(\omega)\, P(d\omega)$$

Let $P^*(.)$ be a probability distribution such that P(.) is absolutely continuous with respect to $P^*(.)$; that is, $P(A) > 0$ implies $P^*(A) > 0$ for every $A \in \mathcal{F}$. *Importance sampling* involves choosing the proability distribution $P^*(.)$ and observing that α can be written as

$$\alpha = \int g(\omega) \frac{dP}{dP^*}(\omega)\, P^*(d\omega) \tag{2.2}$$

where $dP/dP^*(.)$ is the radon-Nikodym derivative of P(.) with respect to $P^*(.)$. The name importance sampling derives from the fact that one can choose P^* to be large in the regions that are most important, namely where $|g(\omega)|$ is large. We shall call P^* the *importance sampling distribution.*

A standard simulation formula for estimating an expected value is to use a sample mean expression. In this case, the importance sampling estimator is obtained as a

"empirical evaluation" of the integral (2.2) instead of (2.1). For $\ell = 1,...,L$, one generates independent random samples $X^{(1)}, \ldots, X^{(L)}$ using the importance sampling distribution $P^*(.)$ instead of the true distribution $P(.)$. The *importance sampling estimator* is

$$\hat{\alpha} = \frac{1}{L} \sum_{\ell=1}^{L} g(X^{(\ell)})\, w(X^{(\ell)}) \tag{2.3}$$

where

$$w(\omega) = \frac{dP}{dP^*}(\omega). \tag{2.4}$$

The likelihood ratio $w(\omega)$ is called the *importance sampling weight* at ω. Note that if $P^*(.) = P(.)$, then $W(\omega) \equiv 1$ and the sample mean estimator (2.3) is reduced to the ordinary Monte Carlo estimator.

2.2.2 Bias and Variance

Let $var^*[.]$ and $E^*[.]$ denote the variance and expectation operations for the importance sampling distribution $P^*(.)$. Because the simulation data $X^{(1)}, \ldots, X^{(L)}$ are independent random samples generated using $P^*(.)$, it follows from (2.3) and (2.4) that

$$E^*[\hat{\alpha}] = \frac{1}{L} \sum_{\ell=1}^{L} \int g(\omega) \frac{dP}{dP^*}(\omega)\, P^*(d\omega)$$

$$= \alpha.$$

Consequently, the importance sampling estimator (2.3) is unbiased.

Likewise, since the simulation data is independent and identically distributed (i.i.d.), it follows that the variance of the importance sampling estimator $\hat{\alpha}$ is

$$\mathrm{var}^*(\hat{\alpha}) = \frac{1}{L}\left[E^*\left[\left(g(\omega)\frac{dP}{dP^*}(\omega)\right)^2\right] - \left(E^*[\hat{\alpha}]\right)^2 \right]$$

Consequently, we have

$$\mathrm{var}^*(\hat{\alpha}) = \frac{1}{L}\left[\eta(P^*) - \alpha^2 \right] \tag{2.5}$$

where

$$\eta(P^*(.)) = \int \left(g(\omega)\frac{dP}{dP^*}(\omega)\right)^2 P^*(d\omega). \tag{2.6}$$

Note that the impact of the choice of the importance sampling distribution $P^*(.)$ is completely represented by the functional $\eta(.)$. Consequently, our objective is to minimize $\eta(.)$. Furthermore, notice that (2.6) clearly indicates that good choices of $P^*(.)$ will tend to be large relative to $P(.)$ in the region of "importance", namely where $|g(\omega)|$ is large, hence diminishing the variance of $\hat{\alpha}$ for a fixed L, or equivalently, reducing the number of simulation runs L for a given variance or accuracy.

The next theorem shows how to choose $P^*(.)$ in order to minimize (2.6). This result is well known [7], but we shall include its proof for completeness.

Theorem 2.1: Assume $E[\,|g(X)|\,] < \infty$.

$$\eta(P^*(.)) = \int \left(g(\omega)\frac{dP}{dP^*}(\omega)\right)^2 P^*(d\omega) \geq E[\,|g(X)|\,]^2 \tag{2.7}$$

and equality holds if and only if

$$P^*(d\omega) = \frac{|g(\omega)|}{E[\,|g(X)|\,]} P(d\omega) \tag{2.8}$$

$$\triangleq P_o^*(d\omega).$$

We shall call $P_o^*(.)$ the *unconstrained optimal importance sampling distribution.*

Proof: First observe that (2.7) follows directly if (2.8) is substituted into the left hand side of (2.7). Next by the Jensen's inequality

$$\int \left[g(\omega) \frac{dP}{dP^*}(d\omega) \right]^2 P^*(d\omega)$$

$$\geq \left[\int |g(\omega)| \frac{dP}{dP^*}(\omega) P^*(d\omega) \right]^2$$

$$= E[\,|g(X)|\,]^2$$

and equality holds if and only if

$$|g(\omega)| \frac{dP}{dP^*}(\omega) = c \quad \text{a.s.} \quad P^*(.),$$

where c is some constant.

□

Observe that, by (2.6) and (2.7) it follows that $var_o^*(\hat{\alpha}) \leq var^*(\hat{\alpha})$ where $var_o^*(.)$ denotes variance operation for the unconstrained optimal importance sampling distribution $P_o^*(.)$.

Corollary 2.1: If $g(\omega) > 0$, then the optimal importance sampling distribution is

$$P_o^*(d\omega) = \frac{g(\omega)}{\alpha} P(d\omega) \tag{2.9}$$

Furthermore, if $P_o^*(.)$ is used then $\eta(P_o^*(.)) = \alpha^2$, and hence by (2.5) the importance estimator (2.3) is perfect; that is, $var_o^*(\hat{\alpha}) = 0$.

As we have mentioned earlier, the unconstrained optimal importance sampling distribution $P_o^*(.)$ is well known. Unfortunately, $P_o^*(.)$ is not a practical solution as it assumes knowledge of precisely the parameter which we wish to estimate. However, $P_o^*(.)$ indicates certain features which good importance sampling distributions should have. For example, the simulation relative frequency of the event $\{\omega \in d\omega\}$, which is just $P_o^*(d\omega)$, is directly proportional to the true relative frequency $P(d\omega)$. The intuition behind this is that the "important" differential events are those events for which $P(d\omega)$ is relatively large. These are precisely the most likely events to be observed under the true distribution. Consequently, good importance sampling distribution should tend to maximize the relative frequency of these important "differential" events. Such distributions should be selected so that (2.6) is minimized. Hence, good importance sampling distributions should inflate the probability mass assigned by P(.) where $|g(\omega)|$ is large, and deflate it where $|g(\omega)|$ is small. The choice of good importance sampling distributions is the key issue in importance sampling. Further discussions of this important subject will be considered later.

As a final remark in this chapter, we note that when $g(\omega) = I_E(\omega)$; that is, g(.) is the indicator function of an event E then α is simply the probability of that event. Thus, if E is an error event, then by Corollary 2.1 note that $P_o^*(.)$ produces errors with probability 1. Hence, when used to estimate error probabilities, good importance sampling distributions should tend to produce a lot of errors.

CHAPTER 3
TREE SEARCHING AND THE STACK ALGORITHM

3.1 Tree Searching

3.1.1 Introduction

Consider the problem of finding the maximum metric path through a large decision tree with random node metrics. Because the computational complexity grows exponentially with the tree depth, it is often not possible to determine the maximum metric path in the tree using either an exhaustive search or an optimal dynamic programming algorithm. A practical alternative is provided by a class of algorithms called *sequential tree searching algorithms*. These algorithms have been developed primarily in the coding theory literature with early contributions of Wozencraft, Fano, Zigangirov and Jelinek. These algorithms were originally developed for decoding *convolutional codes* based upon the inherent tree structure possessed by this class of codes. We shall define a sequential tree searching algorithm as an algorithm which computes the metric of paths by extending, by one branch only, a path which has been already examined, and which bases the decision on which path to extend only on the metrics of examined paths. The reader is referred to [33-61] for more background and in depth discussions of sequential tree searching algorithms.

3.1.2 Tree Searching

We are interested in finding the maximum metric path through a tree with random node metrics. In this context, a *tree,* is a directed graph consisting of *nodes* which are connected by *branches.* Starting from the unique *root node,* a *path* is a sequence of successively connected nodes. For each node there is a unique path which connects that node to the root node. The *depth* of a node shall refer to the number of branches on the path connecting that node to the root node. A path may be identified by its first node and the sequence of branches which connect the path nodes. The *descendent nodes* of, say node γ at depth j, are those nodes at depth j+1 which are connected to node γ by a single branch. We shall consider trees for which each node has b branches emanating from it. Hence, ln(b) is the *exponential growth rate* of the tree.

Each branch in the tree will have a random weight called the *branch metric.* For each node, the *path metric* is the sum of branch metrics along the path which starts at the root node and terminates at that node. The metric of the root node is zero. In particular, there is a unique *correct path,* and on this correct path the sequence of path metrics should tend on the average to increase. That is, the correct path metric process should have a *"positive drift."* Conversely, as we follow any path disjoint from the correct path the path metrics should tend to decrease. That is, the node metric process on paths disjoint from the correct path should have a *"negative drift."* In this application, we will *assume that the average behavior of the path metric is to increase along the correct path and to decrease otherwise.* Our goal is to identify the correct path. Given the observation of the path metrics, the path most likely to be correct is the one which terminates with the maximum path metric value. Thus, we shall consider algorithms which attempt to find the maximum metric path.

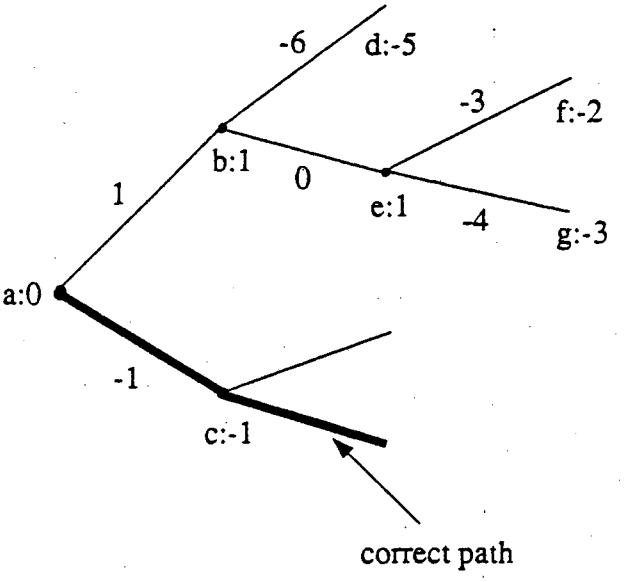
3.2 The Stack Algorithm

3.2.1 Algorithm Description

There are many algorithms which fall under the heading of sequential tree searching algorithms, most notably, the *Fano algorithm* [36] and the *stack* or *Z-J algorithm* of Zigangirov and Jelinek [37], [44]. However, probably the most basic algorithm in this class, and certainly the easiest to understand and describe is the *stack algorithm*. In this algorithm, the *stack* is a list of previously examined nodes and their associated *node metrics*. The stack is ordered by the node metric values, the node with the largest metric is placed on top, and the others are listed in order of decreasing metric. The *top-of-stack* node is the maximum metric node on the stack. The stack algorithm consists of the following steps:

1. Initialize the stack with the root node.
2. Compute the node metrics for each direct descendent of the top-of-stack node.
3. Remove the top-of-stack node from the stack and replace it with its direct descendents.
4. Reorder the stack according to node metric values.
5. Stop if the top-of-stack node in the stack is at the end of the tree. Otherwise return to step 2 and continue.

An example of the stack algorithm search is illustrated in Figure 3.1. This figure shows a binary tree and the path metrics up to depth 3. In addition, it shows the first few steps of the search, indicating the path and their associated metrics after each new pair of nodes have been examined and the stack reordered.



History of ordered stack

Search index	Ordered stack
0	a
1	b, c
2	e, c, d
3	c, f, g, d
	⋮

Figure 3.1: An example of the stack algorithm search.

Because the correct path metric process has a positive drift, and since all incorrect paths have a negative drift, the correct path will tend to "float" to the top of the stack. However, because long paths are built up node by node, it is entirely possible that the

random metrics might become too noisy and as a consequence the algorithm can from time to time mistakenly follow an incorrect path for some depth in the tree. When followed far enough, these incorrect paths should eventually be halted by the resulting decrease in metric. From this, it can be seen that the algorithm is forced to waste computation time on the exploration of incorrect paths which are eventually abandoned. This brings out a key characteristic of sequential tree searching algorithms, namely the number of incorrectly hypothesized branching decisions per correct decision is a random variable.

The *correct node j,* shall refer to the unique node on the correct path at depth j. Next consider the *j'th incorrect subtree* $\mathcal{N}_j$, that is, the subtree of incorrect nodes on paths diverging from the correct path precisely at node j. $\mathcal{X}_j$ shall denote the subset of $\mathcal{N}_j$ which is actually hypothesized (that is, nodes which were on the stack at least once) by the stack algorithm. $\mathcal{X}_j$ is called the *j'th incorrect subset.* $\mathcal{X}_j$ is a random set, and

$$C_j \triangleq \text{the number of nodes in } \mathcal{X}_j$$

is a random variable. As a consequence, it follows that

$$E[C_j] = \begin{array}{c}\text{the expected number of} \\ \text{computations per correct node}\end{array}$$

and

$$\mathcal{P}(C_j \geq M) \qquad \text{for} \quad M \geq 1$$

are obviously the relevant indicators of the algorithm's computational requirement.

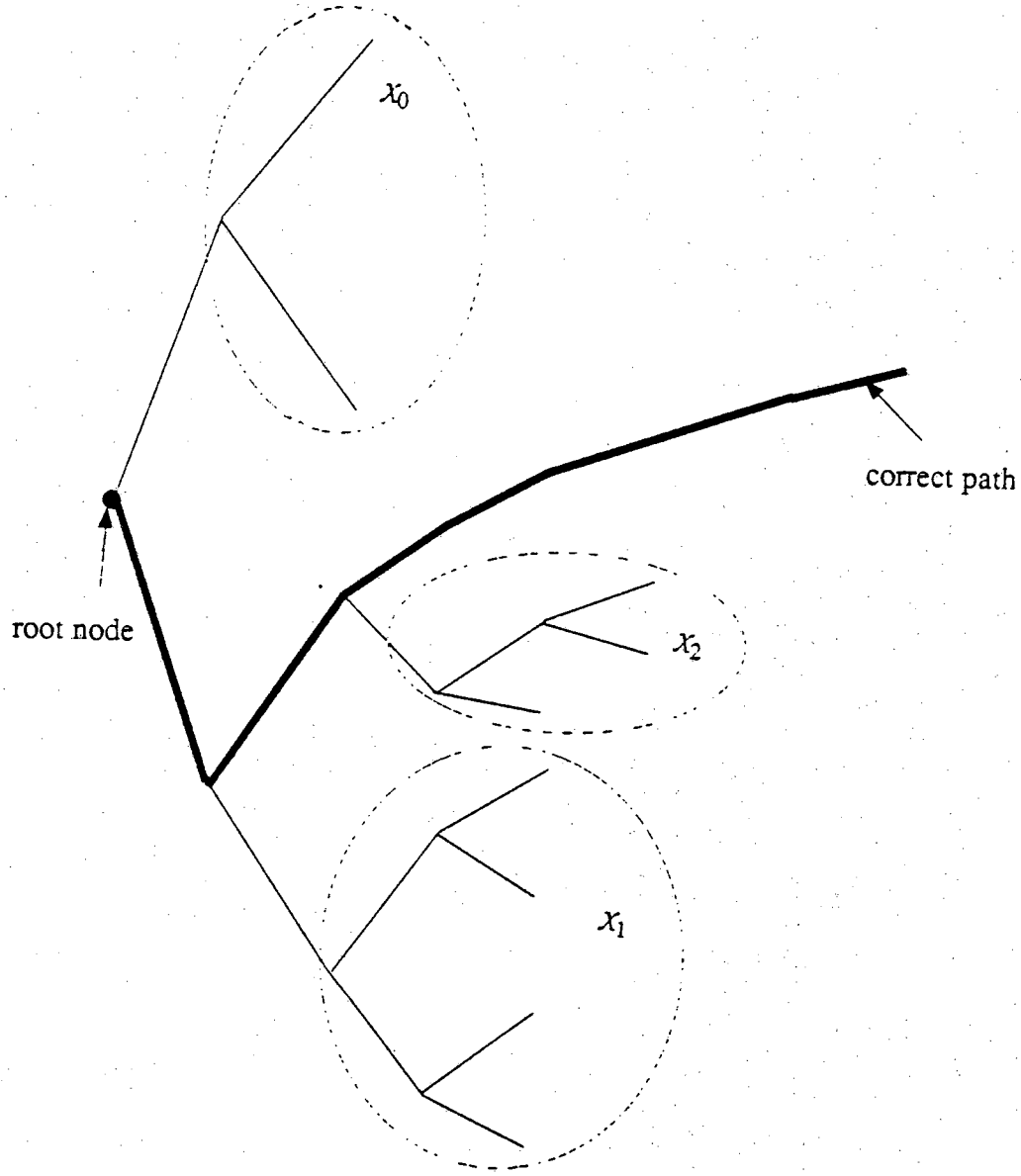


Figure 3.2: An illustration of the subsets X_j for a binary tree.

A great deal of work has gone into the statistical analysis of the probability distribution of C_j and its average value [38], [41], [45], [49] , [53], [61]. The bulk of this work is the classical "information theoretic" analysis of sequential decoding. This previous work assumes two strong conditions: 1) the path metric processes have i.i.d. increments; and 2) only a special "maximum likelihood - like" metric known as the Fano metric has been considered. We should note that real codes actually violates condition 1). However, the information theoretic analysis avoids this problem by using an ensemble of time varying convolutional codes and then proceeds with a probabilistic "ensemble average" performance analysis. Under the above assumptions, it has been demonstrated that the distribution of C_j has a *Pareto tail [57].* That is, $\mathcal{P}(C_j \geq M) \sim M^{-\rho}$ where ρ is called the *pareto exponent.* In addition, it has been shown that as some parameter (such as code rate or noise variance) is varied there exists a critical operating point which we shall call the point of *computational cutoff.* Operating below the cutoff point ensures $E[C_j] < \infty$, while $E[C_j] = \infty$ above cutoff. We refer the reader to [57, chapter 6] for further discussion of this work.

The main problem associated with the above work is that the analysis has been developed in the context of "ensemble average" techniques, and thus cannot predict $E[C_j]$ or the distribution of C_j for specific cases. In a recent work, we have investigated computational cutoff conditions which determine whether $E[C_j]$ is finite or infinite [61]. This analysis differs from the previous analysis in several aspects. Our analysis was developed using *large deviations* theory[1] and is more closely associated with methods of sequential decision theory. We do not require the Fano metric assumption and hence, our analysis is more readily applied to problems which do not relate well to the coding problem. In place of i.i.d. branch metrics we have

1. Large deviations theory is a general probability theory of exponential convergence of small probabilities [26].

considered stationary branch metric processes which are governed by an underlying Markov chain. The Markov chain state space may be infinite dimensional, hence, this model provides a rich class of stationary branch metric process distributions. Of course, the cost of this expanded generality is that we obtain a weaker result: we do not obtain the Pareto tail result. Instead, two disjoint conditions have been derived: the first implies $E[C_j] = \infty$ and the second implies that $E[C_j] < \infty$. These conditions, in effect, provide an upper and a lower bound on the point of computational cutoff. It turns out that these bounds are tight under the classical i.i.d. Fano metric assumption. We should note here that the basic idea behind this analysis was based on the following observations: 1) the probability of searching an incorrect path in the tree decreases exponentially with depth; and 2) the number of paths in the tree grows exponentially with depth. It turns out that the point of computational cutoff corresponds to the case when the rate of decrease of the probability of searching an incorrect path in the tree is equal to the tree growth rate.

Now consider the stack algorithm, and suppose that the correct node j has been hypothesized. Next consider a fixed incorrect node δ, which has depth j+n and is on an incorrect path diverging from the correct path at depth j. δ is a candidate node for x_j.

Define

$$S_n^c = \begin{matrix}\text{metric of} \\ \text{correct node} \\ n+j\end{matrix} - \begin{matrix}\text{metric of} \\ \text{correct node} \\ j\end{matrix} = \sum_{k=1}^{n} Z_k^c \qquad (3.1)$$

where Z_k^c is the k+j'th branch metric on the correct path. Likewise, considering a fixed incorrect path in the j'th incorrect subtree, define

$$S_n^i = \begin{matrix}\text{metric of} \\ \text{incorrect} \\ \text{node } n+j\end{matrix} - \begin{matrix}\text{metric of} \\ \text{correct} \\ \text{node } j\end{matrix} = \sum_{k=1}^{n} Z_k^i \tag{3.2}$$

where Z_k^i is the k+j'th branch metric on the incorrect path.

Define Γ to be the minimum metric value along the correct path metric after time j:

$$\Gamma \triangleq \min_{n \geq 0} S_n^c. \tag{3.3}$$

We shall also define

$$N_\Gamma^i \triangleq \min \{ n : S_n^i \leq \Gamma \}. \tag{3.4}$$

Lemma 3.1: Consider a fixed incorrect path in the j'th incorrect subtree and let δ, S_n^c, and S_n^i be defined as above. Then

(i) δ is always hypothesized when $S_m^i > S_m^c$ for all $m = 1,2,...,n$; and

(ii) δ is never hypothesized if $S_m^i < \Gamma$ for some $m \leq n$.

Proof: The proof of part (i) of Lemma 3.1 is trivial; the incorrect nodes will always occupy a higher position on the stack than the correct nodes, at least until both the correct and incorrect paths are searched to the point where the correct path metrics are larger. Part (ii) is a standard result which has been proven by several authors (see Lemma 6.2.1 in [57].)

□

We note that Lemma 3.1 provides the structural foundation for the analysis of the partitioning method, an importance sampling scheme to be discussed in the next chapter. The conclusions of Lemma 3.1 are illustrated in Figure 3.3.

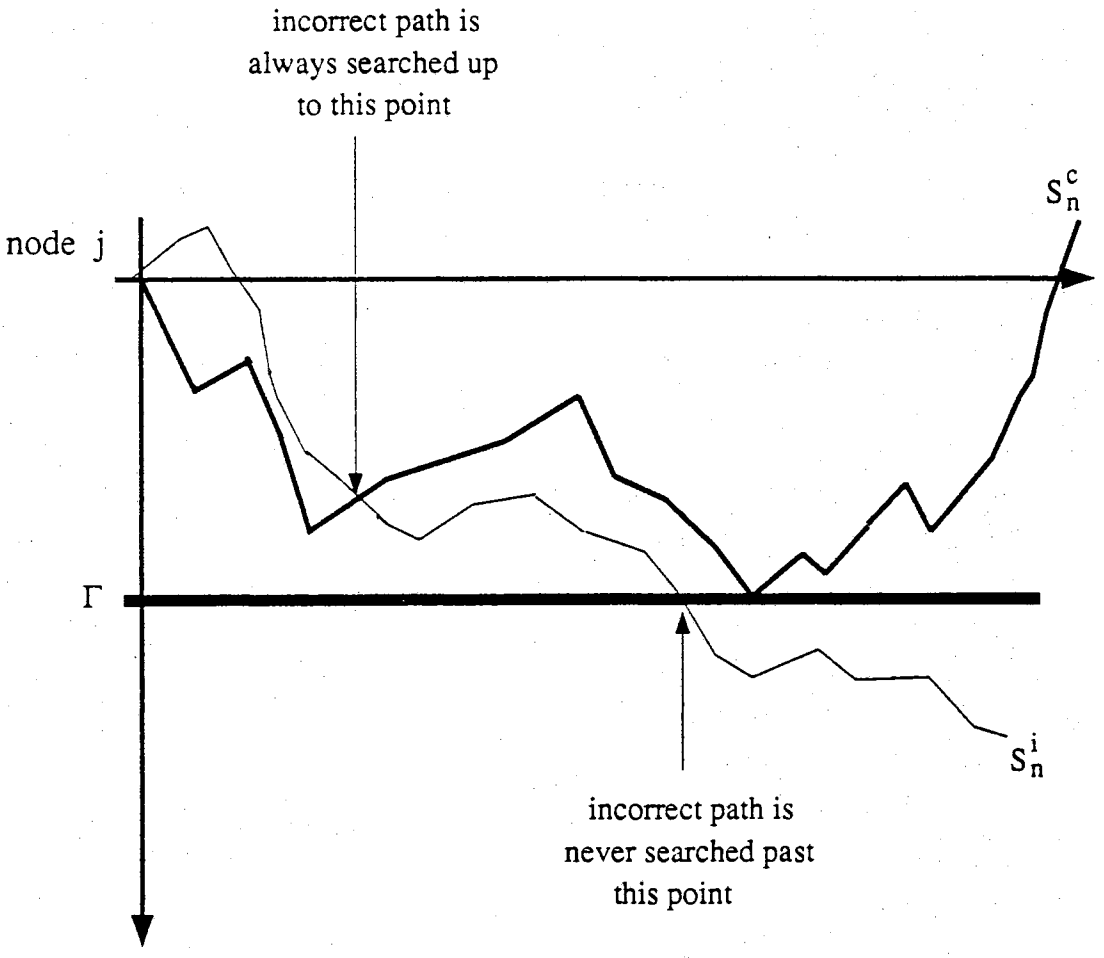


Figure 3.3: An illustration of the conclusions of Lemma 3.1.

3.2.2 The Path Metric

Recall that one of the key elements in the stack algorithm, or any sequential tree searching algorithm, is the concept of path or node metric. The path metric provides a means of comparing all paths hypothesized and reflects in a sense the "closeness" of a given path to the correct path. In order for the stack algorithm to proceed correctly, the path metrics should exhibit certain desirable qualities. First, path metrics should tend to increase along the correct path and decrease elsewhere. More specifically, the correct path metric should have a *slight positive drift* and incorrect paths should have a *large negative drift.* It turns out that these drifts can be adjusted using a *bias term* (a constant substructed from all branch metrics.) Second, because the stack is ordered according to metric values only, successive nodes on the stack may have different tree depths. Thus, the stack algorithm compares nodes of different lengths in its decision process. Algorithms of this type are called *metric first algorithms [60].* It is therefore essential that the path metrics should not be biased by path length. That is, the metric should not favor longer paths over shorter ones and vice versa. Third, path metrics should exhibit a recursive computational efficiency, namely the path metric of a newly hypothesized node should be obtained from the metric of its parent node by adding a correction value which depends only on the new node.

3.3 The Fano Algorithm and Variations of the Stack Algorithm

The stack algorithm is a simplification of a number of successively discovered sequential tree searching algorithms, each of which was progressively simpler to analyze and describe. Perhaps, one of the best features of the stack algorithm is that it requires few metric computations, but this computational savings is offset to a large extent by the computations involved in reordering the stack after every iteration. In an

attempt to alleviate this problem, few variations of the stack algorithm have been developed in the literature. The first is the *stack bucket algorithm* which has been introduced by Jelinek in 1969 [44]. In this algorithm, the stack is divided into smaller stacks which are called *buckets,* with each bucket corresponding to an interval of possible metric values. At each iteration, the paths are placed in the bucket appropriate to their metrics. In contrast to the stack algorithm, in this algorithm no ordering of the metrics in the bucket takes place. Furthermore, the path to be extended is taken from the top of the highest non-empty bucket.

In 1975, Haccoun and Ferguson [55] have introduced the *generalized stack algorithm.* In this algorithm, the paths are ordered and extended as in the stack algorithm, but more than one path can be extended at the same time. The remerging phenomenon (see Chapter 4) is also exploited in this algorithm. When two paths remerge, the path with the lower metric is deleted from the stack, thereby eliminating redundant paths in the stack.

The *multiple stack algorithm* is another variation of the stack algorithm introduced by Chevilat and Costello in 1977 [56]. This algorithm eliminates the problem of buffer overflow which is usually associated with the stack algorithm. The manner in which this is done involves the introduction of additional smaller stacks to which the generalized stack algorithm turns when the main stack fills up. The first of these stacks is made large enough so that only very noisy cases force the use of additional stacks. In contrast to the stack algorithm which advances slowly in these noisy situations because it is forced to search many incorrect subtrees before extending the correct path, the multiple stack algorithm penetrates quickly through the tree and finds "reasonably good" paths. Thus, this algorithm trades away some performance, as its search space becomes smaller, for a substantial improvement in speed.

We conclude this section by briefly describing the *Fano algorithm,* which is generally considered to be the most practical sequential tree searching algorithm to implement. The Fano algorithm was actually the first algorithm to be developed for sequential tree searching. This algorithm was first proposed by Wozencraft [35], and subsequently modified by Fano [36]. One of the best features of the Fano algorithm is that it examines only one path at a time, thereby eliminating the storage of all but one path and its metric. Basically, the Fano algorithm continues to search the most probable (largest metric) path as long as its metric is growing. If the metric begins to drop significantly, the algorithm backs up and extends other paths stemming from previous nodes on the already searched path. This is accomplished by varying a running threshold T which changes by multiples of some constant Δ. This threshold is raised by Δ if the metric is growing on a forward search and lowered by Δ during backward searches. The decision structure in this algorithm is done in such a way that no node is ever searched forward twice with the same threshold.

The Fano algorithm has been the subject of extensive treatments in the coding literature [43]. Its analysis, as well as, its performance is essentially the same as the stack algorithm. In fact, Giest (1973) has shown that the Fano algorithm always chooses the same path through the tree as the stack algorithm [53]. The only difference is that in the Fano algorithm a path may be searched several times, while in the stack algorithm it is searched only once. This disadvantage is usually offset by the substantial reduction in storage requirement.

3.4 A Format for the Simulation of the Stack Algorithm

3.4.1 Terminology and Definitions

We begin this section by developing various definitions which are needed for the discussions to follow. Let,

$\alpha, \beta, \gamma, \delta$ = tree nodes

M_δ = node metric at node δ

D_δ = { the direct descendent nodes of node δ }

S_γ = subtree emanating from node γ (γ and its descendents)

TOS node = the top-of-stack node

F = the terminal path (i.e. the final hypothesized path in the tree)

E_γ = the event { node γ is on **F** }

$\mathcal{G}_\gamma = \sigma(I_\delta : \delta \in S_\gamma)$ where,

$$I_\delta \triangleq \text{indicator random variable for node } \delta$$

$$= \begin{cases} 1 & \text{if } \delta \text{ is searched} \\ 0 & \text{if } \delta \text{ is not searched} \end{cases}$$

Now, for any event A, we shall let

$$I_A \triangleq \text{indicator random variable for A}$$

$$= \begin{cases} 1 & \text{if event A occurs} \\ 0 & \text{if event A does not occur} \end{cases}$$

Next for our purposes the terms "reached", "searched", and "extended", will be defined as follows:

(1) We will say that a node has been *reached* by the algorithm, if it becomes the TOS node.

(2) We will say that a node has been *searched* by the algorithm if it is on the stack but not necessarily extended, and

(3) *Extending* a node δ will refer to the process of deleting δ from the stack and replacing it with its direct descendents.

Note that all nodes that are "reached" are subsequently extended.

In the sequel, we shall let Ω denotes the underlying probability space and ω be an event in Ω. A typical example of an event ω in the sample space Ω can be defined as follows: Assuming that the branch metrics are independent, then ω can be defined to be the collection of all the branch metric values associated with the tree. That is, the event ω specifies all the values of the branch metrics in the tree. It turns out that for the proof of the main result of this chapter, namely Theorem 3.1, it is not necessary to specify the underlying probability space. Consequently, in the sequel we shall assume that Ω is given.

3.4.2 The Fundamental Theorem

We are interested in estimating some key parameters which are associated with the stack algorithm. A typical example is the estimation of the *distribution of computation:*

$$\mathcal{P}(C_j \geq M) \qquad \text{for } M \geq 1 \tag{3.5}$$

where j is the correct node at depth j. In general, however, we shall consider the following problem.

The Basic Problem: Given the event E_γ, estimate

$$E[\, X \mid E_\gamma \,]$$

where X is a $\mathcal{G}_\gamma$-measurable random variable.

In other words, *given the event that node γ is on the terminal path* **F**, we are interested in *estimating expectations of random variables that are functions of only the algorithm searching history in the subtree* S_γ. To motivate this problem and illustrate its significance consider the following examples:

(i) Suppose that γ is a given node on the correct path and let,

$\alpha_1 \triangleq$ direct descendent node of γ which is also on the correct path; and

$\beta_1 \triangleq$ direct descendent node of γ which lies on the terminal path **F**.

Next let E denotes the event $\{\, \beta_1 \neq \alpha_1 \,\}$, which is the basic error event, and E^c its complement. Then

$$X = I_E \tag{3.6}$$

$\triangleq$ the indicator random variable for the event E

is $\mathcal{G}_\gamma$–measurable and

$E[\, X \mid E_\gamma \,]$ = the probability of error following the correct decision at node γ.

(ii) Let γ, α_1, β_1, and E^c be defined as above. Next let

$$C_\gamma \triangleq \left[\sum_{\delta \in \tilde{S}_\gamma} I_\delta \right] I_{E^c} \tag{3.7}$$

where

$$\tilde{S}_\gamma \triangleq \bigcup_{\delta \in D_\gamma - \beta_1} S_\delta .$$

Then C_γ is $\mathcal{G}_\gamma$–measurable, and

$E[\, C_\gamma \mid E_\gamma \,]$ = expected number of metric computations per correct decision.

As pointed out earlier, $E[\, C_\gamma \mid E_\gamma \,]$ is one of the relevant indicators of the algorithm's computational performance.

(iii) Let C_γ be defined as above, and let M be some positive integer. Next let

$$X \triangleq \begin{cases} 1 & \text{if } C_\gamma \geq M \\ 0 & \text{if } C_\gamma < M \end{cases} \tag{3.8}$$

Then X is $\mathcal{G}_\gamma$–measurable and

$$E[\, X \mid E_\gamma \,] = \mathcal{P}(C_\gamma \geq M \mid E_\gamma).$$

Thus, the above expectation will allow us to estimate the distribution of computation for the stack algorithm.

(iv) Let γ be defined as above and let

L_γ = length of branching errors following the correct decision at node γ.

That is, L_γ is the length of an error burst. Then L_γ is $\mathcal{G}_\gamma$–measurable and

$E[\, L_\gamma \mid E_\gamma \,]$ = the expected number of branching errors per correct decision.

(v) Let L_γ be defined as above, and let $\ell > 0$. Next let

$$X \triangleq \begin{cases} 1 & \text{if } L_\gamma \geq \ell \\ 0 & \text{if } L_\gamma < \ell \end{cases} \tag{3.9}$$

Then X is $\mathcal{G}_\gamma$–measurable and

$$E[\, X \mid E_\gamma \,] = \mathcal{P}(\, L_\gamma \geq \ell \mid E_\gamma \,).$$

Consequently, the error burst length distribution can be also written as an expectation of a $\mathcal{G}_\gamma$–measurable random variable.

These examples and others indicate that the problem of estimating most of the key parameters associated with the stack algorithm can indeed be formulated as in the basic problem.

Now recall that we are interested in estimating expectations of random variables which are $\mathcal{G}_\gamma$–measurable. Next observe that if one needs to generate the data associated with the whole tree in order to estimate such expectations, it follows that the simulation complexity is an exponential function of the depth of γ. Thus the simulation of such problems will be much simpler and more efficient if only the data associated with nodes in S_γ is generated.

Since the stack is ordered according to metric values only, the stack algorithm compares nodes of different tree depths in its decision process. Thus, at a given instant, the stack may contain nodes in both S_γ and S_γ^c[2]. Consequently, the search performed by the algorithm in S_γ will be affected by the search performed by the algorithm in S_γ^c before and/or after node γ has been reached, and therefore, to estimate expectations of $\mathcal{G}_\gamma$–measurable random variables knowledge of the entire history of the stack is required. However, if conditioned on the event E_γ (i.e., that γ is on the terminal path), then this last statement is no longer true. In other words, *given the event* E_γ, *then the search performed by the stack algorithm in* S_γ *is not affected by the search outside* S_γ

2. In this context, S_γ^c will denote the complement of S_γ.

before and/or *after reaching node* γ.

The next theorem proves this statement and thus provides the structural foundation for the simulation of the stack algorithm.

Suppose that node gamma is on the terminal path and let

$$\mathbf{F}_\gamma = \text{the terminal path in } \mathbf{S}_\gamma \text{ given the entire history of the stack}$$

$$\triangleq (\gamma, \beta_1, \beta_2, \beta_3, \cdots). \tag{3.10}$$

Next let $\mathbf{Z}_0 \triangleq \{ \gamma \}$ and for $i \geq 1$ define,

$$\mathbf{Z}_i \triangleq \{ \delta \in \mathbf{S}_\gamma \text{ which are searched before}$$

$$\beta_i \text{ is extended , and } \delta \notin \mathbf{Z}_j \text{ for } j < i \}$$

Observe that the $\mathbf{Z}_i$'s are disjoint and the $\bigcup_{i=0}^{\infty} \mathbf{Z}_i$ is simply the collection of all the nodes searched in $\mathbf{S}_\gamma$.

We are now ready to state our main result (Theorem 3.1.) The basic idea in the proof of this result is to show that given E_γ, then for any node $\delta \in \mathbf{S}_\gamma$, the event $\{ I_\delta(\omega) = 0 \text{ or } 1 \}$ does not depend on the stack algorithm searching history outside $\mathbf{S}_\gamma$ for any $\omega \in E_\gamma$. Because this hold for any node $\delta \in \mathbf{S}_\gamma$, it follows that given E_γ, then for any event E in $\mathcal{G}_\gamma$, the event $\{ I_E(\omega) = 0 \text{ or } 1 \}$ does not depend on the stack algorithm searching history outside $\mathbf{S}_\gamma$ for any $\omega \in E_\gamma$. The proof of Theorem 3.1 will then follow from this last statement.

Theorem 3.1 (The Fundamental Theorem): Let X be any $\mathcal{G}_\gamma$–measurable random variable, and let E_γ be defined as in Section 3.4.1. Then,

$$E[X \mid E_\gamma] = E[X \mid \gamma \text{ is the root node }]. \tag{3.11}$$

Proof: Let E_γ be defined as above and let us consider a fixed $\omega \in E_\gamma$. Note that in this case, the terminal path in $\mathbf{S}_\gamma$ is $\mathbf{F}_\gamma$ as specified by (3.10). Furthermore, notice that since ω is fixed, it follows that $\mathbf{F}_\gamma$ is determined.

Now let δ be a candidate node for $\mathbf{Z}_j$ with $j \geq 1$; that is, $\delta \in \mathbf{S}_\gamma$ and $\delta \notin \mathbf{Z}_i$ for $i=0, \cdots, j-1$. Consequently, it follows that there exists some node $\alpha \in \mathbf{Z}_i$ for some $i < j$ such that α has not been extended before β_j; and furthermore, there exists a path

$$(\alpha, \eta_1, \eta_2, \cdots, \eta_n, \delta)$$

which connects node α to δ. Note that if $\delta \in D_\alpha$, then this path is just (α, δ). Furthermore, observe that $\eta_1, \eta_2, \cdots, \eta_n$, and δ are all candidates for $\mathbf{Z}_j$.

Now

$$\delta \in \mathbf{Z}_j \quad \text{if and only if} \quad \eta_k \text{ are extended before } \beta_j \text{ for all } k=1, \cdots, n.$$

It follows that for all $j \geq 1$,

$$\delta \in \mathbf{Z}_j \quad \text{if} \quad \mathbf{M}_{\eta_k} > \mathbf{M}_{\beta_j} \quad \text{for} \quad k= 1, \cdots, n, \tag{3.12a}$$

and

$$\delta \in \mathbf{Z}_j \quad \text{only if} \quad \mathbf{M}_{\eta_k} \geq \mathbf{M}_{\beta_j} \quad \text{for} \quad k= 1, \cdots, n. \tag{3.12b}$$

Note that (3.12b) establishes only necessity and not sufficiency as in (3.12a). The reason is that whenever $\mathbf{M}_{\beta_j} = \mathbf{M}_{\eta_k}$ for some $j \geq 1$ and some $1 \leq k \leq n$, then whether the algorithm extends β_j or η_k will simply depend on 1) the way the direct descendent nodes are inserted into the stack, 2) the rule by which the stack is ordered when metric ties occur between the direct descendent nodes, and 3) the rule by which the stack is reordered when metric ties occur between the direct descendent nodes and prior examined nodes. Thus, the sets $\mathbf{Z}_i$ for all $i \geq 1$ are characterized by (3.12a), (3.12b), and the tie handling rule. As a consequence, it follows that whether a node $\delta \in \mathbf{S}_\gamma$ has

been examined or not by the stack algorithm is determined once E_γ is given. Furthermore, this fact does not depend on the stack algorithm searching history in S_γ^c. In other words, when we condition on the event that node γ is on the terminal path, then for any node δ in S_γ, the events $\{ I_\delta(\omega) = 1 \}$ and $\{ I_\delta(\omega) = 0 \}$ are (conditionally) independent from the stack algorithm searching history in S_γ^c. Consequently, for any fixed $\omega \in E_\gamma$, we conclude that

$$\mathcal{P}(I_\delta(\omega) = 1 \mid E_\gamma) = \mathcal{P}(I_\delta(\omega) = 1 \mid \gamma \text{ is the root node}) \quad \text{for any } \delta \in S_\gamma. \tag{3.13}$$

Since this is true for any node δ in S_γ, we conclude that given E_γ , then the entire search performed by the stack algorithm in S_γ does not depend on the search performed by the stack algorithm outside S_γ before and/or after reaching node γ. As a result, it follows that given E_γ, then for any event E in $\mathcal{G}_\gamma$, the event $\{ I_E(\omega) = 0 \text{ or } 1 \}$ does not depend on the stack algorithm searching history outside S_γ for any $\omega \in E_\gamma$. As a consequence, we get our desired result.

□

As a result of the above theorem, we may estimate $E[X \mid E_\gamma]$ using multiple independent simulations, observing X (with γ being the root node) for each simulation, and then estimating $E[X \mid \gamma \text{ is the root node}]$ using the sample mean estimator.

If $M_{\eta_1} > M_{\beta_5}$ Then δ is in Z_5

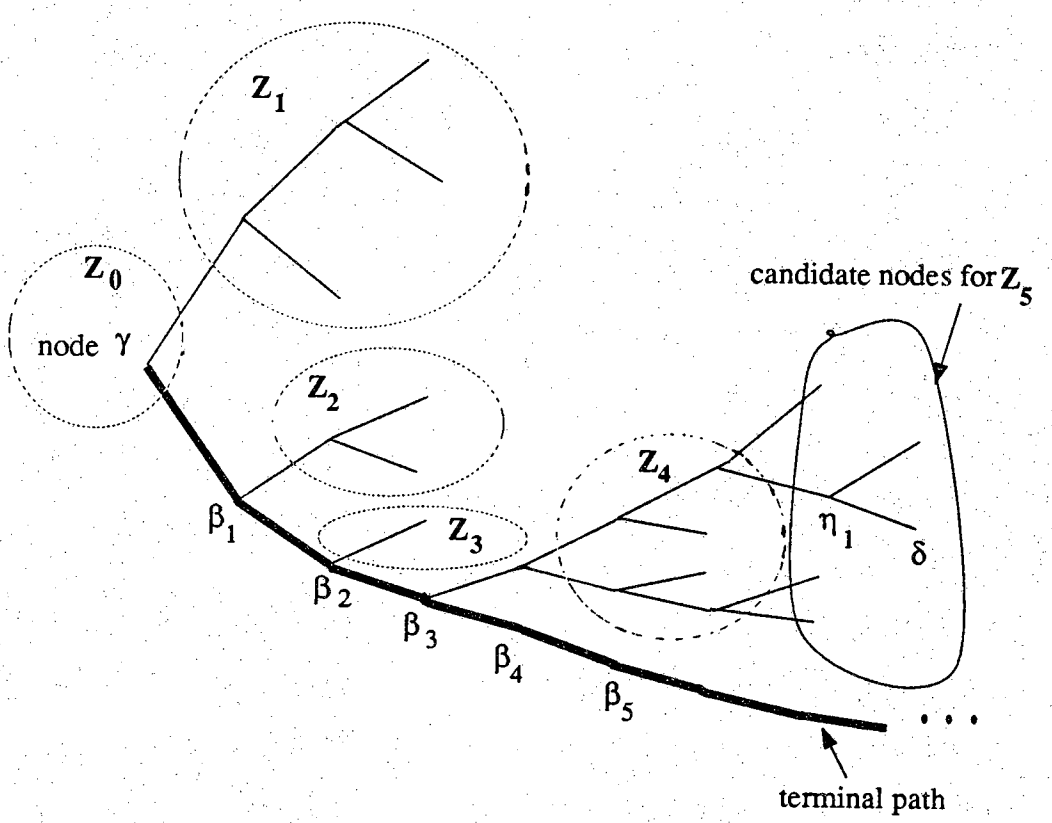


Figure 3.4: An example of the sets Z_i.

We conclude this section by noting two important issues. First, for practical reasons a *termination strategy* for the stack algorithm is always needed. For example, one might delete any node from the stack if its metric is less than the TOS node metric minus some threshold Δ. The second issue is the *"remerging phenomenon"* which occurs in practical applications and ultimately results in branching errors [57]. This phenomenon adds additional complexity to the simulation of the stack algorithm as it corresponds to incorrect paths in the tree which behave exactly like the correct path after the point of remerging. These issues will be addressed in full details in the next chapters.

3.5 The Modified Stack Algorithm Simulation

Recall that we are interested in applying importance sampling in order to estimate some key performance parameters that are associated with the stack algorithm. It turns out that in order to estimate most of these parameters, only the search performed by the stack algorithm in the j'th incorrect subtree $\mathcal{N}_j$ is required. For example, to estimate $E[C_j]$ or $P(C_j \geq M)$ for some $M \geq 1$, it is apparent that only the search performed by the stack algorithm in $\mathcal{N}_j$ is needed. Keeping this in mind, we conclude that any simulation scheme which modifies the stack algorithm so that only the incorrect nodes in $\mathcal{N}_j$ are extended will most likely be more efficient than any other simulation scheme which uses the stack algorithm. In other words, using a modified stack algorithm which operates exactly like the stack algorithm except that it

1) extends only the j'th incorrect subtree and
2) replaces every top-of-stack node which is on the correct path by only its direct descendent which is on the correct path,

will lead to a substantial improvement in speed and thus increases the efficiency of the importance sampling simulations. In the sequel, we will refer to such algorithm as the *modified stack algorithm simulation* (MSAS).

Because the search performed by the stack algorithm in $\mathcal{N}_j$ can be affected by the search performed by the stack algorithm in other incorrect subtrees, $\mathcal{N}_{j+1}$, $\mathcal{N}_{j+2}$, estimates obtained using the modified stack algorithm simulation might be incorrect. We shall see in Chapter 5 that the difference between the results obtained using the stack algorithm and the modified stack algorithm simulation is apparently insignificant. In other word, it does appear that the modified stack algorithm simulation gives estimates which are very close to the ones obtained when the stack algorithm is actually used.

CHAPTER 4
IMPORTANCE SAMPLING APPLIED TO SEQUENTIAL DECODING

4.1 Introduction

In 1955, Elias has introduced a general class of error control codes called *convolutional codes* as an alternative to *block codes [32]*. Shortly thereafter (1957), Wozencraft [33] introduced an efficient scheme for decoding convolutional codes which is called *sequential decoding*. Then in 1967, Viterbi [39] introduced an algorithm for decoding convolutional codes which has since become known as the *Viterbi Algorithm [52]*. This scheme, together with improved versions of sequential decoding led to the application of convolutional codes to practical communication channels such as satellite and deep-space communication channels [40], [48], [59].

In contrast to sequential decoding schemes, the Viterbi algorithm performs a *full maximum likelihood search*. This algorithm is known to be optimum in the sense that it minimizes the probability of error in decoding the entire transmitted sequence of information bits [57], [59]. In addition, it is more "robust" with respect to the model variations. The main difficulty with the Viterbi algorithm is that in practice arbitrary small error probabilities are not achievable. This is due to the fact that only small constraint lengths can be used because of the limitations on the decoder memory.

Another difficulty with the Viterbi algorithm is that its computational complexity grows exponentially with codes constraint length. Sequential decoding, on the other hand, is a very powerful technique for decoding convolutional codes which appears as a natural method of reducing the amount of computations per decoded information block by a trial-and-error rather than an exhaustive search. In contrast to the Viterbi algorithm, sequential decoding is essentially independent of the encoder memory, and hence arbitrary low error probabilities can be achieved provided operation under cutoff rate. Its major drawback is its inherent inability to deal effectively with severe noisy bursts which sometimes take large amounts of computations, and occasionally cause information to be lost or erased.

As stated earlier, sequential decoding was first proposed and analyzed in 1957 by Wozencraft [33] as a practical means of decoding convolutional codes. In 1963, Fano [36] introduced a new version of sequential decoding, subsequently referred to as the *Fano algorithm*. Various minor modifications of the Fano algorithm have been analyzed by Yudkin (1964), Wozencraft and Jacobs (1965), and Gallager (1968). A few years later, another version of sequential decoding, called the *stack algorithm* (also called the *Z-J algorithm*), was independently discovered by Zigangirov [37] and Jelinek [44]. In this Chapter, we will consider the stack algorithm exclusively. We should note, however, that most our results and conclusions can be applied to the other sequential decoding schemes. The reader is referred to [33-61] and references therein for more discussions about sequential decoding.

In this chapter, we apply *importance sampling* to the problem of simulating the sequential decoders decision process, in particular, ones that use the stack algorithm. We shall present three importance sampling techniques which we shall refer to as the *reference path method,* the *partitioning method,* and the *M-method*. The reference path

method and the M-method are ad hoc importance sampling techniques. However, we should note that the reference path method is based on the distance properties of the code being simulated. The partitioning method, on the other hand, is motivated by *large deviations theory* and an information theoretic ensemble averaging argument similar to the one used in [26]. It is noted that in all of the above techniques, we do not consider decision error probabilities. In other words, we assume that the decoder ultimately chooses the correct transmitted path. In Chapter 6, we shall present an other importance sampling technique which we will refer to as the *error event simulation method*. In contrast to the above importance sampling methods, the error event simulation method deals specifically with the problem of simulating the error events associated with stack algorithm sequential decoders. In particular, we shall use this method to estimate bit error rates for such decoders.

4.2 Channel Model and Convolutional Codes

Because of the inherent tree structure of convolutional codes, we shall start this section by briefly describing *tree codes*. However, before doing so, we will first describe our channel model.

4.2.1 Channel Model

Let us assume that we desire to communicate over a memoryless coding channel. At each time instant k, a channel symbol input u_k is transmitted over the channel to produce the channel output symbol V_k. Given u_k transmitted, V_k is a random quantity with conditional density[1] $f_k(v_k \mid u_k)$. Since the channel is memoryless, the sequence of output symbols $\mathbf{V} = (V_1, V_2, ..., V_n)$, is a sequence of independent random channel output symbols with joint density

1. We will use the term "density" to mean probability density function or probability mass function depending on whether the channel outputs are continuous or discrete.

$$\mathbf{f}(\mathbf{v}\,|\,\mathbf{u}) = \prod_{k=1}^{n} f_k(v_k\,|\,u_k) \tag{4.1}$$

for the transmission of n channel symbols. We should note that if $f_k(v_k\,|\,\mathbf{u}_k)$ does not depend on k, then the channel is also stationary. We shall see later that it is worthwhile to think of $f_k(v_k\,|\,\mathbf{u}_k)$ as a function of k.

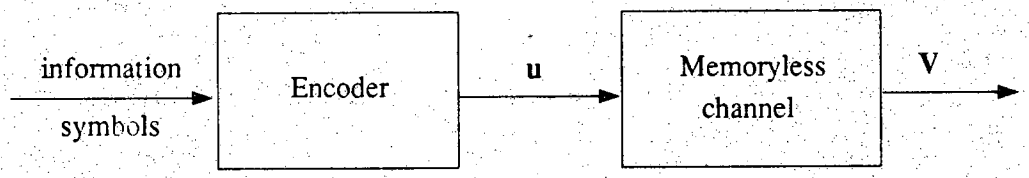


Figure 4.1: The channel model.

Throughout this chapter, we will consider convolutional codes which operates on the *binary symmetric channel* (BSC) and the *additive white gaussian noise* (AWGN) *Channel*. Assuming that the input alphabet and the output alphabet of the channel are both equal to {-1,1}. Then the BSC can be characterized as follows:

$$V_k = \begin{cases} u_k & \text{with probability } 1-\varepsilon \\ -u_k & \text{with probability } \varepsilon \end{cases} \tag{4.2}$$

ε *is called the crossover probability*. The BSC is shown in Figure 4.2

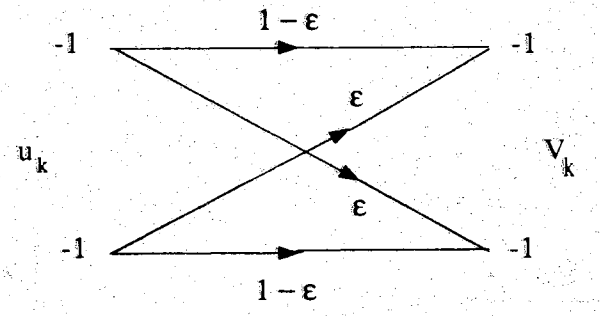


Figure 4.2: Binary symmetric channel.

In a similar fashion, we can define the AWGN channel to be a continuous channel for which the channel output symbol density is given by

$$f_k(v_k \mid u_k) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(v_k - u_k)^2}{2\sigma^2}} \tag{4.3}$$

where

$$\sigma^2 = \frac{N_0}{2}$$

and N_0 is the one sided power spectral density of the noise which is assumed to be white. The AWGN channel is represented in Figure 4.3 with $\{N_k\}$ k= 1,2,.. being a Gaussian random sequence with zero mean and variance σ^2.

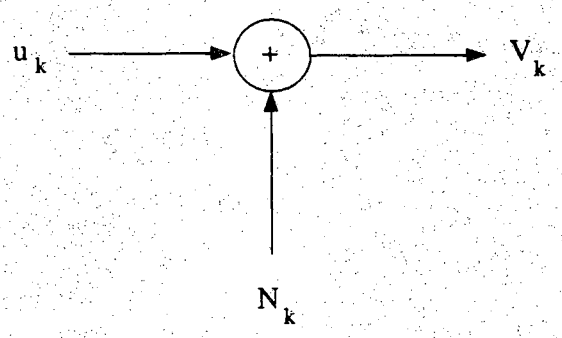


Figure 4.3: Additive white gaussian noise channel.

4.2.2 Tree Codes

For brevity, we shall restrict our attention to *binary tree codes* of *rate* R = 1/2. Generalizations for rate b/n codes is straightforward.

A binary tree code of Rate R = 1/2 is formed by assigning two channel input symbols to each branch of a rooted *binary tree*. Depending on whether the information bit symbol is 0 or 1, the encoder follows the upper or lower branch and transmits through the channel the code sequence associated with the branch which was followed. In this way a sequence of information bits traces a path through the tree and the code sequence corresponding to that path is then transmitted. Thus in Figure 4.4, the information bit sequence 0100 determines the path indicated by the bold line and causes the code sequence 00111011 to be transmitted through the channel.

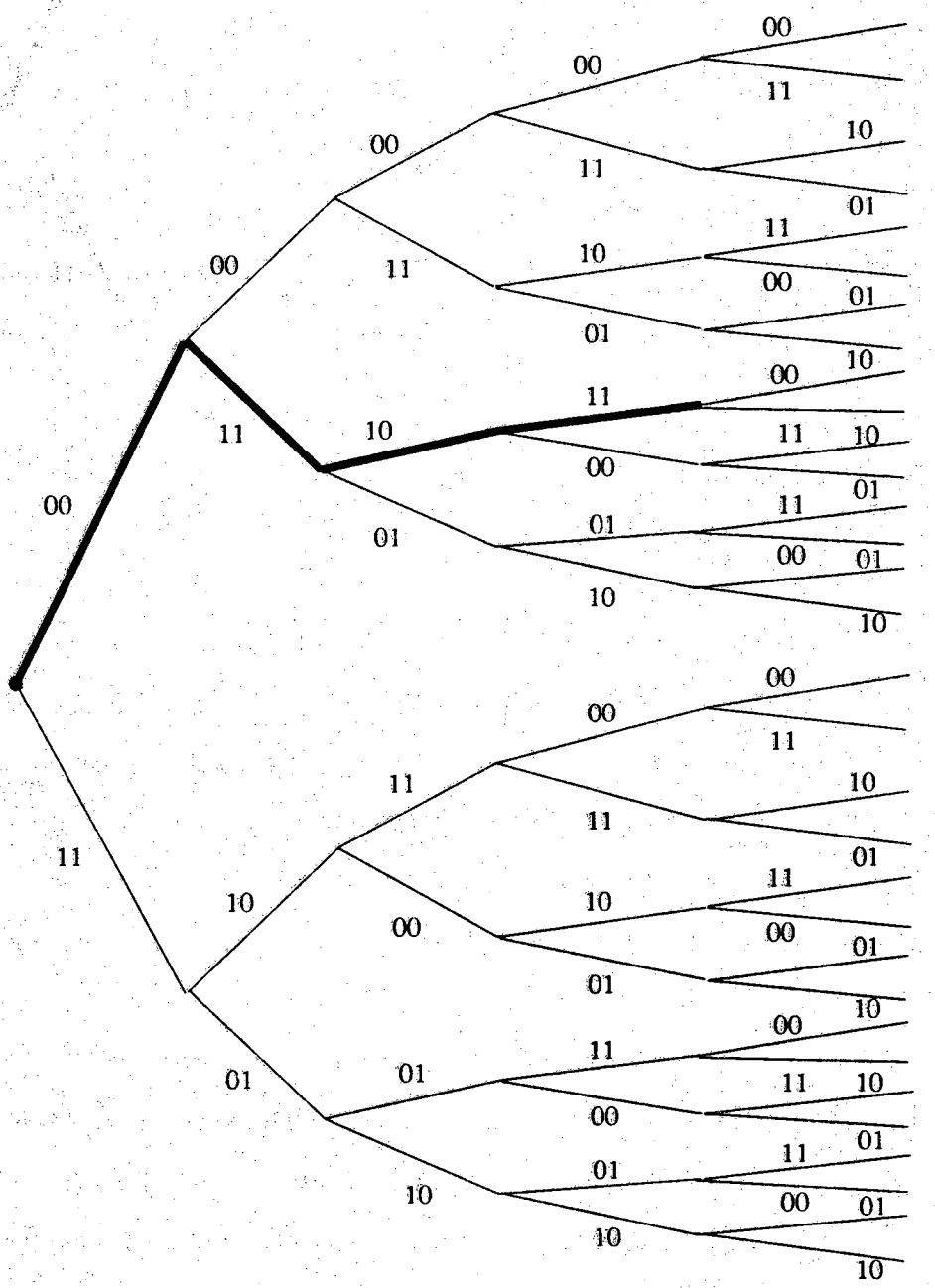


Figure 4.4: An example of a binary tree code of rate 1/2.

For a given sequence of information bits, let $\mathbf{u} = (u_1, u_2, ...)$ denotes the sequence of encoder output symbols. Given the sequence of channel output symbols $\mathbf{V} = (V_1, V_2, ...)$, the job of the decoder is to determine the information bit sequence which was most likely to have been transmitted, or equivalently, its corresponding path on the tree. Note that because the code rate is 1/2, it follows that the encoder and channel output symbols u_k and V_k are symbols which consist of 2 bits. That is, $V_k = (V_{k1}, V_{k2})$ and $u_k = (u_{k1}, u_{k2})$.

For each branch on the tree, the decoder computes a *branch metric* $\mathbf{m}(u_k, V_k) = \sum_{i=1}^{2} m(u_{ki}, V_{ki})$ which is an indicator of the likelihood that u_k was indeed the channel input symbol which produced the channel output symbol V_k. The *Fano metric* is a maximum likelihood like metric which is commonly used in sequential decoding. The Fano metric is defined as

$$m(u_{ki}, v_{ki}) = \log[\frac{f_{ki}(v_{ki} \mid u_{ki})}{f_v(v_{ki})}] - R \quad i = 1,2. \tag{4.4}$$

In the above equation, R is denotes the rate of the code, $f_{ki}(v_{ki} \mid u_{ki})$ is the channel transition density, and $f_v(v_{ki})$ the channel output symbol density. The metric of (4.4) was first introduced by Fano on intuitive grounds [34], and hence the name Fano metric. In 1972, Massey [51] has given analytical justification for using the Fano metric in sequential decoding. It is noted that the average behavior of the Fano metric is to increase along the correct path and to decrease otherwise as long as $R < R_0$ where R_0 is called the *computational cutoff rate* of the channel [57].

4.2.3 Convolutional Codes

Convolutional codes are tree codes that are generated by passing the information bit sequence to be transmitted through a linear finite state shift register. At each operation of the encoder, b *information bits* are shifted into the shift register which contains a total of Kb bits. We shall refer to these b bits as *information symbols*. Hence, the shift register contains K information symbols. The output of the encoder is a block of n bits which we refer to as a *code symbol*. Consequently the code rate is defined as

$$R = \frac{b}{n} \quad \text{bits per output symbol} \tag{4.5}$$

For a convolutional encoder of rate b/n, the shift register consists of Kb stages and n linear algebraic function generators, or simply, n modulo-2 adders which are often implemented as exclusive-or gates. As in block codes, convolutional codes can be described by giving their generator matrices [57]. An equivalent representation which is commonly used consists of specifying a set of n vectors $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_n$, one for each of the n modulo-2 adders is used. Each vector has Kb dimensions. That is, for each i=1,..n, $\mathbf{g}_i = [g_{i1}, g_{i2}, \ldots, g_{ikb}]$ (with $g_{ij} = 0$ or 1, for i= 1,...,n and j= 1,...,Kb.) A 1 in the jth position of the vector $\mathbf{g}_i$ indicates that the corresponding stage in the shift register is connected to the modulo-2 adder and a 0 in a given position indicates that no connection exists between that stage and the modulo-2 adder. For example, the generators for the convolutional codes shown in Figure 4.5 are

$$g_1 = [101]$$

and

$$g_2 = [111]$$

In octal form, these vectors are (5,7)

The complexity of the code is determined by its *constraint length* Kb which is the total number of bits used to compute the output code symbol. In this context K-1 is simply the number of information bits stored in the encoder shift register, not counting the most recent information bit input. In a similar fashion, the *encoder state* is defined to be the last K-1 information symbols in the shift register with the last most recent bit being the last bit in the state. Since there are (K-1)b bits which determine the state, the total number of states is $2^{(K-1)b}$. For each state at time k there are 2^b possible predecessor states, one corresponding to each of the 2^b information symbols that could have been shifted out of the shift register during the state transition from time k to time $k + 1$. The encoder output at time k is thus a function of the encoder states at time k and $k - 1$.

Convolutional codes are often described using three alternative methods. These are the *tree diagram,* the *trellis diagram,* and the *state diagram* [57]. To demonstrate the use and insight provided by such diagrams we consider the convolutional encoder of Figure 4.5.

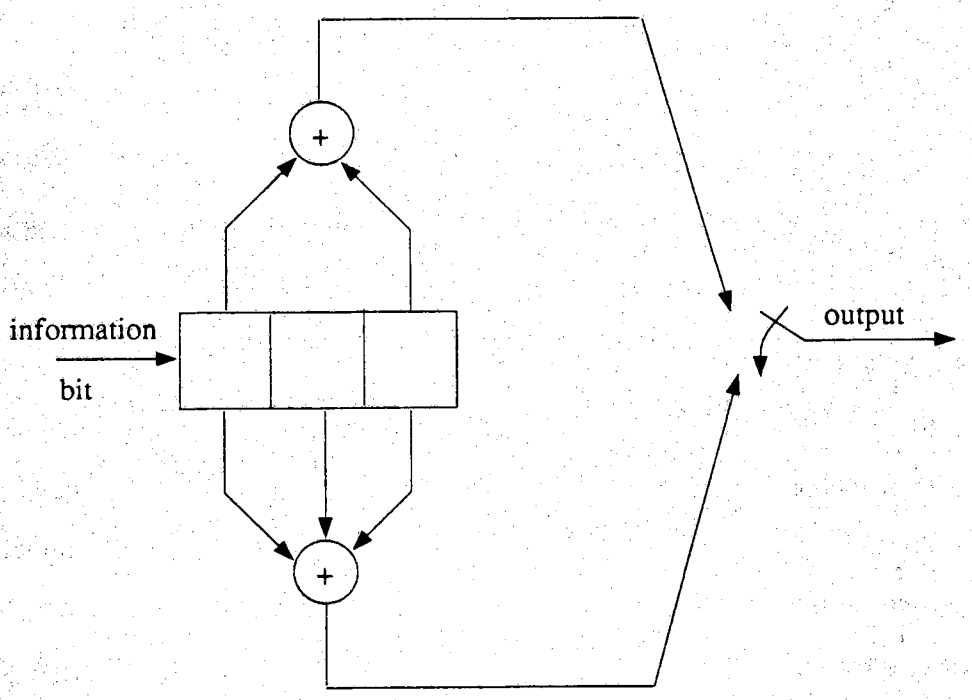


Figure 4.5: A rate 1/2 and constraint length 3 convolutional encoder.

The tree code representation (or tree diagram) for the encoder of Figure 4.5 is shown in Figure 4.4. Assuming that the encoder is in the all-zero state initially, the diagram shows that, if the first input information bit is a 1, the code symbol is 11 and, if the first bit is a 0, the output sequence is 00. In other words, an input 1 specifies the lower branch, and an input 0 specifies the upper one. For example, the information bit sequence 0100 traces the thick path shown in Figure 4.4 and produces the code symbol sequence which is indicated along the branches traversed: 00, 11, 10, 11.

Close observation of the tree in Figure 4.4 reveals that after the first three branches, the structure repeats itself. This behavior is obvious from examination of the

encoder and is consistent with the fact that the constraint length is 3. When the third information bit is shifted into the encoder, the first information bit (i.e., the bit in the last stage of the shift register) is shifted out at the right and thereafter no longer affects the code symbols. Thus we may say that the 2-bit code symbol is determined by the information bit and the four possible states of the shift register: 00, 01, 10, and 11. This leads to redrawing the tree diagram as shown in Figure 4.6. This new representation, which is more compact, is called the *trellis diagram.* In drawing this diagram, we use the convention that a dotted line denotes the output generated by the information bit 1 and a solid line denotes the output generated by the information bit 0.

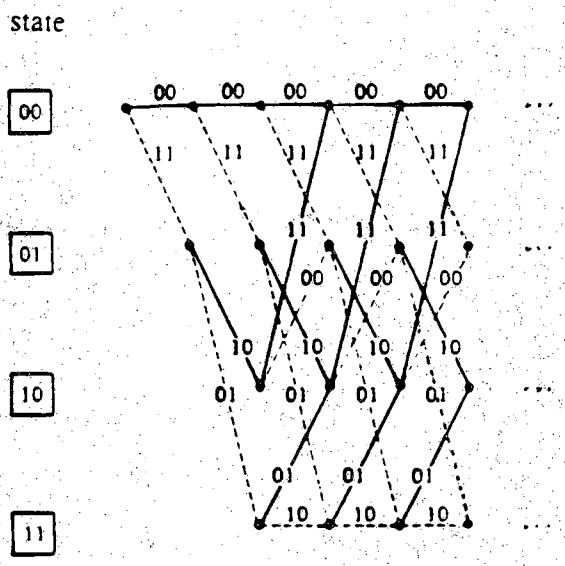


Figure 4.6: Trellis diagram for the encoder of Figure 4.5.

Since the output is determined by the information bit and the encoder states and because of the repetitive structure of the trellis diagram, an even more compact representation than the trellis diagram has been suggested. This new representation is called the state diagram. The state diagram is simply a graph of all possible states of the encoder and the possible transition from one state to another. For example, the state digram of the convolutional encoder shown in Figure 4.5 is illustrated in Figure 4.7. Again, in drawing this diagram, we use the convention that a dotted line denotes the output generated by the information bit 1 and a solid line denotes the output generated by the information bit 0.

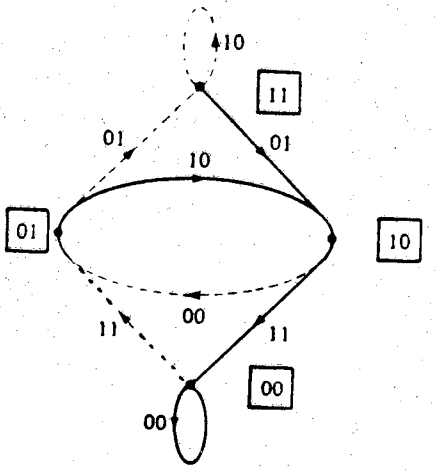


Figure 4.7: State diagram for the encoder of Figure 4.5.

4.3 Error Events and Remerging

Close observation of the trellis diagram reveals that it is possible for incorrect paths in the tree to merge with the correct path. This is known as the *"remerging phenomenon"* [57]. These remerged paths correspond to incorrect paths in the tree which behave exactly like the correct path after the point of remerging. That is, the remerged paths are incorrect paths in the tree which briefly exhibit a negative metric drift, and then begin to exactly parallel the behavior of the correct path after the point of remerging. On the trellis diagram however, a remerged path corresponds to a trellis path which briefly diverges from the correct trellis path. The divergent path of the incorrect path is called an *error event*. Specifically, if we let $\mathbf{u}^f = (u_1^f, u_2^f, ..)$ denotes the final path hypothesized by the sequential decoder, and $\mathbf{u}^c = (u_1^c, u_2^c, ..)$ denotes the correct (that is, the transmitted) path, then decoding errors occur when $\mathbf{u}^f$ diverges from $\mathbf{u}^c$. More precisely, an error event is a partial sequence of $\mathbf{u}^f$ which begins at a correct path node, ends at a correct path node, and has no correct path nodes in between the correct beginning and ending nodes. The *error event length* is the number of branches in the error event. For example, Figure 4.8 shows that $\mathbf{u}^f$ coincides with the correct path $\mathbf{u}^c$ up to some depth m in the trellis, branches off at depth m, and then remerges with $\mathbf{u}^c$ at depth m + 3. In this case, $\mathbf{u}^f$ contains an error event of length 3.

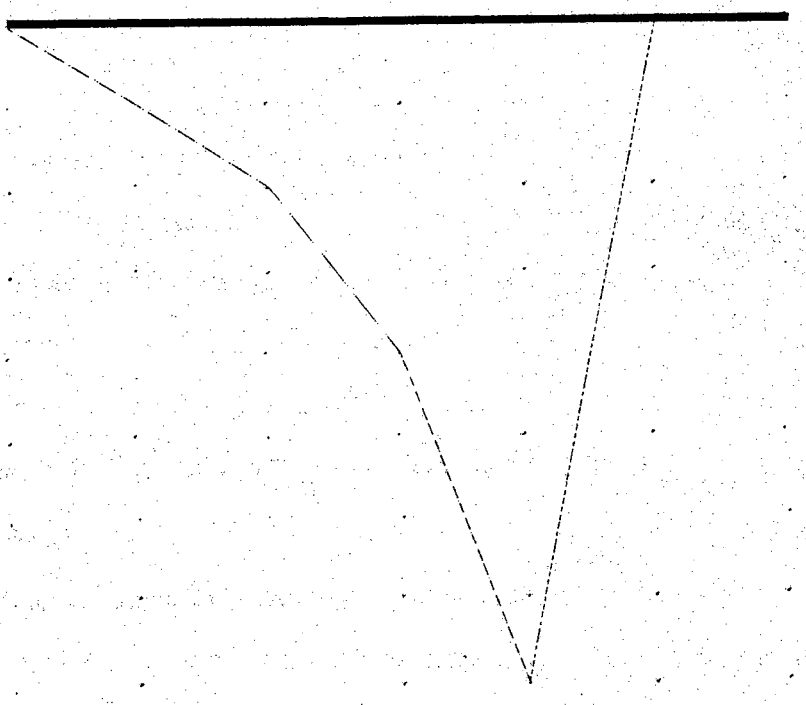


Figure 4.8: An illustration of error events and the remerging phenomenon. The bold line is the correct path and the dashed line is a remerged path.

4.4 The Reference Path Method

In this section, we propose a simulation model design which directly exploits the distance information of convolutional codes. Our goal is to apply this scheme to simulate (stack algorithm) sequential decoders for convolutional codes.

Let $\mathbf{u}^c = (u_1^c, u_2^c, \ldots)$ be the encoder output sequence associated with the correct path. That is, the transmitted tree path. Likewise, let $\mathbf{u}_N^i = (u_1^i, u_2^i, \ldots, u_N^i)$ be the encoder output sequence associated with any incorrect path of depth N that diverges from the correct path at the root node. Next suppose that $\mathbf{u}_N^i$ differs from $\mathbf{u}^c$ in exactly d_N positions. The number d_N is then said to be the *Hamming distance* [57] between $\mathbf{u}^c$

and $\mathbf{u}_N^i$. The reason for our interest in this distance is that the "important" branching error events are precisely those with low to moderate distance d_N. In other words, it is more likely for the stack algorithm to search incorrect paths with low distances than those with large distances even if the lengths of the paths are quite long. In fact, the distance of a path is a direct indicator of its likelihood of being searched. The path length for the most part is irrelevant, except to the extent that long paths are more likely to have higher distances. To properly understand this consider the following example.

Suppose that the channel is a binary symmetric channel with crossover probability ε. Furthermore, assume that the Fano metric is used. In this case, the path metric of the incorrect path associated with $\mathbf{u}_N^i$ is given by

$$S_N^i = \log[\frac{\mathbf{f(v|u^c)}}{\mathbf{f_v(v)}}] - n\,N\,R \tag{4.6}$$

where R is the rate of the code (b/n,) $\mathbf{f(v|u^c)}$ is the conditional probability of receiving $\mathbf{V}=(V_1, V_2, \ldots, V_N)$ given that $\mathbf{u^c}$ was transmitted and $\mathbf{f_v(v)}$ is the probability of receiving the sequence $\mathbf{V}$. A straightforward computation indicates that

$$\mathbf{f(v|u^c)} = \varepsilon^{d_N}\,(1-\varepsilon)^{nN-d_N} \tag{4.7}$$

and

$$\mathbf{f(v)} = \frac{1}{2^{nN}} \tag{4.8}$$

Consequently, the path metric in (4.6) can then be written as

$$S_N^i = -a\,d_N + b\,N \tag{4.9}$$

where

$$a \triangleq \log(\frac{1-\varepsilon}{\varepsilon})$$

and

$$b \triangleq \log(2(1-\varepsilon))$$

Without loss of generality, we may assume that $0 < \varepsilon < 1/2$ (for if this is not the case, we can make it such by just interchanging the indices on v_1 and v_2 in Figure 4.2.) In this case, we have $a > 0$, and thus from (4.9) we conclude that it is more likely for the stack algorithm to search incorrect paths with low distances than those with large distances. In fact, close observation of (4.9) indicates that any incorrect path will be rejected by the stack algorithm if its distance from the received sequence **V** is sufficiently large.

To further illustrate the relationship between the distance properties of the code and the search performed by the stack algorithm, consider Figures 4.9-4.11. These figures show the actual search performed by the stack algorithm in the 0th incorrect subtree $\mathcal{N}_0$ (the all zero path is assumed to be the transmitted path.) In this case, ordinary Monte Carlo was used to simulate the stack algorithm decoder for a rate 1/2 and constraint length 14 convolutional code operating on an AWGN channel with variance equal to .36.

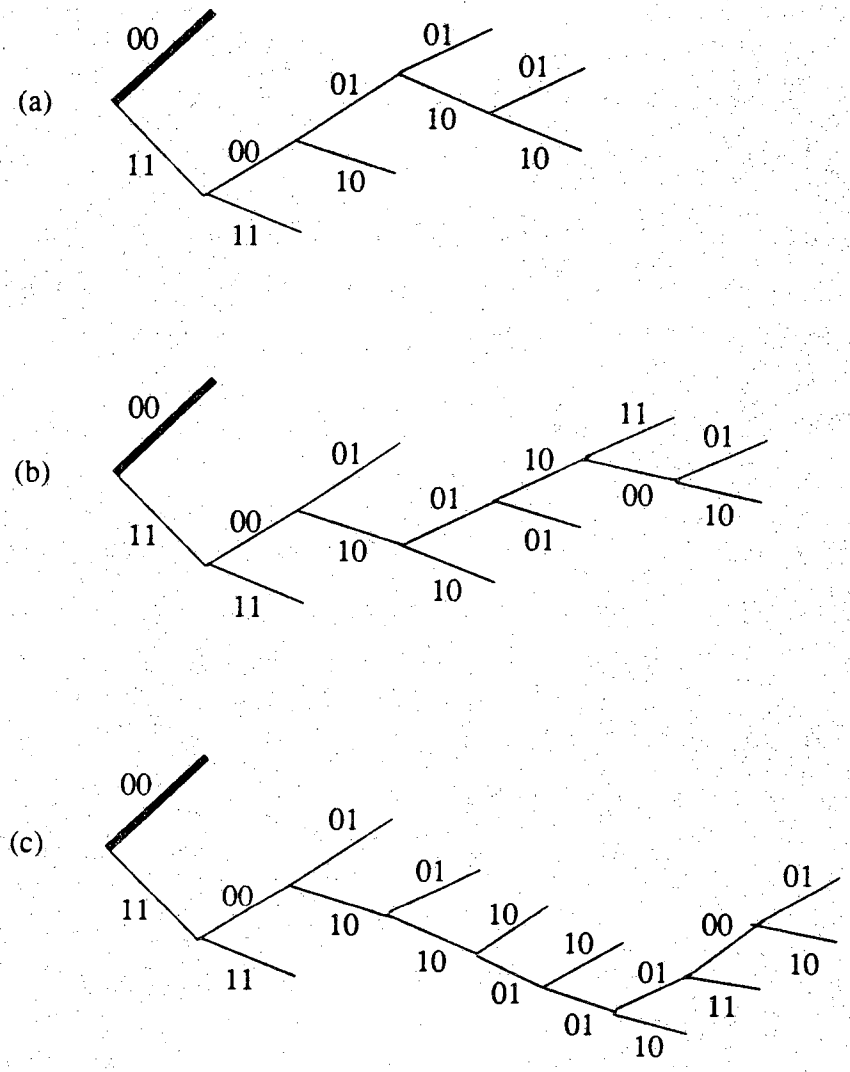


Figure 4.9: The stack algorithm searching history in $\mathcal{N}_0$. (a) $C_0 = 9$, (b) $C_0 = 13$, and (c) $C_0 = 17$.

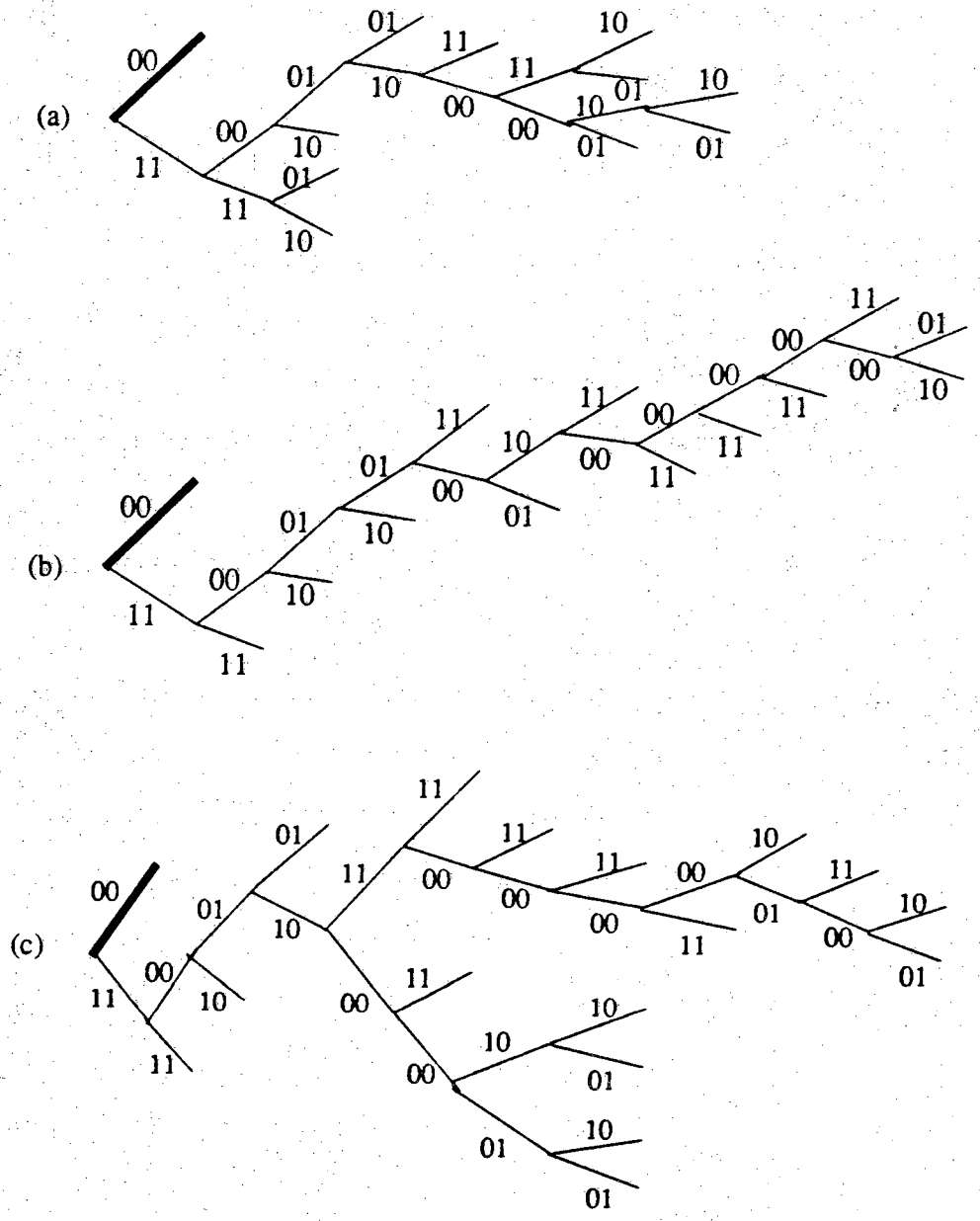
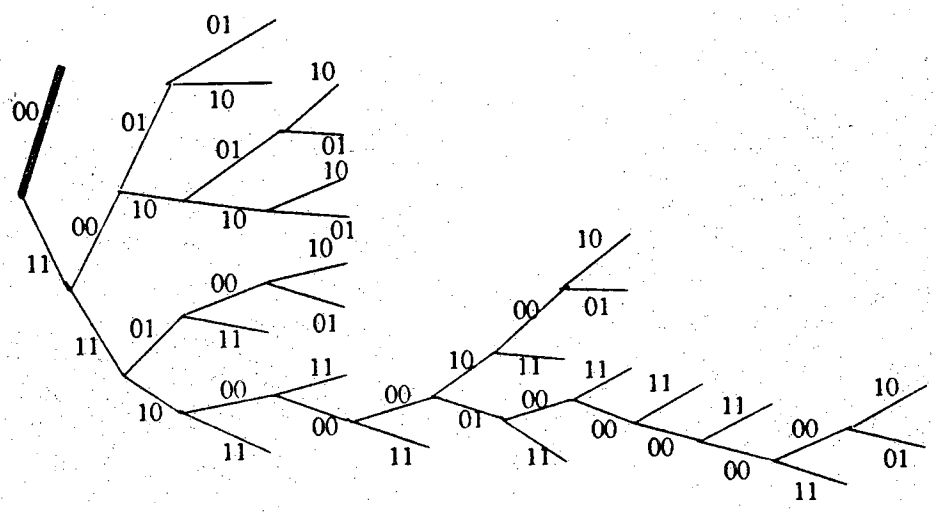
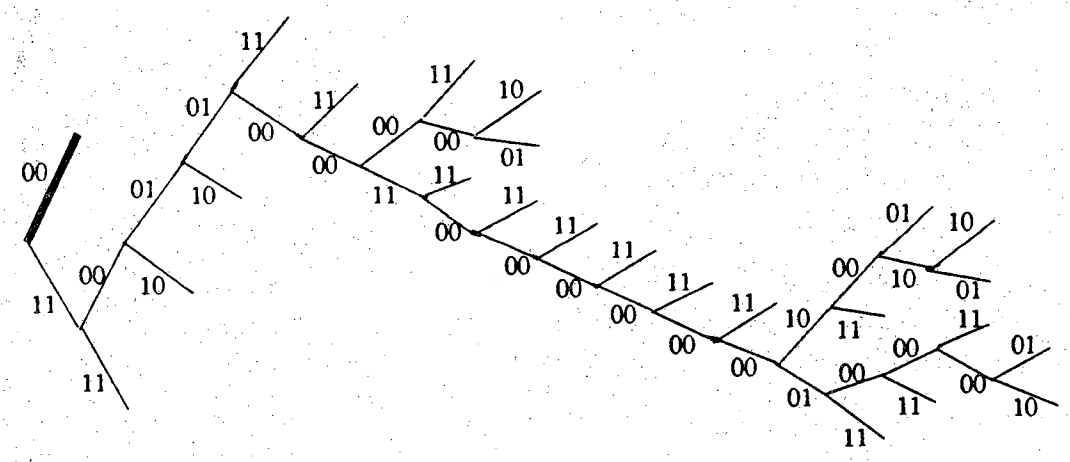


Figure 4.10: The stack algorithm searching history in $\mathcal{N}_0$. (a) $C_0 = 19$, (b) $C_0 = 23$, and (c) $C_0 = 31$.



(a)



(b)

Figure 4.11: The stack algorithm searching history in $\mathcal{N}_0$. (a) $C_0 = 43$, and (b) $C_0 = 45$.

Figures 4.9-4.11 indicate that the search performed by the stack algorithm in an incorrect subtree is done in such away that 1) if the algorithm searches an incorrect path for some depth in the tree, then this path is most likely an incorrect path with a low distance from the correct path, and 2) most of the incorrect paths extended by the stack algorithm are short paths (i.e. with few branches) which emanates from the low distance paths. Thus we can see that the "important" branching error events are precisely those associated with incorrect paths with low to moderate distances from the correct path. Consequently, a good importance sampling scheme must be designed in such a way that the relative frequency of these events is increased. Close observation of Figure 4.9-4.11 indicates that this can be done by "forcing" the stack algorithm to follow a *reference path* with a low distance, and hence the name *reference path method.*

Let $\mathbf{u}_N^r = (u_1^r, u_2^r, ..., u_N^r)$ be the encoder output sequence associated with a *reference path* of depth N. Such a path is chosen in such a way that its hamming distance from the correct path is minimized over all incorrect paths of depth N. These reference paths can be found by an exhaustive search for various depths N.

Now for a given reference path, our *basic principle* is to design an importance sampling channel model which will tend to "trick" the stack algorithm into decoding the reference path instead of the correct path. Specifically, each simulation run will produce a randomly generated channel output sequence $\mathbf{V} = (V_1, V_2, ...)$ to produce a sequence of branching decisions which are biased toward the reference path. That is, branching decisions which attempt to follow the reference path instead of the correct transmitted path. This importance sampling channel model however must be chosen in such a way that the stack algorithm will ultimately make a correct decision at depth j+1, even though the data is biased towards producing the reference path (note that we

do not need to do this when we are interested in decision errors.) This can be done, for instance, by switching the importance sampling simulation channel model to the original channel model at depth N. As an example, consider the binary symmetric channel with *crossover probability* ε. Then the following non-stationary model is an example of a reference path simulation model. This IS simulation model is characterized by a time varying crossover probability

$$\varepsilon_k^* = \begin{cases} 1/2 & \text{if } u_k^c \neq u_k^I \\ \varepsilon & \text{if } u_k^c = u_k^I \end{cases}$$

for $k \leq N$. For $k > N$ we use $\varepsilon_k^* = \varepsilon$. It should be clear that with roughly probability 1/2 the reference path will be examined up to depth N.

4.5 The Partitioning Method Analysis

The reference path method is an ad hoc importance sampling scheme which leads to substantial efficiency in comparison to ordinary Monte Carlo. In this section, we shall discuss another importance sampling scheme which is inspired by the asymptotics of large deviations theory and an "ensemble average" variance-Chernoff bound argument (similar to that used in [26].)

We begin by developing few definitions which are needed for the discussion to follow.

4.5.1 Preliminaries and Definitions

Consider the memoryless coding channel discussed previously in section 4.2.1. Next let $\{U_k^c\}$, k= 1,2,.. denote the encoder output sequence associated with the correct path. Likewise, let $\{U_k^i\}$, k= 1,2,.. denote the encoder output sequence corresponding to an incorrect path. In the analysis of the partitioning method, $\{U_k^c\}$ and $\{U_k^i\}$ are both assumed to be independent and i.i.d. random processes with the same univariate density q(.). Next let $\{ X_k : k= 1,2,... \}$ be an i.i.d. sequence of random variables with univariate density g(x) such that

$$X_k \triangleq (U_k^c, V_k)$$

where V_k is the channel output symbol. Consequently,

$$g(x_k) = f_k(v_k \mid u_k^c)\, q(u_k^c) \tag{4.10}$$

Now consider the stack algorithm and suppose that the correct node j has been hypothesized. Next let Z_k^c and Z_k^i denote the k+j'th branch metric on the correct path and on any incorrect path which diverges from the correct path at node j. Likewise, let

$$S_n^c = \sum_{k=1}^{n} Z_k^c \tag{4.11}$$

and

$$S_n^i = \sum_{k=1}^{n} Z_k^i \tag{4.12}$$

denote the correct and incorrect node metric processes respectively.

We shall also define

$$N_\gamma^i \triangleq \inf \{ n : S_n^i \le \gamma \} \tag{4.13}$$

and

$$N_\gamma^c \triangleq \inf \{ n : S_n^c \leq \gamma \}. \tag{4.14}$$

In the sequel we shall assume the following hypothesis:

Hypothesis H:

The branch metric processes are bounded. That is, there exists some $\Delta < \infty$ such that $|Z_k^c| \leq \Delta$ and $|Z_k^i| \leq \Delta$.

Define

$$\mathcal{F}^X \triangleq \sigma(X_0, X_1, \ldots). \tag{4.15}$$

Then note that because $X_k \triangleq (U_k^c, V_k)$, it follows that given the entire history of $\{X_k\}$, $\mathcal{F}^X$, $\{Z_k^c\}$ and $\{Z_k^i\}$ are conditionally independent[2]. In other words, we have

$$\mathcal{L}(dz_k^c, dz_k^i \mid x_k) = F_c(dz_k^c \mid x_k)\, F_i(dz_k^i \mid x_k) \tag{4.16}$$

where $F_c(.|.)$ and $F_i(.|)$ are respectively the conditional distributions of Z_k^c and Z_k^i given x_k.

It is now convenient to define the following exponential transformation:

$$F_i^{(\alpha^i)}(dz_k^i \mid x_k) = e^{\alpha^i Z_k^i - \Lambda^i(\alpha^i \mid x_k)}\, F_i(dz_k^i \mid x_k) \tag{4.17}$$

where

$$\Lambda^i(\alpha^i \mid x_k) \triangleq \ln \left[E[\, e^{\alpha^i Z_k^i} \,] \right] \tag{4.18}$$

$$= \ln \left[\int e^{\alpha^i z_k^i}\, F_i(dz_k^c \mid x_k) \right].$$

Likewise, we can also define

2. This is true because Z_k^c and Z_k^i depend on only (U_k^c, V_k) and (U_k^i, V_k), respectively.

$$\Lambda^c(\alpha^c \mid x_k) \triangleq \ln\left[E[\, e^{\alpha^c Z_k^c}\,]\right] \tag{4.19}$$

$$= \ln\left[\int e^{\alpha^c z_k^c} F_c(dz_k^c \mid x_k)\right]$$

We should note, however, that since $X_k = (U_k^c, V_k)$ and since Z_k^c is only a function of U_k^c and V_K, it follows that

$$\Lambda^c(\alpha^c \mid x_k) = \alpha^c Z_k^c. \tag{4.20}$$

$\Lambda^c(.|.)$ and $\Lambda^i(.|.)$ are respectively the conditional *log-moment generating functions* associated with the correct path and incorrect path metric processes. It turns out that $\Lambda^c(.|.)$ and $\Lambda^i(.|.)$ have very nice smoothness properties. In particular they are analytic and strictly convex on **R**.

4.5.2 The Partitioning Method Results

Consider the stack algorithm, and suppose that the correct node j has been hypothesized. Next consider a fixed incorrect node δ, which has depth j+n and is on an incorrect path diverging from the correct path at depth j. δ is a candidate node for the incorrect subset $\mathcal{X}_j$. The kernel of our analysis is to consider the probability that δ is hypothesized by the stack algorithm, that is, $\mathcal{P}(\delta \in \mathcal{X}_j)$. The analysis is then simplified by comparing the correct path metric process with the metric process along a single fixed incorrect path emanating from node j. Specifically, we compare an incorrect path in the j'th incorrect subtree to the correct path.

Let $\underline{f}$ denotes the joint density of $\mathbf{U}^c = (U_1^c, U_2^c, ..., U_n^c)$, $\mathbf{U}^i = (U_1^i, U_2^i, ..., U_n^i)$, and $\mathbf{V} = (V_1, V_2, ..., V_n)$. Then

$$\underline{f}(\mathbf{u}^c, \mathbf{u}^i, \mathbf{v}) = \mathbf{f}(\mathbf{v} \mid \mathbf{u}^c)\, \mathbf{q}(\mathbf{u}^c)\, \mathbf{q}(\mathbf{u}^i) \tag{4.21}$$

where

$$\mathbf{q}(\mathbf{u}) = \prod_{k=1}^{n} q(u_k) \tag{4.22}$$

is the channel input symbols joint density.

It is now convenient to define

$$\begin{aligned} P_\delta &\triangleq \mathcal{P}(\,\delta \text{ hypothesized}\,) \\ &= \mathcal{P}(\,\delta \in \mathcal{X}_j\,) \\ &= E[\,I_\delta\,] \end{aligned} \tag{4.23}$$

where

I_δ = Indicator random variable for node δ.

The basic idea behind the partitioning method is to consider the problem of estimating P_δ using importance sampling. Thus, if we let $\hat{P}_\delta$ be the *importance sampling estimator* for P_δ. Then

$$\hat{P}_\delta = \frac{1}{L} \sum_{\ell=1}^{L} I_\delta(\,\mathbf{U}^c,\mathbf{U}^i,\mathbf{V}^{(\ell)})\, \underline{w}(\,\mathbf{U}^c,\mathbf{U}^i,\mathbf{V}^{(\ell)}\,) \tag{4.24}$$

where

$$\underline{w}(\mathbf{u}^c,\mathbf{u}^i,\mathbf{v}) \triangleq \frac{\underline{f}(\mathbf{u}^c,\mathbf{u}^i,\mathbf{v})}{\underline{f}^*(\mathbf{u}^c,\mathbf{u}^i,\mathbf{v})} \tag{4.25}$$

$$\triangleq \frac{f(\mathbf{v}\,|\,\mathbf{u}^c)\,q(\mathbf{u}^c)\,q(\mathbf{u}^i)}{f^*(\mathbf{v}\,|\,\mathbf{u}^c)\,q(\mathbf{u}^c)\,q(\mathbf{u}^i)}$$

$$= \frac{f(\mathbf{v}\,|\,\mathbf{u}^c)}{f^*(\mathbf{v}\,|\,\mathbf{u}^c)}$$

$$= \prod_{k=1}^{n} \frac{f_k(v_k\,|\,u_k^c)}{f_k^*(v_k\,|\,u_k^c)}. \tag{4.26}$$

The likelihood function (4.25) is the *importance sampling weight,* and the joint density $\underline{f}^*$ is the *importance sampling simulation joint density* from which the random samples $V^{(1)}, V^{(2)}, \ldots, V^{(L)}$ are generated.

A straightforward computation indicates that the weight function specified by (4.25) does indeed produce an unbiased estimator. That is,

$$E^*[\hat{P}_\delta] = \frac{1}{L}\sum_{\ell=1}^{L}\sum_{\mathbf{u}^c}\sum_{\mathbf{u}^i}\int\!\!\ldots\!\int I_\delta(\mathbf{u}^c,\mathbf{u}^i,\mathbf{v})\,\underline{w}(\mathbf{u}^c,\mathbf{u}^i,\mathbf{v}) \tag{4.27}$$

$$\times f^*(\mathbf{v}\,|\,\mathbf{u}^c)\,d\mathbf{v}\,q(\mathbf{u}^i)\,q(\mathbf{u}^c)$$

$$= \frac{1}{L}\sum_{\ell=1}^{L} E[I_\delta(\mathbf{U}^c,\mathbf{U}^i,\mathbf{V})]$$

$$= P_\delta.$$

A similar computation indicates that the variance of $\hat{P}_\delta$ is given by

$$\mathrm{var}^*[\hat{P}_\delta] = \frac{1}{L}\left[\eta_\delta - P_\delta{}^2\right] \tag{4.28}$$

where

$$\eta_\delta = E^*[\,I_\delta(\mathbf{U}^c,\mathbf{U}^i,\mathbf{V})\,(\underline{w}(\mathbf{U}^c,\mathbf{U}^i,\mathbf{V}))^2\,] \tag{4.29}$$

Notice that the impact of the choice of the IS distribution is completely represented by the functional η_δ. Consequently, our objective is to minimize η_δ.

Now from part (ii) of Lemma 3.1, it is apparent that

$$\mathcal{P}(\delta \text{ hypothesized }) = \mathcal{P}(\delta \in \mathcal{X}_j) \leq \mathcal{P}(N_\Gamma^i > n). \tag{4.30}$$

Recall that Γ is the correct path metric minimum defined by (3.3), and note that N_Γ^i is the point where the incorrect path metric process crosses Γ (see Figure 3.3.) In a recent work [61], we have shown that asymptotically (under certain conditions,) the probabilities $\mathcal{P}(N_\Gamma^i > n)$ decay exponentially. By letting r be the rate of decrease of these probabilities and ρ be the exponential growth rate of the tree (which is ln(b) when the tree has b branches per node,) then it is apparent that the critical point which we have called the point of computational cutoff (recall section 3.2.1) corresponds to the case when $\rho = r$. Thus simulating the stack algorithm using a simulation data which is generated from an importance sampling distribution that makes $r \leq \rho$ will make the operating condition of the stack algorithm very noisy and hence, ensures a high percentage of error events. This last observation together with the fact that one of the key performance criterion of the stack algorithm, namely the average number of metric computations per correct decision, is directly related to P_δ, led us to consider the problem of estimating P_δ. Indeed, if we let $\mathcal{N}_{j,n}$ be the set of candidate nodes for the incorrect subset $\mathcal{X}_j$ at depth j+n, then

$$C_j = \sum_{n=1}^{\infty} \sum_{\delta \in \mathcal{N}_{j,n}} I_\delta$$

and consequently,

$$E[C_j] = \sum_{n=1}^{\infty} \sum_{\delta \in \mathcal{N}_{j,n}} P_\delta. \tag{4.31}$$

Therefore, it follows that the problem of estimating $E[C_j]$ is equivalent to the problem of estimating P_δ.

Now notice that (4.30) implies that

$$\eta_\delta \le \overline{\eta_\delta} \tag{4.32}$$

where

$$\overline{\eta_\delta} = E^*[\, I_{\{N_\Gamma^i > n\}}\, (\mathbf{U}^c, \mathbf{U}^i, \mathbf{V})\, (\underline{w}(\mathbf{U}^c, \mathbf{U}^i, \mathbf{V}))^2\,] \tag{4.33}$$

$$= \sum_{\mathbf{u}^c} \sum_{\mathbf{u}^i} \int \cdots \int I_\delta(\mathbf{u}^c, \mathbf{u}^i, \mathbf{v}) \left[\prod_{k=1}^{n} \left(\frac{f_k(v_k \mid u_k^c)}{f_k^*(v_k \mid u_k^c)} \right)^2 f_k^*(v_k \mid u_k^c) \right] d\mathbf{v}\, q(\mathbf{u}^i)\, q(\mathbf{u}^c)$$

and $I_{\{N_\Gamma^i > n\}}$ is the indicator random variable of the event $\{ N_\Gamma^i > n \}$.

Define N^* to be the depth at which Γ, the minimum value along the correct path (after depth j) occurs. Then we can partition the underlying probability space by the events $\{ N^* = m \}$, $m = 0, 1, 2, \ldots$, and hence,

$$\overline{\eta}_\delta = \sum_{m=0}^{\infty} \overline{\eta}(\delta ; N^* = m) \tag{4.34}$$

where

$$\overline{\eta}(\delta ; N^* = m) \triangleq E^*[\, I_{\{N_\Gamma^i > n\, ;\, N^* = m\}}\, (\mathbf{U}^c, \mathbf{U}^i, \mathbf{V})\, (\underline{w}(\mathbf{U}^c, \mathbf{U}^i, \mathbf{V}))^2\,] \tag{4.35}$$

$$= \sum_{n'=n+1}^{\infty} \overline{\eta}_{n'}(\delta ; N^* = m) \tag{4.36}$$

and

$$\overline{\eta}_{n'}(\delta ; N^* = m) \triangleq E^*[\, I_{\{N_\Gamma^i = n'\, ;\, N^* = m\}}\, (\mathbf{U}_c, \mathbf{U}_i, \mathbf{V})\, (\underline{w}(\mathbf{U}^c, \mathbf{U}^i, \mathbf{V}))^2\,] \tag{4.37}$$

The next theorem identifies the importance sampling density which minimizes $\overline{\eta}_{n'}(\delta ; N^* = m)$ for all $n' \ge m$. The key complication in the proof of this theorem is

that N_{Γ}^{i} is not a *stopping time* [63] because the correct path metric minimum is determined by the entire history of the correct path metric process. In the proof of Theorem 4.1, we avoid this difficulty by utilizing the following facts: 1) for a fixed γ, N_{γ}^{i} is a stopping time; and 2) given the entire history of the $\{X_k\}$, $\mathcal{F}^X$, the correct and incorrect branch metric processes are conditionally independent. These facts are applied to an integration by parts which upper bounds $\mathcal{P}(N_{\Gamma}^{i} = n' ; N^* = m \mid \mathcal{F}^X)$, and then we use Jensen's inequality to minimize $\bar{\eta}_{n'}(\delta ; N^* = m)$ for all $n' \geq m$.

We should note here that in the proof of Theorem 4.1, we do not consider the problem of estimating the distribution of computation directly. Instead, we only consider the problem of estimating the probability that a given node δ at depth n on an incorrect path that diverges from the correct path at depth j is hypothesized by the stack algorithm. That is, we only consider the estimation of P_δ. As a result, the optimal importance sampling density that is given in Theorem 4.1 is not an optimal simulation density for estimating $\mathcal{P}(C_j \geq M)$ for a given M. Furthermore, we should note that the partitioning method simulation density does not minimize η_δ (and hence the variance of P_δ.) Specifically, the results of Theorem 4.1 are based on the minimization of $\bar{\eta}_\delta$ which is an upper bound of η_δ. Nontheless, when the partitioning method simulation model was used to estimate the distribution of computation, the resulting computational efficiency gains were very high in comparison to conventional Monte Carlo.

Lemma 4.1: Let Γ, N^*, and $\mathcal{F}^X$ be defined as above. Then for any $\alpha^c \leq 0$,

$$\mathcal{P}(\Gamma \leq \gamma ; N^* = m \mid \mathcal{F}^X) \leq \exp\left[-\alpha^c (\gamma - \sum_{k=1}^{m} Z_k^c) \right] \tag{4.38}$$

Proof: Let I_E be the indicator of the event $\{ \Gamma \le \gamma ; N^* = m \mid \mathcal{F}^X \}$. Then for any $\alpha^c \le 0$,

$$I_E \le \exp\left[-\alpha^c(\gamma - \Gamma)\right]$$

$$= \exp\left[-\alpha^c(\gamma - \sum_{k=1}^{m} Z_k^c)\right].$$

The proof of Lemma 4.1 now follows because

$$\mathcal{P}(\Gamma \le \gamma ; N^* = m \mid \mathcal{F}^X) = E[\, I_E \,].$$

□

Lemma 4.2: Let Γ, N^*, N_Γ^i, $\Lambda^i(.\,|\,)$ and $\mathcal{F}^X$ be defined as above. Then for all $\alpha^i > 0$,

$$\mathcal{P}(N_\Gamma^i = n' ; N^* = m \mid \mathcal{F}^X , \Gamma = \gamma) \le \exp\left[-\alpha^i(\gamma - \Delta) + \sum_{k=1}^{n'} \Lambda^i(\alpha^i \,|\, x_k)\right] \quad (4.39)$$

Proof: Let $I^i_{n',\gamma}(z_1^i,...,z_n^i)$ be the indicator of the event $\{N_\gamma^i = n'\}$. We have

$$\mathcal{P}(N_\Gamma^i = n' ; N^* = m \mid \mathcal{F}^X , \Gamma = \gamma) = \mathcal{P}(N_\Gamma^i = n' ; N^* = m \mid \mathcal{F}^X)$$

$$= \mathcal{P}(N_\Gamma^i = n' \mid \mathcal{F}^X)$$

$$= \int\!\!..\!\!\int I^i_{n',\gamma}(z_1^i, \ldots, z_{n'}^i)\, F_i(dz_1^i \,|\, x_1) ... F_i(dz_{n'}^i \,|\, x_{n'})$$

$$= \int\!\!..\!\!\int I^i_{n',\gamma}(z_1^i, \ldots, z_{n'}^i) \exp(-\alpha^i \sum_{k=1}^{n'} z_k^i + \sum_{k=1}^{n'} \Lambda^i(\alpha^i \,|\, x_k))$$

$$\times F_i^{(\alpha^i)}(dz_1^i \,|\, x_1) \cdots F_i^{(\alpha^i)}(dz_{n'}^i \,|\, x_{n'})$$

$$= e^{\sum_{k=1}^{n'} \Lambda^i(\alpha^i \,|\, x_k)}\, E^{(\alpha^i)}[\, e^{\alpha^i \sum_{k=1}^{N_\gamma^i} z_k^i} ; N_\gamma^i = n' \mid \mathcal{F}^X \,]$$

$$\leq \exp\left[-\alpha^i(\gamma-\Delta)+\sum_{k=1}^{n'}\Lambda^i(\alpha^i\,|\,x_k)\right].$$

The first line above exploits the conditional independence of Z_k^c and Z_k^i given $\mathcal{F}^X$. The last line above follows from the bounded branch metric assumption (Hypothesis **H**) and from definition (4.13) which together imply that $\gamma-\Delta \leq S^i_{N^i_\gamma} \leq \gamma$. The upper bound in this equation holds only if $\alpha^i \geq 0$.

□

Lemma 4.3: Let N^*, $\Lambda^i(.)$, N^i_Γ, and $\mathcal{F}^X$ be defined as above. Then there exists some constant $K < 0$ such that for all $n' \geq m$,

$$\mathcal{P}(N^i_\Gamma = n'\,;\,N^* = m \mid \mathcal{F}^X) \leq K\exp\Big(\sum_{k=1}^{n'}\Lambda^i(\alpha^i\,|\,x_k)+\alpha^c\sum_{k=1}^{m}z_k^c\Big) \qquad (4.40)$$

whenever

$$0 \leq \alpha^i < -\alpha^c \qquad (4.41)$$

Proof: From Lemma 4.1 and Lemma 4.2, we have

$$\mathcal{P}(N^i_\Gamma = n'\,;\,N^* = m) = \int_{-\infty}^{0}\mathcal{P}(N^i_\gamma = n'\,;\,N^* = m \mid \mathcal{F}^X\,,\,\Gamma=\gamma)\,d\mathcal{P}(\Gamma\leq\gamma\,;\,N^*=m\mid\mathcal{F}^X)$$

$$\leq e^{\alpha^i\Delta}\,e^{\sum_{k=1}^{n'}\Lambda^i(\alpha^i|x_k)}\int_{-\infty}^{0}e^{-\alpha^i\gamma}\,d\mathcal{P}(\Gamma\leq\gamma\,;\,N^*=m\mid\mathcal{F}^X)\,d\gamma$$

$$\leq e^{\alpha^i\Delta}\,e^{\sum_{k=1}^{n'}\Lambda^i(\alpha^i|x_k)}\left[1+\alpha^i\int_{-\infty}^{0}\mathcal{P}(\Gamma\leq\gamma\,;\,N^*=m\mid\mathcal{F}^X)\,e^{-\alpha^i\gamma}\,d\gamma\right]$$

$$\leq e^{\alpha^i\Delta}\,e^{\sum_{k=1}^{n'}\Lambda^i(\alpha^i|x_k)}\left[1+\alpha^i\,e^{\alpha^c\sum_{k=1}^{m}z_k^c}\int_{-\infty}^{0}e^{-(\alpha^c+\alpha^i)\gamma}\,d\gamma\right]$$

(by Lemma 4.1.) Consequently, we have

$$\mathcal{P}(N^i_\Gamma = n' ; N^* = m) \le K_1 \, e^{\sum_{k=1}^{n'} \Lambda^i(\alpha^i | x_k)} + K_2 \, e^{(\sum_{k=1}^{n'} \Lambda(\alpha^i | x_k) + \alpha^c \sum_{k=1}^{m} z_k^c)}$$

where

$$K_1 \triangleq e^{\alpha^i \Delta}$$

and

$$K_2 \triangleq e^{\alpha^i \Delta} \, \alpha^i \int_{-\infty}^{0} e^{-(\alpha^c + \alpha^i)\gamma} \, d\gamma$$

$$= e^{\alpha^i \Delta} \, \frac{-\alpha^i}{\alpha^c + \alpha^i}$$

whenever α^c and α^i satisfy

$$0 \le \alpha^i \le -\alpha^c$$

Next note that the fact that $\alpha^c \le 0$ which together with the fact that $\sum_{k=1}^{m} z_k^c \triangleq \Gamma$ imply that

$$\alpha^c \sum_{k=1}^{m} z_k^c \ge 0.$$

Consequently

$$\exp(\sum_{k=1}^{n'} \Lambda^i(\alpha^i | x_k)) \le \exp(\sum_{k=1}^{n'} \Lambda(\alpha^i | x_k) + \alpha^c \sum_{k=1}^{m} z_k^c)$$

and thus

$$\mathcal{P}(N^i_\Gamma = n' ; N^* = m \mid \mathcal{F}^X) \le K \exp(\sum_{k=1}^{n'} \Lambda^i(\alpha^i | x_k) + \alpha^c \sum_{k=1}^{m} z_k^c)$$

where $K = K_1 + K_2$.

□

We should note that the proof of Lemma 4.3 indicates that (4.40) holds for any n' and m. However, for the proof of the next theorem, we only need that $n' \geq m$.

Theorem 4.1: Assume the conditions of Lemma 4.3 and let $\bar{\eta}_{n'}(\delta ; N^* = m)$ be defined as in (4.37). Then the importance sampling simulation density $\underline{f}^*_{PM}$ that minimizes $\bar{\eta}_{n'}(\delta ; N^* = m)$ for all $n' \geq m$ is given by

$$\underline{f}^*_{PM} = \prod_{k=1}^{n'} f^*_k(v_k \mid u^c_k) \tag{4.42}$$

where

$$\frac{f^*_k(v_k \mid u^c_k)}{f_k(v_k \mid u^c_k)} = \begin{cases} c_k \exp\left[.5\,(\alpha^c Z^c_k + \Lambda^i(\alpha^i \mid x_k))\right] & \text{if } k \leq m \\ d_k \exp\left[.5\,\Lambda^i(\tilde{\alpha} \mid x_k)\right] & \text{otherwise} \end{cases} \tag{4.43}$$

and c_k and d_k are constants which are chosen so that $f^*_k(.\,|\,.)$ is a probability density function. The optimal values of α^c, α^i, and $\tilde{\alpha}$ are chosen so that

$$\left[\{ E[\exp(.5(\alpha^c Z^c_k + \Lambda^i(\alpha^i \mid x_k)))] \}^m \{ E[\exp(.5(\Lambda^i(\tilde{\alpha} \mid x_k)))] \}^{n'-m} \right]^2 \tag{4.43}$$

is minimized.

Proof: Using successive conditioning along with (4.40), we get

$$\bar{\eta}_{n'}(\delta; N^* = m) \triangleq E^*[\, I_{(N_I^i = n';\, N^* = m)}\, (U^c, U^i, V)\, (\underline{w}(U^c, U^i, V))^2\,]$$

$$= E^*[\, E[\, I_{(N_I^i = n';\, N^* = m)}\, (U^c, U^i, V)\, (\underline{w}(U^c, U^i, V))^2 \mid \mathcal{F}^X\,]$$

$$= E^*[\, \mathcal{P}(N_I^i = n'; N^* = m)(\underline{w}(U^c, U^i, V))^2 \mid \mathcal{F}^X\,]$$

$$\leq K\, E^*[\, \exp(\alpha^c \sum_{k=1}^{m} Z_k^c + \sum_{k=1}^{n'} \Lambda^i(\alpha^i \mid x_k))(\underline{w}(U^c, U^i, V))^2\,]$$

$$= K\, E^*[\, \prod_{k=1}^{m} \exp(\alpha^c Z_k^c + \Lambda^i(\alpha^i \mid x_k)) \prod_{k=m+1}^{n'} \exp(\Lambda^i(\tilde{\alpha} \mid x_k))$$

$$\times (\underline{w}(U^c, U^i, V))^2\,].$$

Next by Jensen's inequality,

$$E^*[\, \prod_{k=1}^{m} \exp(\alpha^c Z_k^c + \Lambda^i(\alpha^i \mid x_k)) \prod_{k=m+1}^{n'} \exp(\Lambda^i(\tilde{\alpha} \mid x_k))(\underline{w}(U^c, U^i, V))^2\,]$$

$$\geq \left[E[\, \prod_{k=1}^{m} \exp(.5(\alpha^c Z_k^c + \Lambda^i(\alpha^i \mid x_k))) \prod_{k=m+1}^{n'} \exp(.5(\Lambda^i(\tilde{\alpha} \mid x_k)))\,] \right]^2$$

$$= \left[\{ E[\, \exp(.5(\alpha^c Z_k^c + \Lambda^i(\alpha^i \mid x_k)))\,] \}^m \{ E[\, \exp(.5(\Lambda^i(\tilde{\alpha} \mid x_k)))\,] \}^{n'-m} \right]^2$$

with equality if and only if

$$\prod_{k=1}^{m} \exp(\alpha^c Z_k^c + \Lambda^i(\alpha^i \mid x_k)) \prod_{k=m+1}^{n'} \exp(\Lambda^i(\tilde{\alpha} \mid x_k))(\underline{w}(U^c, U^i, V)) = c \quad (4.44)$$

almost surely with respect to the importance sampling density $\underline{f}_{PM}^*$ (c in the above equation is simply a constant.) Theorem 4.1 now follows because

$$\underline{w}(U^c, U^i, V) \triangleq \prod_{k=1}^{n'} \frac{f_k(v_k \mid u_k^c)}{f_k^*(v_k \mid u_k^c)}$$

□

It turns out that for the practical implementation of the importance sampling model (4.43), one should choose $\tilde{\alpha} \equiv 0$. In this case, the partitioning method simulation model becomes

$$\frac{f_k^*(v_k \mid u_k^c)}{f_k(v_k \mid u_k^c)} = \begin{cases} c_k \exp\left[.5\,(\alpha^c Z_k^c + \Lambda^i(\alpha^i \mid x_k))\right] & \text{if } k \le m \\ 1 & \text{otherwise} \end{cases} \tag{4.45}$$

For this simulation model, the optimal values of α^c and α^i must be chosen so that

$$E[\exp(.5(\alpha^c Z_k^c + \Lambda^i(\alpha^i \mid x_k)))] \tag{4.46}$$

is minimized.

Notice that letting $\tilde{\alpha} = 0$ insures that the importance sampling weight will converge to 1. Furthermore, note that if we continue the simulation of the stack algorithm using $\tilde{\alpha} \neq 0$ (that is, with a noisy channel,) then the resulting simulations will tend to produce a lot of errors and the incorrect subtrees may become excessively large. This is the justification for choosing $\tilde{\alpha} = 0$.

4.5.3 The Partitioning Method Applied to the BSC

In this section we will apply the results of the partitioning method to the special case when the channel is a binary symmetric channel. As in the previous section, let $\{U_k^c\}$, and $\{U_k^i\}$, k= 1,2,... denote the encoder output sequences associated with the correct path and any incorrect path, respectively. Recall that $\{U_k^c\}$ and $\{U_k^i\}$ are both assumed to be independent and i.i.d. random processes with the same univariate density q(.). We shall assume that both the input alphabet and the output alphabet of the channel are $\{-1,1\}$ with

$$q(u) = \begin{cases} 1/2 & \text{if } u = 1 \\ 1/2 & \text{if } u = -1 \end{cases} \tag{4.47}$$

Now let Z_k^c and Z_k^i be the correct and the incorrect branch metrics. Furthermore, let $\Lambda^c(.|.)$ and $\Lambda^i(.|.)$ be the conditional log-moment generating functions associated with the correct and incorrect branch metric processes. In order to make our notation for these parameters as simple as possible, we will assume in the reminder of this chapter that there is only 1 use of the channel for each information symbol input to the convolutional encoder. In other words, we shall assume that the rate of the code is 1 bit per output symbol. Generalization of the above expressions for rate b/n codes is straightforward. Thus, assuming a Fano metric. Then

$$Z_k^c = \begin{cases} \ln(2(1-\varepsilon)) - R & \text{if } U_k^c = V_k \\ \ln(2p) - R & \text{if } U_k^c \neq V_k \end{cases} \tag{4.48}$$

and

$$Z_k^i = \begin{cases} \ln(2(1-\varepsilon)) - R & \text{if } U_k^i = V_k \\ \ln(2p) - R & \text{if } U_k^i \neq V_k \end{cases} \tag{4.49}$$

where ε is the crossover probability, V_k is the channel output symbol, and R is the code rate.

A straightforward computation indicates that

$$Z_k^c = \begin{cases} \ln(2(1-\varepsilon)) - R & \text{with probability } 1-\varepsilon \\ \ln(2p) - R & \text{with probability } \varepsilon \end{cases} \tag{4.50}$$

and

$$Z_k^i = \begin{cases} \ln(2(1-\varepsilon)) - R & \text{with probability } 1/2 \\ \ln(2p) - R & \text{with probability } 1/2 \end{cases} \tag{4.51}$$

Consequently, when conditioned on $X_k \triangleq (U_k^c, V_k^i)$, Z_k^c becomes a deterministic

quantity, and thus for any $\alpha^c \in \mathbf{R}$

$$\Lambda^c(\alpha^c \mid x_k) = \alpha^c z_k^c \tag{4.52}$$

Close observation of (4.51) shows that Z_k^i is independent of X_k in this special case. Consequently for any $\alpha^i \leq 0$,

$$\Lambda^i(\alpha^i \mid x_k) \equiv \Lambda^i(\alpha^i) \tag{4.53}$$

$$\triangleq \ln(E[\, e^{\alpha^i z_k^i}\,])$$

$$= \ln\left[\frac{1}{2}(e^{\alpha^i a} + e^{-\alpha^i b})\right] - \alpha^i R \tag{4.54}$$

where a and b are defined as

$$a \triangleq \ln(2(1-\varepsilon)) \tag{4.55a}$$

and

$$b \triangleq -\ln(2\varepsilon) \tag{4.55b}$$

Now recall that the partitioning method model (with $\tilde{\alpha} \neq 0$) is given by

$$\frac{f_k^*(v_k \mid u_k^c)}{f_k(v_k \mid u_k^c)} = \begin{cases} c_k \exp\left[.5\,(\alpha^c Z_k^c + \Lambda^i(\alpha^i \mid x_k))\right] & \text{if } k \leq m \\ d_k \exp\left[.5\,\Lambda^i(\tilde{\alpha} \mid x_k)\right] & \text{otherwise} \end{cases} \tag{4.56}$$

For a given α^i and $\tilde{\alpha}$, notice that $\exp(\Lambda^i(\tilde{\alpha} \mid x_k)) = \exp(\Lambda^i(\tilde{\alpha}))$. Consequently, the simulation model (4.56) is automatically reduced to the simulation model (4.45). Thus, the partioning method simulation model in the BSC case is given by

$$\frac{f_k^*(v_k \mid u_k^c)}{f_k(v_k \mid u_k^c)} = \begin{cases} c_k \exp(.5\alpha^c Z_k^c) & \text{if } k \leq m \\ 1 & \text{otherwise} \end{cases} \tag{4.57}$$

Solving for c_k, we get

$$c_k = e^{-\Lambda^c(.5\alpha^c)} \tag{4.58}$$

where

$$\Lambda^c(\alpha^c) \triangleq \ln(E[\, e^{\alpha^c Z_k^c}\,])$$

$$= \ln(\,(1-\varepsilon)e^{\alpha^c a} + \varepsilon e^{-\alpha^c b}\,) - \alpha^c R \tag{4.59}$$

and a and b are defined as in (4.55a) and (4.55b). Consequently, for the BSC case the partitioning method simulation model is given by

$$f_k^*(v_k \mid u_k^c) = \begin{cases} e^{.5\alpha^c Z_k^c - \Lambda^c(\alpha^c/2)}\, f_k(v_k \mid u_k^c) & \text{if } k \le m \\ f_k(v_k \mid u_k^c) & \text{otherwise} \end{cases} \tag{4.60}$$

or equivalently, it is based on a non-stationary memoryless BSC model which is characterized by a time varying crossover probability

$$\varepsilon_k^* = \begin{cases} e^{.5\alpha^c Z_k^c - \Lambda^c(\alpha^c/2)}\, \varepsilon & \text{if } k \le m \\ \varepsilon & \text{otherwise} \end{cases} \tag{4.61}$$

where $\Lambda^c(\alpha^c)$ is given in (4.59).

Finally note that from (4.46), it follows that the optimal value of $\alpha^c \le 0$ for the partitioning method simulation model must be chosen

$$E[\, \exp(\, .5(\, \alpha^c Z_k^c + \Lambda^i(\alpha^i \mid x_k)\,)\,)\,]$$

is minimized. Since Z_k^i is independent of X_k in the BSC case, we have $\Lambda^i(\alpha^i \mid x_k) \equiv \Lambda^i(\alpha^i)$. Consequently,

$$E[\, \exp(\, .5(\, \alpha^c Z_k^c + \Lambda^i(\alpha^i \mid x_k)\,)\,)\,] = e^{.5\Lambda^i(\alpha^i)}\, E[\, \exp(\, .5\alpha^c Z_k^c\,]$$

and thus the optimal α^c should be chosen so that

$$E[\, \exp(\, .5\alpha^c Z_k^c\,] \triangleq e^{\Lambda^c(\alpha^c/2)}$$

is minimized.

4.5.4 The Partitioning Method Applied to the AWGN Channel

We shall now apply the partitioning method to the additive white gaussian noise channel case. Without loss of generality, we shall assume that the all zero path is the transmitted path. Consequently, $U_k^c \equiv -1$. Since the channel is assumed to be an AWGN channel, then the output channel symbols V_k can be written as follows:

$$V_k = -1 + N_k$$

where N_k is the white gaussian random process which is assumed to have zero mean and variance σ^2 (recall section 4.2.1.) As in the previous section, we shall assume that for any incorrect path the corresponding encoder output symbols $\{U_k^i\}$ for k= 1,2,.. are mutually independent, with univariate density q(.). Thus on any incorrect path, we have $U_k^i = \pm 1$ with probability 1/2.

Assuming a Fano metric, then the incorrect and the correct branch metrics are given by

$$Z_k^c = -v_k - \sigma^2[\ \ln(\cosh(\frac{v_k}{\sigma^2}) + R\ ] \tag{4.62}$$

and

$$Z_k^i = \begin{cases} -v_k - \sigma^2[\ \ln(\cosh(\frac{v_k}{\sigma^2}) + R\ ] & \text{if } U_k^i = 1 \\ v_k - \sigma^2[\ \ln(\cosh(v_k\sigma^2) + R\ ] & \text{if } U_k^i = -1 \end{cases} \tag{4.63}$$

Consequently, we have

$$\Lambda^c(\alpha^c \mid x_k) \equiv \alpha^c Z_k^c \tag{4.64}$$

where Z_k^c is given by (4.62). Likewise, the incorrect branch metric conditional log-

moment generating function is given by

$$\Lambda^i(\alpha^i \mid x_k) = \ln\left[E[e^{\alpha^i z_k^i} \mid x_k] \right] \tag{4.65}$$

$$= \ln\left[.5(e^{-\alpha^i(v_k + a(v_k) + b(v_k))} + e^{\alpha^i(v_k + a(v_k) + b(v_k))} \right]$$

$$= \ln\left[\cosh(\alpha^i v_k)\, e^{\alpha^i(a(v_k) + b(v_k))} \right]$$

or equivalently,

$$\Lambda^i(\alpha^i \mid x_k) = \ln(\cosh(\alpha^i v_k)) - \alpha^i(a(v_k) + b(v_k)) \tag{4.65}$$

where

$$a(v_k) \triangleq \sigma^2 \ln(\cosh(\frac{v_k}{\sigma^2}) \tag{4.66a}$$

and

$$b(v_k) \triangleq \sigma^2 R. \tag{4.66b}$$

Define

$$\Psi(u_k^c, u_k^i, v_k) \triangleq \alpha^c z_k^c + \Lambda^i(\alpha^i \mid x_k). \tag{4.67}$$

Then for any $0 \le \alpha^i < -\alpha^c$, the partitioning method importance sampling density is given by

$$f_k^*(v_k \mid u_k^c) \propto \begin{cases} \exp\left[.5\, \Psi(u_k^c, u_k v_k)\right] f_k(v_k \mid u_k^c) & \text{if } k \le m \\ f_k(v_k \mid u_k^c) & \text{otherwise} \end{cases} \tag{4.68}$$

Thus if we approximate ln(x) by |x| - ln(2), it follows that

$$\Psi(u_k^c, u_k^i, v_k) = -\alpha^c v_k - (\alpha^c + \alpha^i)\, \sigma^2\, [\ln(\cosh(\frac{v_k}{\sigma^2}) + R] + \ln(\cosh(\alpha^i v_k))$$

$$\approx -\alpha^c v_k - (\alpha^c + \alpha^i)\,\sigma^2\,[\frac{|v_k|}{\sigma^2} - \ln(2) + R] + \alpha^i\,|v_k| - \ln(2)$$

$$= -\alpha^c(v_k + |v_k|) + (\alpha^c + \alpha^i)\,\sigma^2(\ln(2) - R) - \ln(2).$$

Consequently,

$$\Psi(u_k^c, u_k^i, v_k) \approx \begin{cases} (\alpha^c + \alpha^i)\sigma^2\,(\ln(2) - R) - \ln(2) & \text{if } v_k \le 0 \\ -2\alpha^c v_k + ((\alpha^c + \alpha^i)\,\sigma^2\,(\ln(2) - R) - \ln(2) & \text{if } v_k > 0 \end{cases} \tag{4.69}$$

and thus by substituting the above equation in (4.68), we can see that for all $k \le m$

$$f_k^*(v_k \mid u_k^c) = \begin{cases} C\, f_k(v_k \mid u_k^c) & \text{if } v_k \le 0 \\ C\, e^{-\alpha^c v_k}\, f_k(v_k \mid u_k^c) & \text{if } v_k > 0 \end{cases} \tag{4.70}$$

where C is some constant for a given α^c and α^i. Since $f_k^*(.\,|\,.)$ is a probability density function, it follows that

$$\frac{C}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{0} e^{-\frac{(v_k+1)^2}{2\sigma^2}}\, dv_k + \exp[\,\frac{(1+\alpha^c\sigma^2)^2 - 1}{2\sigma^2}\,] \int_{0}^{\infty} e^{-\frac{(v_k + (1+\alpha^c\sigma^2))^2}{2\sigma^2}}\, dv_k = 1.$$

Consequently,

$$C = \left[(1 - Q[\frac{1}{\sigma}]) + \exp[\,\frac{(1+\alpha^c\sigma^2)^2 - 1}{2\sigma^2}\,]\, Q[\frac{1+\alpha^c\sigma^2}{\sigma}]\right]^{-1} \tag{4.71}$$

where

$$Q(t) \triangleq \int_t^{\infty} \frac{1}{\sqrt{2\pi}}\, e^{-y^2/2}\, dy$$

It is now convenient to define the following density functions:

$$f^{-}(y) \triangleq \frac{\frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(y+1)^2}{2\sigma^2}}}{1 - Q[\frac{1}{\sigma}]} \tag{4.72}$$

and

$$f^+(y) \triangleq \frac{\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y+(1+\alpha^c\sigma^2))^2}{2\sigma^2}}}{Q[\frac{1+\alpha^c\sigma^2}{\sigma}]} \tag{4.73}$$

$f^-(y)$ is simply the conditional density of y given that $y \leq 0$ and y is a gaussian random variable with mean -1 and standard deviation σ. Similarly, $f^+(y)$ is the conditional density of y given that $y > 0$ and y is a gaussian random variable with mean $= -(1+\alpha^c\sigma^2)$ and standard deviation σ.

A close observation of (4.70)-(4.73) indicates that the importance sampling density $f_k^*(v_k \mid u_k^c)$ in (4.70) can be rewritten as

$$f_k^*(v_k \mid u_k^c) = \begin{cases} q^- f^-(v_k) & \text{if } v_k \leq 0 \\ q^+ f^+(v_k) & \text{if } v_k > 0 \end{cases} \tag{4.74}$$

where $f^-(.)$ and $f^+(.)$ are defined in (4.72) and (4.73). The constants q^- and q^+ are given by

$$q^- = C\,(1 - Q[\frac{1}{\sigma}]) \tag{4.75}$$

and

$$q^+ = C \exp[\frac{(1+\alpha^c\sigma^2)^2 - 1}{2\sigma^2}]\, Q[\frac{1+\alpha^c\sigma^2}{\sigma}] \tag{4.76}$$

with C being the constant given by (4.71). Consequently, for a given $N^* = m$, it follows that the random observations needed in the partitioning method simulations for all $k \leq m$ can be realized by sampling from $f_k^*(v_k \mid u_k^c)$ as follows: with probability q^-, the random samples will be generated from the density $f^-(.)$ and with probability

$q^+ = 1 - q^-$, the random samples will be generated from the density $f^+(.)$. The optimal q^- and q^+ should be chosen so that (4.46) is minimized.

Figure 4.12 shows the simulation density $f_k^*(v_k \mid u_k)$ for $\sigma = .5$, $q^- = .63$, and $q^+ = .37$ (assuming $k \leq N^*$.)

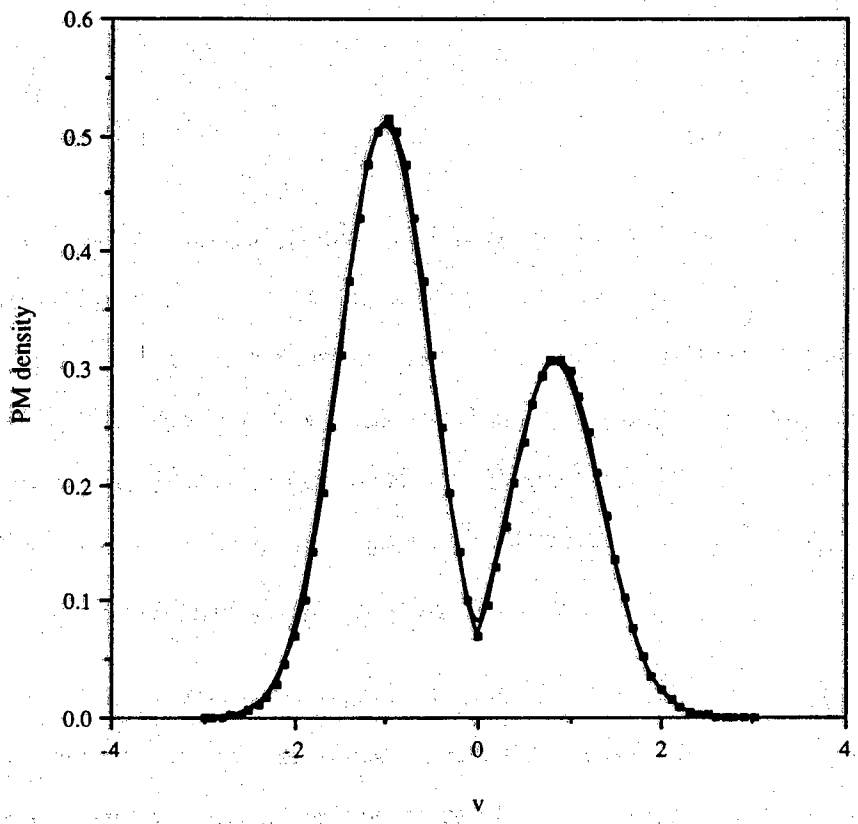


Figure 4.12: The partitioning method simulation density for the AWGN channel with $\sigma = .5$, $q^- = .63$, and $q^+ = .37$.

4.6 The Partitioning Method in the Markov Case

In this section we shall extend the partitioning method analysis to the Markov case. That is, instead of the i.i.d branch metric process assumption we consider stationary branch metrics arising from a *uniformly recurrent Markov-Additive* (MA) process. This model assumes that the branch metric process distributions are governed by an underlying Markov chain. The uniform recurrence hypothesis is very strong, but it allows us to present our results with a minimal amount of preliminaries. Ruffly speaking, uniformly recurrent chains are those chains which behave like chains on a finite state space in the sense that they admit a strong Perron-Frobenius theory. We should note that the uniform recurrence model does include two important cases: the i.i.d. case and the finite state space Markov chain case.

We shall begin this section by presenting the required background on MA processes.

4.6.1 Markov Additive Processes

Let $\{X_k\ ;\ k=0,1,..\}$ be a Markov chain taking values in a measurable space $(\mathbf{E}, \mathcal{E})$. We associate with $\{X_k\}$ the following additive components

$$S_n^c = \sum_{k=1}^{n} Z_k^c \quad \text{and} \quad S_n^i = \sum_{k=1}^{n} Z_k^i$$

where $\{Z_k^c\}$ and $\{Z_k^i\}$ are real valued processes associated with $\{X_k\}$ as follows: For any $x \in E$, $A \in \mathcal{E}$ and $B^c \times B^i \in \mathcal{B}^2$,

$$P((X_{n+1},Z^c_{n+1},Z^i_{n+1}) \in A\times B^c\times B^i \mid X_n=x\,,\ \mathcal{F}_n)$$

$$= P((X_{n+1}Z^c_{n+1},Z^i_{n+1}) \in A\times B^c\times B^i \mid X_n=x)$$

$$\triangleq P(x,A\times B^c\times B^i) \tag{4.77}$$

where $\mathcal{F}_n \triangleq \sigma(X_0,...,X_n,Z^c_1,...,Z^c_n,Z^i_1,...,Z^i_n)$ and $\mathcal{B}$ is the Borel field on $\mathbf{R}$. In this context, the superscripts c and i will denote "correct" and "incorrect" paths. The triple $\{(X_n,S^c{}_n,S^i{}_n)\}$ is a two dimensional *Markov additive* (MA) process and $P : E\times(\mathcal{E}\times B^c\times B^i)\to[0,1]$ is its MA transition Kernel [64-66]. We shall further restrict our Markov model so that when conditioned on $\mathcal{F}^X=\sigma(X_0,X_1,...)$, $\{Z^c_k\}$ and $\{Z^i_k\}$ are conditionally independent. Specifically, we assume that

$$P(x,dy\times B^c\times B^i) = Q(x,dy)\,F_c(B^c \mid x,y)\,F_i(B^i \mid x,y) \tag{4.78}$$

where $Q(x,dy)$ is the transition probability of the Markov chain $\{X_k\}$. $F_i(.\mid x,y)$ and $F_c(.\mid x,y)$ are respectively the conditional distributions of Z^i_k and Z^c_k given $(X_{k-1},X_k)=(x,y)$. An MA process of the form (4.78) is called a *separable* MA Process.

In the sequel we will assume Hypothesis **I** below, in particular, condition (4.79) is what we call the *uniform recurrence* condition.

Hypothesis I:

(i) The triple $\{(X_n,S^c{}_n,S^i{}_n)\}$ is a separable MA process with transition probability of the form (4.78), which is *uniformly recurrent.* That is, there exists a probability measure ν on $\mathcal{E}\times\mathcal{B}^2$, and real numbers $0<c<c'<\infty$, such that for some integer $n_0<\infty$,

$$c\,\nu(A\times\Gamma) \le P^{n_0}(x,A\times\Gamma) < c'\nu(A\times\Gamma) \tag{4.79}$$

for all $x \in E$, $A \in \mathcal{E}$, and $\Gamma \in \mathcal{B}^2$.

(ii) The increments processes Z^c_k and Z^i_k are bounded, that is, $|Z^c{}_k| \le \Delta$ and $|Z^i{}_k| \le \Delta$ for some $\Delta < \infty$; and

(iii) $\nu(\mathbf{E}\times\Gamma) > 0$ for $\Gamma = (0,\infty)\times\mathbf{R}$, $(-\infty,0)\times\mathbf{R}$, $\mathbf{R}\times(0,\infty)$ and $\mathbf{R}\times(-\infty,0)$.

Note that part (iii) of hypothesis I insures that neither $\{S^c_n\}$ nor $\{S^i_n\}$ are monotone increasing or decreasing processes. For example, if $\nu(E\times(-\infty,0)\times B^c) = 0$ then from (4.79) it is apparent that $P(x, A\times(-\infty,0)\times B^c) = 0$ for all $x \in \mathbf{E}$ and $A \in \mathcal{E}$, and hence $Z^c_k \ge 0$ almost surely.

Now for each $\alpha \in \mathbf{R}^2$, define the following non-negative Kernel $\hat{P}(\alpha) : \mathbf{E} \times \mathcal{E} \to [0,\infty]$ as

$$\hat{P}(x, A;\alpha) \triangleq \int_{\mathbf{R}^2} \exp(\alpha z)\, P(x, A\times dz) \tag{4.80}$$

for any $x \in E$ and $A \in \mathcal{E}$. $\{ \hat{P}(\alpha) : \alpha \in \mathbf{R}^2 \}$ is called the family of transform kernels [64]. Part (ii) of hypothesis I implies that $\int \exp^{\alpha z} \nu(\mathbf{E}\times dz) < \infty$, and this in turn implies that $\hat{P}(\alpha)$ is a bounded operator ($\hat{P}(\alpha)f(x) = \int f(y)\, \hat{P}(x,dy;\alpha)$) on the Banach space of bounded real valued Borel measurable functions. Hence, the spectral radius of $\hat{P}(\alpha)$ is well defined and finite for all $\alpha \in \mathbf{R}^2$. The following Lemmas are taken from [66]. In particular, Lemma 1 is a generalized Perron-Frobenius result originally due to T. Harris which states that the spectral radius of $\hat{P}(\alpha)$ is actually obtained by a non-negative eigenvalue.

Lemma 4.4: Assume hypothesis I. Then for each $\alpha \in \mathbf{R}^2$, $\hat{P}(\alpha)$ has a maximal simple real eigenvalue $0 < \lambda(\alpha) < \infty$, and an essentially unique right eigenfunction $r(x;\alpha)$. Furthermore, the eigenfunction is uniformly positive and bounded; that is, there exist constants $k_1(\alpha)$ and $k_2(\alpha)$ such that

$$0 < k_1(\alpha) \leq r(x;\alpha) \leq k_2(\alpha) < \infty \tag{4.81}$$

for all $x \in \mathbf{E}$.

Define

$$\Lambda(\alpha) = \ln(\lambda(\alpha)) \quad \text{for } \alpha \in \mathbf{R}^2 \tag{4.82}$$

Lemma 4.5: Assume hypothesis I. Then (i) $\Lambda(\alpha)$ is analytic and strictly convex on $\mathbf{R}^2$, and (ii) $\Lambda(\alpha) \uparrow \infty$ as $||\alpha|| \to \infty$.

4.6.2 The Partitioning Method Results

Let S_n^c and S_n^i denote the correct and incorrect path metric processes. We shall assume that the triple $\{(X_n, S_n^c, S_n^i)\}$ is a separable MA process with transition probability of the form (4.78). Furthermore, we shall assume that hypothesis I is satisfied.

Define

$$\Lambda^c(\alpha^c) \triangleq \Lambda(\alpha^c, 0) \tag{4.83}$$

and

$$\Lambda^i(\alpha^i) \triangleq \Lambda(0, \alpha^i). \tag{4.84}$$

These are the log-eigenvalue functions for the one dimensional MA processes $\{(X_n, S_n^c)\}$ and $\{(X_n, S_n^i)\}$ respectively. We shall assume the following *drift condition:*

$$\frac{d}{d\beta}\Lambda^i(0) < 0 \quad \text{and} \quad \frac{d}{d\beta}\Lambda^c(0) > 0. \tag{4.85}$$

We note that from a large deviations theorem for uniformly recurrent MA processes (see Theorem 5.1 in [64]), $S^c{}_n/n$ and $S^i{}_n/n$ converge to $d\Lambda^c(0)/d\beta$ and $d\Lambda^i(0)/d\beta$ at an exponential rate. Consequently, the derivatives in (4.85) determine the *drift* of the additive processes $\{S^c_n\}$ and $\{S^i_n\}$. Recall that the stack algorithm will not work if the incorrect nodes tend to increase faster than the correct ones. Thus, (4.85) is intuitively a minimal drift condition.

Now recall that in section 4.5.1, we assumed that $\{X_k\}$ is an i.i.d. sequence. In this section, however, we shall let $\{X_k: k=0,1,...\}$ be a Markov chain taking values in a measurable $(\mathbf{E}, \mathcal{E})$ with transition probability $Q(x,dy)$ and recurrence measure μ. This Markov chain state space may be infinite dimensional, hence, this model provides a rich class of stationary branch metric process distribution.

As in the section 4.5.2 we consider the problem of estimating P_δ (recall (4.23)) via importance sampling. We shall restrict our attention to importance sampling simulation with uniformly recurrent Markov chains generated by a transition probability $Q^*(x,dy)$. Furthermore, we shall assume that $Q(x,dy)$ is absolutely continuous with respect to $Q^*(x,dy)$ (This is required to get an unbiased importance sampling estimator.) In this case, the importance sampling weight (4.26) becomes

$$\underline{w}(\mathbf{u}^c,\mathbf{u}^i,\mathbf{v}) = \prod_{k=1}^{n} \frac{Q(x_{k-1},x_k)}{Q^*(x_{k-1},x_k)}. \tag{4.86}$$

The next theorem is basically an extension of Theorem 4.1 to the Markov case.

Theorem 4.2: Assume the conditions of Lemma 4.3 and Hypothesis I. Furthermore, let Q(x,dy), $Q^*(x,dy)$ be defined as above, and $\bar{\eta}_{n'}(\delta ; N^* = m)$ be defined as in (4.37). Then the importance sampling model that minimizes $\bar{\eta}_{n'}(\delta ; N^* = m)$ is characterized by

$$\frac{Q^*(x_{k-1},x_k)}{Q(x_{k-1},x_k)} = \begin{cases} r_k \exp\left[.5\,(\alpha^c Z_k^c + \Lambda^i(\alpha^i \mid x_k))\right] & \text{if } k \le m \\ s_k \exp\left[.5\,\Lambda^i(\tilde{\alpha} \mid x_k)\right] & \text{otherwise} \end{cases} \tag{4.87}$$

where r_k and s_k are constants which are chosen so that $Q^*(x,dy)$ is a valid transition probability. The optimal values of α^c, α^i, and $\tilde{\alpha}$ are chosen so that

$$\left[\{ E[\exp(.5(\alpha^c Z_k^c + \Lambda^i(\alpha^i \mid x_k)))]^m \{ E[\exp(.5(\Lambda^i(\tilde{\alpha} \mid x_k)))]^{n'-m} \right]^2 \tag{4.89}$$

is minimized.

Proof: First notice that under the assumptions of Theorem 4.2, Lemmas 4.1-4.3 are still valid. Consequently, the proof of Theorem 4.2 can be done by simply following the same steps used in the proof of Theorem 4.1. The main difference between the proofs of both of these theorems is the importance sampling weight. In the proof of Theorem 4.1, the IS weight is given by (4.26). In the proof of Theorem 4.2, however, the IS weight is given by (4.86).

□

4.7 The M-method

In this section, we propose another importance sampling technique which we shall refer to as the *M-method*. The M-method is an importance sampling simulation model which is inspired by the partitioning method. Our goal is is to apply this

scheme to simulate (stack algorithm) sequential decoders.

We shall begin our discussion of the M-method by developing various definitions which are needed for the discussions to follow.

Let X be a random variable with probability density function $f_X(x)$ and let g(.) be a real valued function of X. As in Section 2.2.1, we shall consider the problem of estimating

$$\alpha = E[g(X)] \tag{4.90}$$
$$= \int g(x)\, f_X(x)\, dx$$

Now let M be a discrete random variable with a probability mass function p_m. That is,

$$\mathcal{P}(M = m) = p_m \quad \text{for} \quad m = 1,2,...,J. \tag{4.91}$$

We shall assume that the random variables M and X are statistically independent so that their joint density $f_{XM}(x,m)$ is given by

$$f_{XM}(x,m) = f_X(x)\, p_m \tag{4.92}$$

Next let $f^*_{XM}(x,m)$ be the importance sampling density which we shall use in order to estimate α. Note that this density can be written as

$$f^*_{XM}(x,m) = f^*_{X|M}(x|m)\, p^*_m \tag{4.93}$$

with p^*_m and $f^*_{X|M}(.|.)$ being respectively the conditional and marginal importance sampling densities.

The *basic idea* behind the M-method is to write α in (4.90) as

$$\alpha = E^*[\, g(X)\, w(X,M)\,] \tag{4.94}$$

where

$$w(x,m) \triangleq \frac{f(x,m)}{f^*_{XM}(x,m)} \tag{4.95}$$

is the *joint importance sampling weight* and $E^*[.]$ is the expectation with respect to $f^*(x,m)$. Notice here that the basic importance sampling principles developed in Section 2.2.1 apply directly to the joint sampling of the pair (X,M). However, observe that we can rewrite the joint weight (4.95) as

$$w(x,m) = w_{X|M}(x|m)\, w_M(m) \tag{4.96}$$

where

$$w_{X|M}(x|m) \triangleq \frac{f_X(x)}{f^*_{X|M}(x|m)} \tag{4.97}$$

and

$$w_M(m) \triangleq \frac{p_m}{p^*_m}. \tag{4.98}$$

The importance sampling estimator for α is given by

$$\hat{\alpha} = \frac{1}{L} \sum_{\ell=1}^{L} g(X^{(\ell)})\, w(X^{(\ell)}|M^{(\ell)})\, w(M^{(\ell)}) \tag{4.99}$$

where the simulation data $X^{(1)}, \ldots, X^{(L)}$ and $M^{(1)}, \ldots, X^{(L)}$ are i.i.d. random samples being respectively generated from $f^*_{X|M}(x|m)$ and p^*_m. Notice here that the estimator (4.99) is implemented as follows: For ℓ=1,2,...,L, the samples $M^{(\ell)}$ are generated from the marginal density p^*_M. For each of these samples, the samples $X^{(\ell)}$ are then generated from the conditional density $f^*_{X|M}(x|m)$.

Because the simulation data is i.i.d., it follows that the importance sampling estimator $\hat{\alpha}$ is unbiased if and only if it is unbiased for L = 1. Thus, it is sufficient to show that $E^*[\, g(X)\, w_{X|M}(X|M)\, w_M(M)\,] = \alpha$. By considering (4.94)-(4.98), we have

$$
\begin{aligned}
&E^*[\, g(X)\, w_{X|M}(X|M)\, w_M(M)\,] \\
&\quad = \sum_{m=1}^{J} E^*[\, g(X)\, w_{X|M}(X|m) \mid M=m\,]\, w_M(M)\, p^*_M \\
&\quad = \sum_{m=1}^{J} \int g(X)\, \frac{f_X(x)}{f^*_{X|M}(x|m)}\, f^*_{X|M}(x|m)\, dx\, \frac{p_m}{p^*_m}\, p^*_m \\
&\quad = \sum_{m=1}^{J} \int g(X)\, f_X(x)\, dx\, p_m \\
&\quad = \alpha \sum_{m=1}^{J} p_m \\
&\quad = \alpha.
\end{aligned}
$$

This completes the proof.

A similar computation indicates that the variance of the estimator (4.99) is

$$\mathrm{var}^*[\hat{\alpha}] = \frac{1}{L}[\,\eta - \hat{\alpha}^2\,] \tag{4.100}$$

where

$$
\begin{aligned}
\eta &= E^*[\, g^2(X)\, w^2_{X|M}(X|M)\, w^2_M(M)\,] \qquad (4.101)\\
&= E^*[\, E^*[\, g^2(X)\, w^2_{X|M}(X|m) \mid M=m\,]\, w^2_M(M)\,] \\
&= \sum_{m=1}^{J} E^*[\, g^2(X)\, w^2_{X|M}(X|m) \mid M=m\,]\, w^2_M(m)\, p^*_m \\
&= \sum_{m=1}^{J} E^*[\, g^2(X)\, w^2_{X|M}(X|m) \mid M=m\,]\, \frac{1}{p^*_m}\, p^2_m \\
&= \sum_{m=1}^{J} \eta_m \frac{1}{p^*_m}\, p^2_m
\end{aligned}
$$

where

$$\eta_m \triangleq E^*[\, g^2(X)\, w^2_{X|M}(X|m) \mid M=m \,]. \tag{4.102}$$

Now note that if we define

$$a_m \triangleq \frac{\eta_m}{p^*_m}. \tag{4.103}$$

Then for a given m, a_m is simply a constant. However, notice that p_m is independent of the simulation densities $f^*_{X|M}(x|m)$ and p^*_m. Thus, by writing η as

$$\eta = \sum_{m=1}^{J} a_m\, p_m^2. \tag{4.104}$$

Then for a given simulation densities $f^*_{X|M}(x|m)$ and p^*_m, one can select p_m in such a way that the variance of the estimator (4.99) is minimized. Thus, for a given $f^*_{X|M}(x|m)$ and p^*_m, we may consider the following minimization problem:

$$\text{Minimize} \qquad \eta = \sum_{m=1}^{J} a_m\, p_m^2 \tag{4.105}$$

subject to

$$\sum_{m=1}^{J} p_m = 1 \quad \text{and} \quad p_m \geq 0 \quad \text{for } m = 1,2,\ldots,J. \tag{4.106}$$

A straightforward computation (using Lagrange multipliers) indicates that the solution to the above (constrained) minimization problem is given by

$$p_m = \left[\sum_{i=1}^{J} \frac{\eta_i}{p^*_i}\right]^{-1} \frac{p^*_m}{\eta_m}. \tag{4.107}$$

In this case, η is given by

$$\eta = \sum_{m=1}^{J} \frac{p^*_m}{\eta_m} \left[\sum_{i=1}^{J} \frac{\eta_i}{p^*_i}\right]^{-2} \tag{4.108}$$

where η_m is defined in (4.102).

As a final remark in this section, we notice that in the special case where $p_M^* \equiv p_M$, it follows that $w_M(m) \equiv 1$, and hence the importance sampling estimator (4.99) reduces to

$$\hat{\alpha} = \frac{1}{L} \sum_{\ell=1}^{L} g(X^{(\ell)})\, w(X^{(\ell)} | M^{(\ell)}). \tag{4.109}$$

The variance for the above estimator is still given by (4.100). However, in this case η is given by

$$\eta = \sum_{m=1}^{J} E^*[\, g^2(X)\, w^2_{X|M}(X|m) \mid M=m \,]\, p_m. \tag{4.110}$$

CHAPTER 5
SEQUENTIAL DECODERS SIMULATION USING IMPORTANCE SAMPLING

5.1 Introduction

In this chapter, we shall discuss and demonstrate the potential of the importance sampling techniques which we have developed in the previous chapter. Our main objective is to verify and demonstrate the power and accuracy of these importance sampling schemes. Throughout this chapter, we will consider convolutional codes with rate 1/2 and constraint lengths 5, 14 and 21. These codes are sufficiently complex that their simulations via ordinary Monte Carlo is quite difficult, if not impossible, for low noise conditions. The convolutional encoders that generate these codes are shown in Figures 5.1-5.3. Because the communications channels on which the convolutional codes are going to operate are assumed to be discrete memoryless binary input-output channels, it follows that we can arbitrarily set the correct path to be the *all-zero path.*

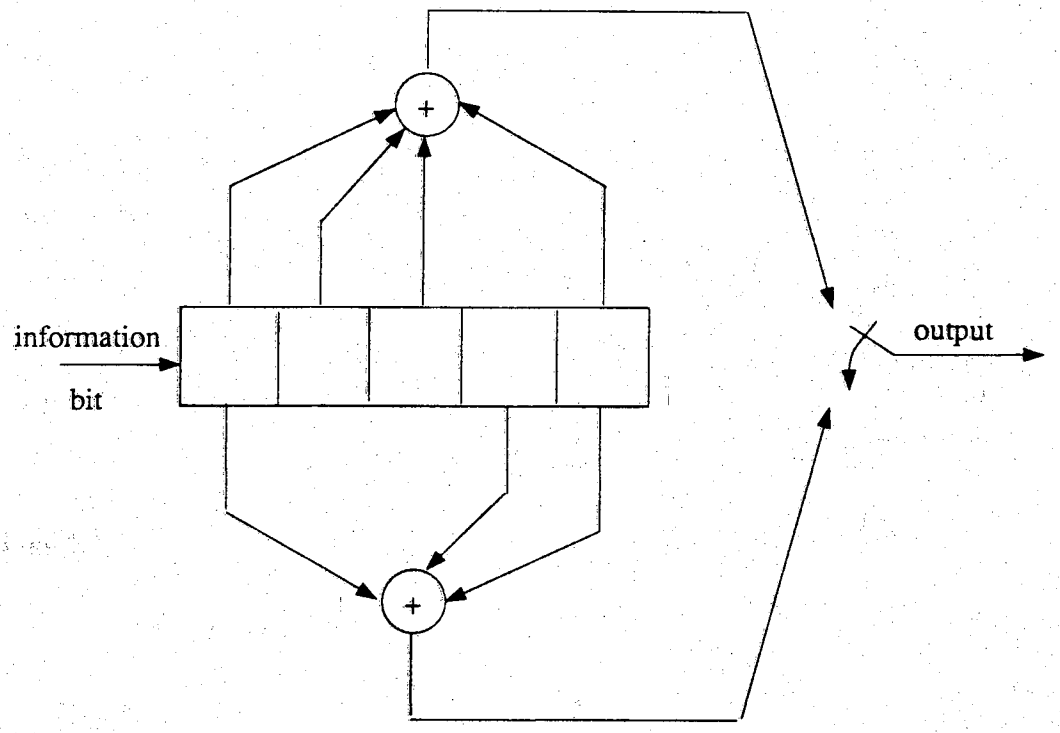
Let j denotes the node on the correct path at depth j and let

$$C_j = \text{the number of nodes in } \mathcal{X}_j$$

where $\mathcal{X}_j$ is the j'th incorrect subset (recall Section 3.2.1.) Recall that C_j is simply the number of tree nodes examined by the stack algorithm in order to make a correct branching decision at depth j. Furthermore, recall that C_j is a random variable and that

its distribution is one of the key quantities that characterize the performance of the stack algorithm. In this chapter, our main goal is to estimate the *distribution of computation*. That is, the distribution of C_j. It is noted that the techniques which we shall discuss here can be used to estimate other key parameters such as the average number of metric computations per correct decision. That is, $E[C_j]$.

The organization of this chapter is as follows. In Section 5.2. we shall briefly review the importance sampling background for coded communications systems. Specifically, we will present the required importance sampling background that is needed to estimate the distribution of computation. This presentation also includes a discussion of some of the issues which are relevant to the estimation of the distribution of computation. The presentation of our simulation results will be considered next. For simplicity and in order to gain valuable insight about our importance sampling techniques, we will first consider the binary symmetric channel (BSC) case exclusively. Once this case is fully presented, we will then consider the additive white gaussian noise (AWGN) channel case. As we have mentioned previously, we are not interested in decision errors (we shall consider that in Chapter 7.) Specifically, all of our estimates throughout this chapter will be conditioned on the event E_j where E_j is the event that correct decisions at both depth j and j+1 have been made once the search is terminated. For notational simplicity, we shall drop the conditioning on E_j in our notation. Hence, for a given $M \geq 1$, we will write $\mathcal{P}(C_j \geq M)$ instead of $\mathcal{P}(C_j \geq M \mid E_j)$. We should finally mention that all of the simulation results that we shall present in this chapter were obtained using the modified stack algorithm simulation (MSAS.) The only exception is at the end of Section 5.3.3 where we compare the performance of the MSAS with the stack algorithm.

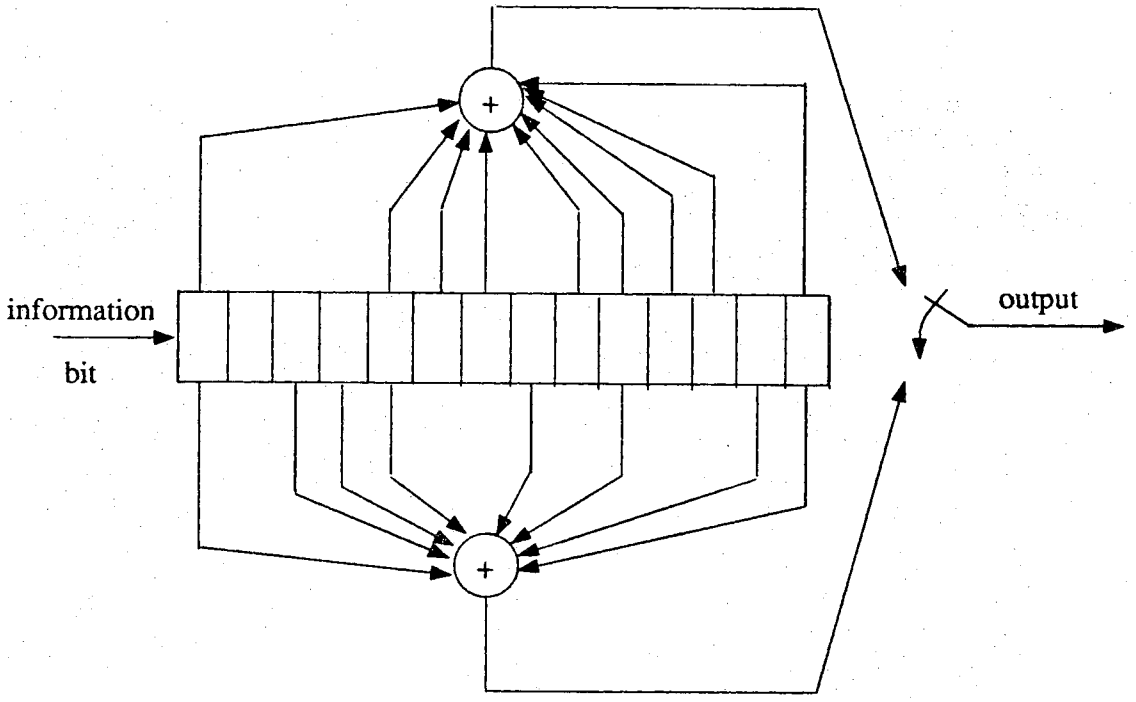


$g_1 = 35$ (octal)

$g_2 = 23$ (octal)

states = 16

Figure 5.1: A rate 1/2 convolutional encoder with constraint length 5.

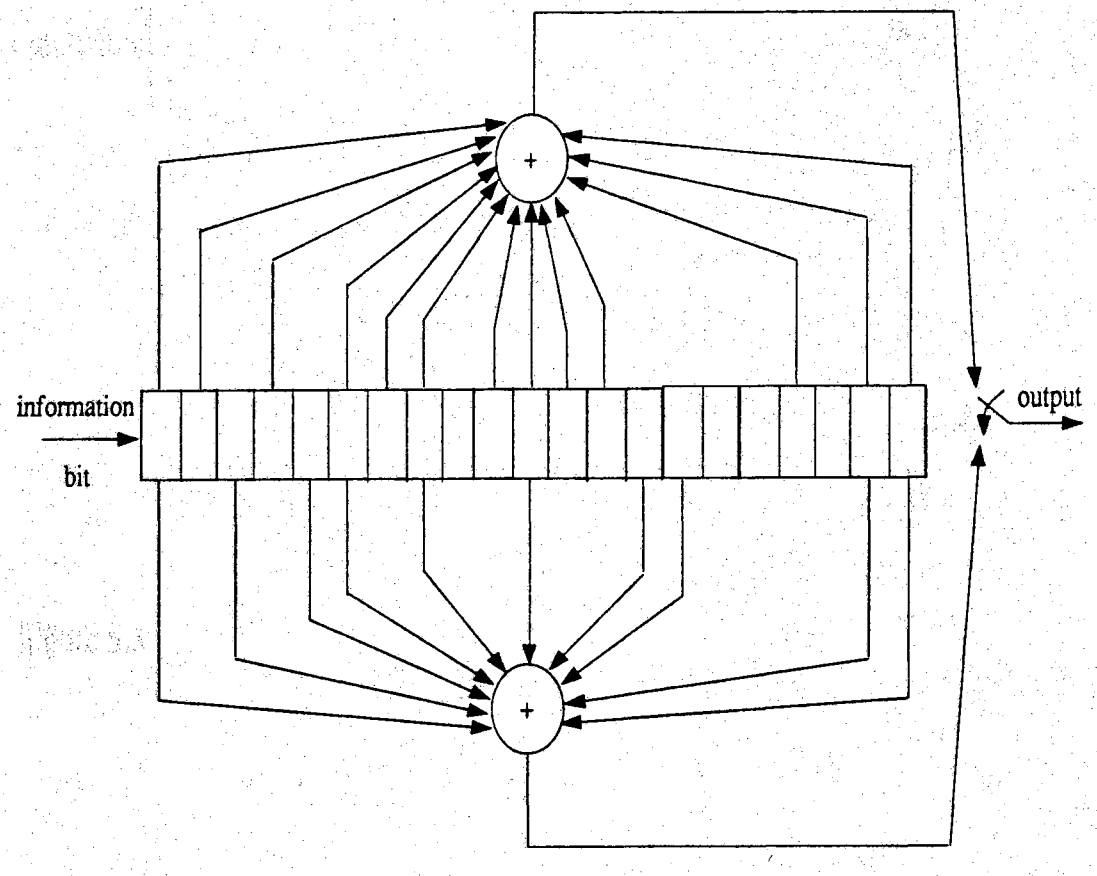


$g_1 = 21675$ (octal)

$g_2 = 27123$ (octal)

states = 8192

Figure 5.2: A rate 1/2 Convolutional encoder with constraint length 14.



g_1 = 6567413 (octal)

g_2 = 5322305 (octal)

states = 1048576

Figure 5.3: A rate 1/2 Convolutional encoder with constraint length 21.

5.2 Importance Sampling

Consider the memoryless coding channel of Section 4.2. Recall that for a given sequence of channel input symbols $\mathbf{u}^c = (u_1^c, u_2^c, ..., u_n^c)$, the sequence of output symbols $\mathbf{V} = (V_1, V_2, ..., V_n)$ consist of independent random samples with joint density

$$\mathbf{f}(\mathbf{v} \mid \mathbf{u}^c) = \prod_{k=1}^{n} f_k(v_k \mid u_k^c) \tag{5.1}$$

where $f_k(v_k \mid u_k^c)$ is the channel transition density function.

Now for any $M \geq 1$, let

$$P_M \triangleq \mathcal{P}(C_j \geq M)$$

and

$$I_M(.) \triangleq \text{the indicator of the event } \{ C_j \geq M \}.$$

Then the *important sampling estimator* of $P_M = E[I_M(.)]$ is

$$\hat{P}_M = \frac{1}{L} \sum_{\ell=1}^{L} I_M(\mathbf{V}^{(\ell)}) \, w(\mathbf{V}^{(\ell)} \mid \mathbf{u}^c) \tag{5.2}$$

where

$$w(\mathbf{v} \mid \mathbf{u}^c) = \frac{\mathbf{f}(\mathbf{v} \mid \mathbf{u}^c)}{\mathbf{f}^*(\mathbf{v} \mid \mathbf{u}^c)} \tag{5.3}$$

$$= \prod_{k=1}^{n} \frac{\mathbf{f}_k(v_k \mid u_k^c)}{\mathbf{f}_k^*(v_k \mid u_k^c)}$$

and $\mathbf{V}^{(1)}, ..., \mathbf{V}^{(L)}$ are independent random samples from the *importance sampling distribution* $\mathbf{f}^*(.|.)$. The likelihood ratio $w(.|.)$ is the *importance sampling weight.* Again note that if $\mathbf{f}_k^*(\mathbf{v} \mid \mathbf{u}^c) = \mathbf{f}_k(\mathbf{v} \mid \mathbf{u}^c)$ then $w(\mathbf{v} \mid \mathbf{u}^c) \equiv 1$ and the sample mean estimator (5.2) is reduced to the ordinary Monte Carlo relative frequency estimator.

Let $E^*[.]$ and $\text{var}^*[.]$ denote the expectation and variance operations for the simulation density. Then because the simulation data $V^{(1)}, \ldots, V^{(L)}$ are independent random samples from the simulation density $\mathbf{f^*(v \mid u^c)}$, it follows that

$$\begin{aligned} E^*[\,\hat{P}_M\,] &= E^*[\,I_M(V^{(\ell)})\, w(V^{(\ell)} \mid \mathbf{u^c})\,] \\ &= \iint I_M(\mathbf{v})\, w(\mathbf{v} \mid \mathbf{u^c})\, \mathbf{f^*(v \mid u^c)}\, d\mathbf{v} \\ &= \iint I_M(\mathbf{v})\, \frac{\mathbf{f(v \mid u^c)}}{\mathbf{f^*(v \mid u^c)}}\, \mathbf{f^*(v \mid u^c)}\, d\mathbf{v} \\ &= P_M. \end{aligned}$$

Consequently, the importance sampling estimator (5.2) is unbiased.

A similar computation indicates that the variance of $\hat{P}_M$ is

$$\begin{aligned} \text{var}^*[\hat{P}_M] &= \frac{1}{L}\, \text{var}^*\left[\, I_M(\mathbf{V})\, w(\mathbf{V} \mid \mathbf{u^c}) \,\right] \\ &= \frac{1}{L}\left[\iint I_M(\mathbf{v}) \left[\frac{\mathbf{f(v \mid u^c)}}{\mathbf{f^*(v \mid u^c)}} \right]^2 \mathbf{f^*(v \mid u^c)}\, d\mathbf{v} - (P_M)^2 \right] \end{aligned} \tag{5.4}$$

Expression (5.4) indicates that a good choice of the simulation density $\mathbf{f^*(v \mid u^c)}$ will tend to be large relative to $\mathbf{f(v \mid u^c)}$. This will tend to minimize (5.4), and hence, diminish the estimator's variance for a fixed L, or equivalently, reduce the number of simulation runs L for a given variance or accuracy.

On a first exposure, it may seem reasonable to deduce that the best importance sampling models will be stationary and memoryless, especially if the true channel model is stationary and memoryless. However, our experience indicates that this is not the case. To properly understand this, first note that a good importance sampling simulation model tend to increase the relative frequency of the "important" events.

Thus, it should "bias" or force the stack algorithm to search nodes in the j'th incorrect subtree $\mathcal{N}_j$. This suggests that we should begin our importance sampling simulations with a very noisy operating condition to ensure a high percentage of error events. However, if we continue the simulation runs with a noisy channel, then these simulations will tend to produce a lot of errors in other incorrect subtrees and the desired incorrect subset may become excessively large. Our desire is to emphasize only the "important events," that is, those error events which tend to hypothesize paths in the j'th incorrect subtree $\mathcal{N}_j$. Thus, it is clear that the simulation channel model should start out with a high noise operating condition in order to initiate the error events of interest, and then in some fashion be programmed to become less noisy as time progresses.

Now consider the importance sampling estimator (5.2) and let

$$\hat{V}^*(\mathbf{V}^{(\emptyset)} \mid \mathbf{u}^c) \triangleq \frac{1}{L} \left[\sum_{\emptyset=1}^{L} (w(\mathbf{V}^{(\emptyset)} \mid \mathbf{u}^c)^2 \, I_M(\mathbf{V}^{(\emptyset)}) \right] - (\hat{P}_M)^2 \tag{5.5}$$

because the simulation data are independent, it follows that $\frac{1}{L}\left(\sum_{\emptyset=1}^{L} (w(\mathbf{V}^{(\emptyset)} \mid \mathbf{u}^c)^2 \, I_M(\mathbf{V}^{(\emptyset)}) \right)$ is an unbiased estimator for $E^*[\, (w(\mathbf{V}^{(\emptyset)} \mid \mathbf{u}^c) \, I_M(\mathbf{V}^{(\emptyset)}))^2 \,]$ and hence,

$$\hat{V}_M^* = \frac{\hat{V}^*(\mathbf{V}^{(\emptyset)} \mid \mathbf{u}^c)}{L} \tag{5.6}$$

is a good empirical estimate of $\text{var}^*[\hat{P}_M]$.

The *nominal variance;* that is, the variance for the ordinary Monte Carlo estimator can be also estimated using good estimates of $\hat{P}_M$. Noting that $I_M(\mathbf{V}^{(\emptyset)})$ is a 0–1 Bernoulli random variable, it follows that the appropriate estimator for $\text{var}(\, I_M(\mathbf{V}^{(\emptyset)}) \,)$ is given by

$$\hat{V}(V^{(0)} | u^c) = \hat{P}_M (1 - \hat{P}_M). \tag{5.7}$$

Consequently, the estimator for the nominal variance is

$$\hat{V}_M = \frac{\hat{V}(V^{(0)} | u^c)}{L} \tag{5.8}$$

Now given expression (5.6), we can then estimate the *accuracy* that is, the estimator standard deviation as a percentage of $\hat{P}_M$. The estimates for the accuracy can be computed using both $\hat{P}_M$ and the sample variance estimator (5.6) as follows:

$$\text{Accuracy estimate} \triangleq \frac{\sqrt{\hat{V}_M^*}}{\hat{P}_M}. \tag{5.9}$$

It is now convenient to let L_{MC} and L_{IS} denote the numbers of simulation runs required to estimate P_M to a specified accuracy, respectively for ordinary Monte Carlo and with importance sampling. Then the *relative efficiency gain (reg)* can be defined as:

$$\text{reg} \triangleq \frac{L_{MC}}{L_{IS}}. \tag{5.10}$$

The relative efficiency gains can be estimated using the sample variance estimates (5.6) and (5.8) as follows. Suppose that an accuracy is specified by a variance v. Then, for L_{IS} simulations the importance sampling variance is $\hat{V}_M^*/L_{IS}$. Hence, to obtain a given variance v, the required number of importance sampling simulation runs is $L_{IS} = \hat{V}_M^*/v$. Likewise, the number of ordinary Monte Carlo simulation runs which are required to obtain a given variance v is $L_{MC} = \hat{V}_M/v$. Consequently, from (5.10) we conclude that

$$\text{reg estimate} = \frac{\hat{V}_M}{\hat{V}_M^*}. \tag{5.11}$$

We should mention that the relative efficiency gains can sometimes be misleading because they do not take into account a number of factors. For example, recall that efficient importance sampling schemes should cause errors to occur more often. Consequently, it follows that the same number of simulation runs will require more metric computations with importance sampling than with ordinary Monte Carlo. Thus, a comparison based on required number of simulations runs will be biased toward importance sampling. As a consequence, it is sometimes appropriate to take as a basic figure of merit the *importance sampling efficiency* (η_{is})

$$\eta_{is} \triangleq \frac{C_{MC}}{C_{IS}}. \tag{5.12}$$

$$= \frac{L_{MC}\, E[\, T\,]}{L_{IS}\, E^*[\, T^*\,]} \tag{5.13}$$

where, $E[\, T\,]$ and $E^*[\, T^*\,]$ are respectively the average number of metric computations for ordinary Monte Carlo and with importance sampling. Likewise, C_{MC} and C_{IS} denote the expected number of metric computations required to estimate $\hat{P}_M$ to a specified accuracy with ordinary Monte Carlo and importance sampling respectively.

By letting $T^{(\ell)}$ be the total number of nodes examined in the ℓ'th simulation, it follows that a good estimate for $E[\, T\,]$ is

$$\bar{T} = \frac{1}{L} \sum_{l=1}^{L} T^{(\ell)}\, W(\mathbf{V}^{(\ell)} \,|\, \mathbf{u}^c). \tag{5.14}$$

Likewise, for the importance sampling simulation model, the estimate of $E^*[\, T^*\,]$ is easily computed as

$$\bar{T}^* = \frac{1}{L} \sum_{\ell=1}^{L} T^{(\ell)} \tag{5.15}$$

Observe that an identical computation to that given in the proof of the unbiasedness of

$\hat{P}_M$ proves that $\overline{T}$ is an unbiased estimator of E[T].

Finally by recalling (5.12) we conclude that the appropriate estimator for the importance sampling efficiency as defined in (5.13) is

$$\hat{\eta}_{is} = \frac{\hat{V}_M \overline{T}}{\hat{V}_M^* \overline{T}^*}. \tag{5.16}$$

We conclude this section by discussing the termination of the simulation issue. Recall that for practical implementation of the stack algorithm, a termination strategy is always needed. To properly understand our termination strategy, recall that because of the randomness of the node metrics in the tree, it is entirely possible that the stack algorithm can mistakenly follow an incorrect path for some depth in the tree. However, because of the average behavior of the node metrics, incorrect paths will eventually become inactive as they get longer and longer. Consequently, discarding any node whose metric is very small compared to the TOS node metric would probably have a negligible effect on the stack algorithm performance. Understanding this is the key to our termination strategy. To state this strategy, few definitions are needed.

Assume that node j is the root node and suppose that a given simulation starts at t = 0. Next let

$$M_{TOS} \triangleq \text{TOS node metric.}$$

Furthermore, for any $\delta \in D_j$ (recall that D_j is the collection of all the direct descendent nodes of node j) let

$$\overline{M}_\delta(t) \triangleq \max \{ M_\eta : \eta \in S_\delta \cap \mathcal{S}_t \}$$

where $\mathcal{S}_t$ is the collection of all nodes in the stack at time t. Finally for a given $\Delta > 0$, we will say that the subtree S_δ is *inactive* if

$$\overline{M}_{\delta}(t) \leq M_{TOS}(t) - \Delta.$$

Now recall that we are not interested in decision errors. Specifically, all of our estimates will be conditioned on the event E_j. That is, on the event that correct decisions at both depth j and j+1. Keeping that in mind, we have adopted the following stopping rule:

The DELTA-Stopping Rule (For the ℓ'th simulation):

Initialize: Start the search at node j.

Simulate until only one subtree is active, call it S_{β_1}.

IF β_1 is a correct node **THEN**

IF $C_j \geq M$ **THEN**

$I_M(V^{(\ell)} | u^c) = 1.$

Stop.

ELSE

$I_M(V^{(\ell)} | u^c) = 0.$

Stop.

ENDIF

ELSE

$I_M(V^{(\ell)} | u^c) = 0.$

Stop.

END IF

5.3 Simulation Results

We are now ready to present our importance sampling simulation results for estimating the distribution of computation. That is, $\mathcal{P}(C_j \geq M)$ for $M \geq 1$. As we mentioned earlier, we will consider rate 1/2 convolutional codes with constraint lengths 5, 14 and 21 which operate on the binary symmetric channel and the AWGN channel. In addition to the importance sampling results, we shall also give some ordinary Monte Carlo estimates (MC). Throughout this chapter, the Monte Carlo simulation data were obtained using a total of 1,000,000 simulation runs. There are two main reasons for presenting these Monte Carlo simulation results. First we would like to show that for small values for M, ordinary Monte Carlo simulation of the $\mathcal{P}(C_j \geq M)$ works just fine. Hence, for such values, the ordinary Monte Carlo simulation estimates can be used as a verification of the accuracy of our importance sampling estimates. On the other hand, for large values of M, we will show that Monte Carlo simulations are simply not efficient enough to obtain any meaningful estimates. However, for such large values we shall see that by using our importance sampling schemes, we will achieve high computational efficiency gains along with accurate estimates.

We begin our presentation by considering the binary symmetric channel case.

5.3.1 The Binary Symmetric Channel Case

A. The Reference Path Method

In this section, we present some importance sampling simulation results which we have obtained via the reference path method (RPM). Recall that the basic idea behind the reference path method is is to design an importance sampling channel model which will tend to "force" or "trick" the stack algorithm into decoding a given reference path

$\mathbf{u}_N^r = (u_1^r, u_2^r, \ldots u_N^r)$ instead of the correct path $\mathbf{u}^c = (u_1^c, u_2^c, \ldots)$. As indicated in the previous chapter, reference paths of a given depth N can be found by an exhaustive search in such a way that their hamming distances from the correct path are minimized over all incorrect paths of depth N. For example, Figure 5.4 shows part of the tree code generated by the constraint length 14 convolutional code. A close observation to Figure 5.4 indicates that at depth 5; for instance, we have two minimum distance paths $\mathbf{u}^1 = (1\ 0\ 0\ 0\ 1)$ and $\mathbf{u}^2 = (1\ 0\ 0\ 1\ 1)$. Hence at depth 5, there are two candidate reference paths: $\mathbf{u}^1$ and $\mathbf{u}^2$.

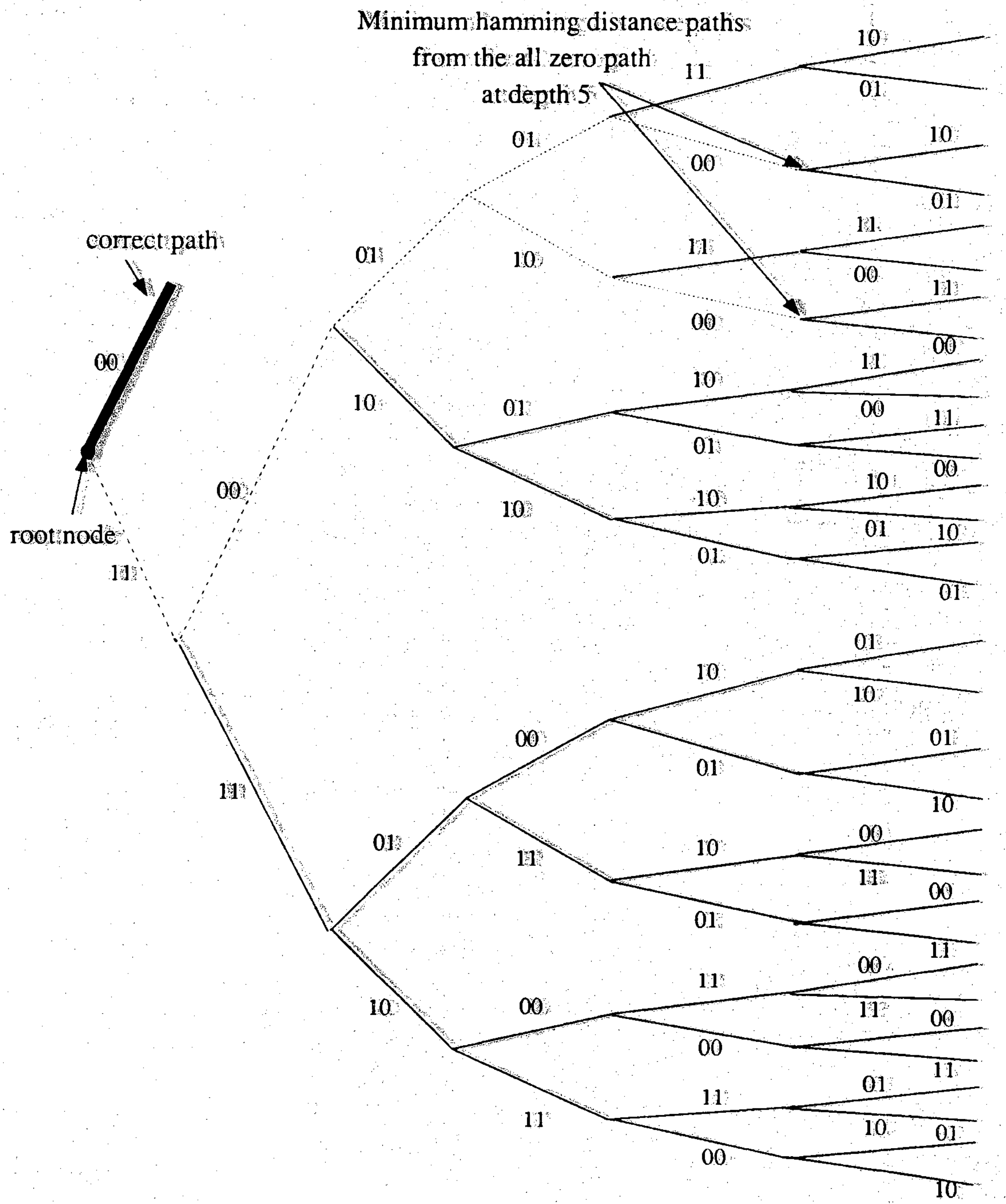


Figure 5.4: Tree code for the rate 1/2 and constraint length 14 convolutional encoder.

Let u_k^c and u_k^r denote respectively the code symbols associated with the correct path and a given reference path. Note that u_k^c and u_k^r are actually two-dimensional because the convolutional code rate is 1/2. That is, $u_k^c = (u_{k1}^c, u_{k2}^c)$ and $u_k^r = (u_{k1}^r, u_{k2}^r)$. Since the correct path is assumed to be the all-zero path, it follows that $u_k^c \equiv (-1,-1)$. On the other hand, all incorrect paths (which include the reference paths) are of the form $u_k^i \equiv (\pm 1, \pm 1)$. Now recall that the main objective of the reference path method is to "force" the stack algorithm into decoding the reference path instead of the correct path. To do so, we have designed the reference path simulation scheme in such a way that 1) we do not bias the channel when $u_{ki}^c = u_{ki}^r$; and 2) we use a uniform biasing for the remaining instances when $u_{ki}^c \neq u_{ki}^r$ (i=1 or 2.) Specifically, we use a crossover probability of $\varepsilon_{ki}^* = 1/2$ at the instances when $u_{ki}^c \neq u_{ki}^r$ (i=1,2.) Once the data is generated up to N, the depth of the reference path, no more biasing is done. Thus, the reference path method is based on a non-stationary memoryless BSC model which is characterized by a time varying crossover probability

$$\varepsilon_{ki}^* = \begin{cases} 1/2 & \text{if } u_{ki}^c \neq u_{ki}^r \\ \varepsilon & \text{if } u_{ki}^c = u_{ki}^r \end{cases} \tag{5.17}$$

for i=1 or 2 and $k \leq N$. For $k > N$ we use $\varepsilon_{ki}^* = \varepsilon$. It should be clear that with roughly probability 1/2 the reference path will be examined up to depth N.

Some of the simulation results for the reference path method are summarized in Tables 5.1-5.5. The first two columns give the estimates of the distribution of computation and the accuracy using the reference path method. The last two columns give the estimates of the distribution of computation and the accuracy using ordinary Monte Carlo.

Table 5.1: The distribution of computation estimates for the constraint length 5 code operating on the BSC with $\epsilon = .01$. $L = 250{,}000$ for the RPM and $L = 1{,}000{,}000$ for MC.

	RPM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 3)$	$.116 \times 10^{-1}$	.70 %	$.115 \times 10^{-1}$	.93 %
$\mathcal{P}(C_j \geq 15)$	$.369 \times 10^{-3}$	2.42 %	$.42 \times 10^{-3}$	0.05 %
$\mathcal{P}(C_j \geq 24)$	$.318 \times 10^{-3}$	1.84 %	$.352 \times 10^{-3}$	5.33 %
$\mathcal{P}(C_j \geq 30)$	$.117 \times 10^{-3}$	1.48 %	$.131 \times 10^{-3}$	8.73 %
$\mathcal{P}(C_j \geq 33)$	$.116 \times 10^{-3}$	1.46 %	$.13 \times 10^{-3}$	8.77 %
$\mathcal{P}(C_j \geq 42)$	$.187 \times 10^{-4}$	6.86 %	$.16 \times 10^{-4}$	25.00 %
$\mathcal{P}(C_j \geq 63)$	$.723 \times 10^{-5}$	2.48 %	$.8 \times 10^{-5}$	35.36 %
$\mathcal{P}(C_j \geq 69)$	$.687 \times 10^{-5}$	2.51 %	$.8 \times 10^{-5}$	35.36 %
$\mathcal{P}(C_j \geq 78)$	$.304 \times 10^{-5}$	3.57 %	$.2 \times 10^{-5}$	70.72 %
$\mathcal{P}(C_j \geq 90)$	$.278 \times 10^{-5}$	3.67 %	$.2 \times 10^{-5}$	70.72 %

Table 5.2: The distribution of computation estimates for the constraint length 5 code operating on the BSC with $\varepsilon = .005$. $L = 250{,}000$ for the RPM and $L = 1{,}000{,}000$ for MC.

	RPM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 3)$	$.540 \times 10^{-2}$	0.60 %	$.547 \times 10^{-2}$	1.35 %
$\mathcal{P}(C_j \geq 6)$	$.147 \times 10^{-3}$	2.10 %	$.149 \times 10^{-3}$	8.19 %
$\mathcal{P}(C_j \geq 12)$	$.136 \times 10^{-3}$	2.16 %	$.136 \times 10^{-3}$	8.58 %
$\mathcal{P}(C_j \geq 30)$	$.266 \times 10^{-4}$	1.35 %	$.230 \times 10^{-4}$	20.96 %
$\mathcal{P}(C_j \geq 45)$	$.150 \times 10^{-5}$	7.64 %	$.300 \times 10^{-5}$	57.80 %
$\mathcal{P}(C_j \geq 54)$	$.784 \times 10^{-6}$	2.25 %	$.100 \times 10^{-5}$	100.0 %
$\mathcal{P}(C_j \geq 72)$	$.296 \times 10^{-6}$	3.67 %	$.100 \times 10^{-5}$	100.0 %
$\mathcal{P}(C_j \geq 78)$	$.288 \times 10^{-6}$	3.72 %	$.100 \times 10^{-5}$	100.0 %
$\mathcal{P}(C_j \geq 84)$	$.278 \times 10^{-6}$	3.79 %	$.100 \times 10^{-5}$	100.0 %
$\mathcal{P}(C_j \geq 90)$	$.273 \times 10^{-6}$	3.82 %	$.100 \times 10^{-6}$	100.0 %

Table 5.3: The distribution of computation estimates for the constraint length 14 code operating on the BSC with $\varepsilon = .01$. $L = 250{,}000$ for the RPM and $L = 1{,}000{,}000$ for MC.

	RPM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 3)$	$.116 \times 10^{-1}$	0.70 %	$.115 \times 10^{-1}$	0.93 %
$\mathcal{P}(C_j \geq 18)$	$.172 \times 10^{-3}$	2.36 %	$.197 \times 10^{-3}$	7.12 %
$\mathcal{P}(C_j \geq 24)$	$.157 \times 10^{-3}$	2.29 %	$.167 \times 10^{-3}$	7.74 %
$\mathcal{P}(C_j \geq 33)$	$.393 \times 10^{-4}$	6.26 %	$.39 \times 10^{-4}$	16.02 %
$\mathcal{P}(C_j \geq 42)$	$.242 \times 10^{-4}$	6.63 %	$.24 \times 10^{-4}$	20.41 %
$\mathcal{P}(C_j \geq 51)$	$.209 \times 10^{-4}$	6.75 %	$.19 \times 10^{-4}$	22.93 %
$\mathcal{P}(C_j \geq 66)$	$.138 \times 10^{-4}$	7.31 %	$.12 \times 10^{-4}$	28.93 %
$\mathcal{P}(C_j \geq 75)$	$.109 \times 10^{-4}$	7.74 %	$.70 \times 10^{-5}$	37.74 %
$\mathcal{P}(C_j \geq 81)$	$.887 \times 10^{-5}$	7.26 %	$.60 \times 10^{-5}$	40.82 %
$\mathcal{P}(C_j \geq 87)$	$.711 \times 10^{-5}$	7.98 %	$.50 \times 10^{-5}$	44.64 %

Table 5.4: The distribution of computation estimates for the constraint length 14 code operating on the BSC with $\varepsilon = .005$. $L = 500{,}000$ for the RPM and $L = 1{,}000{,}000$ for MC.

	RPM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 3)$	$.537 \times 10^{-2}$	0.35 %	$.553 \times 10^{-2}$	1.34 %
$\mathcal{P}(C_j \geq 6)$	$.204 \times 10^{-3}$	1.56 %	$.203 \times 10^{-3}$	7.02 %
$\mathcal{P}(C_j \geq 12)$	$.825 \times 10^{-4}$	1.01 %	$.155 \times 10^{-3}$	8.03 %
$\mathcal{P}(C_j \geq 18)$	$.319 \times 10^{-4}$	1.92 %	$.400 \times 10^{-4}$	15.82 %
$\mathcal{P}(C_j \geq 30)$	$.419 \times 10^{-5}$	8.26 %	$.500 \times 10^{-5}$	44.72 %
$\mathcal{P}(C_j \geq 48)$	$.240 \times 10^{-5}$	7.46 %	$.200 \times 10^{-5}$	71.43 %
$\mathcal{P}(C_j \geq 57)$	$.180 \times 10^{-6}$	6.36 %	$.100 \times 10^{-5}$	100 %
$\mathcal{P}(C_j \geq 66)$	$.995 \times 10^{-6}$	8.16 %	$.100 \times 10^{-5}$	100 %
$\mathcal{P}(C_j \geq 87)$	$.622 \times 10^{-6}$	2.25 %	$.100 \times 10^{-5}$	100 %
$\mathcal{P}(C_j \geq 90)$	$.495 \times 10^{-6}$	2.44 %	$.100 \times 10^{-5}$	100 %

Table 5.5: The distribution of computation estimates for the constraint length 21 code operating on the BSC with $\varepsilon = .005$. $L = 300{,}000$ for the RPM and $L = 1{,}000{,}000$ for MC.

	RPM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 3)$	$.539 \times 10^{-2}$	0.34 %	$.542 \times 10^{-2}$	1.35 %
$\mathcal{P}(C_j \geq 9)$	$.151 \times 10^{-3}$	1.47 %	$.141 \times 10^{-3}$	8.42 %
$\mathcal{P}(C_j \geq 12)$	$.133 \times 10^{-3}$	1.54 %	$.13 \times 10^{-3}$	8.77 %
$\mathcal{P}(C_j \geq 21)$	$.319 \times 10^{-4}$	1.65 %	$.26 \times 10^{-4}$	19.61 %
$\mathcal{P}(C_j \geq 33)$	$.174 \times 10^{-5}$	2.53 %	$.10 \times 10^{-5}$	100 %
$\mathcal{P}(C_j \geq 48)$	$.120 \times 10^{-5}$	2.92 %	$.10 \times 10^{-5}$	100 %
$\mathcal{P}(C_j \geq 57)$	$.796 \times 10^{-6}$	3.33 %	$.10 \times 10^{-5}$	100 %
$\mathcal{P}(C_j \geq 69)$	$.516 \times 10^{-6}$	4.28 %	$.10 \times 10^{-5}$	100 %
$\mathcal{P}(C_j \geq 81)$	$.360 \times 10^{-6}$	4.69 %	$.10 \times 10^{-5}$	100 %
$\mathcal{P}(C_j \geq 90)$	$.336 \times 10^{-6}$	4.77 %	$.10 \times 10^{-5}$	100 %

B. The Partitioning Method

We shall know present and demonstrate the power and accuracy of the partitioning method (PM) for the rate 1/2 and constraint lengths 5 and 14 convolutional codes operating on a binary symmetric channel that is characterized by a crossover probability ε.

Define N^* to be the depth at which the minimum value along the correct path (after depth j) occurs. Then from the partitioning method analysis in Section 4.5.3, recall that the basic idea behind the partitioning scheme is to partition the underlying probability space by the events $\{N^* = m\}$, m= 0,1,2,..., and hence

$$\mathcal{P}(C_j \geq M) = \sum_{m=0}^{\infty} \mathcal{P}(C_j \geq M\,;\, N^* = m). \tag{5.18}$$

For m= 0,1,2.., we bias the simulation model to estimate the terms in (5.18) separately. The simulation model is given by

$$\varepsilon_k^o = \begin{cases} e^{.5\alpha^c Z_k^c - \Lambda^c(\alpha^c/2)}\,\varepsilon & \text{if } k \leq m \\ \varepsilon & \text{otherwise} \end{cases} \tag{5.19}$$

where

$$\Lambda^c(\alpha^c) = 2\left[\ln(\,(1-\varepsilon)e^{\alpha^c a} + \varepsilon e^{-\alpha^c b}\,) - \alpha^c R\right] \tag{5.20}$$

and Z_k^c is the correct path (fano) branch metric. Recall that the constants a and b in the above equation were defined in the previous chapter as

$$a \triangleq \ln(2(1-\varepsilon)) \quad \text{and} \quad b \triangleq -\ln(2\varepsilon). \tag{5.21}$$

Furthermore, recall that the optimal value of α^c that is predicted by the partitioning method analysis should be chosen so that

$$E[\,\exp(\,.5\alpha^c Z_k^c\,]\, = e^{\Lambda^c(\alpha^c/2)} \tag{5.22}$$

is minimized.

Few comments on this scheme are in order here. First, notice that for $k \leq m$ the simulation model basically uses a uniform biasing. Specifically, for $k \leq m$, $\varepsilon_k^o \equiv \varepsilon_o$ where $\varepsilon_o = \exp(-\alpha^c(b+R) - \Lambda^c(\alpha^c/2))\varepsilon$. Second, recall that for a given m, each simulation should "force" the minimum value along the correct path after depth j to occur at depth m. Keeping that in mind, we may conclude that by biasing the simulation model when $m = 0$, we may force the correct path minimum to occur at depth $m \neq 0$. Hence, it seems reasonable to deduce that for $m = 0$ it is better to set $\varepsilon_k^o \equiv \varepsilon$. That is, when $m = 0$ we should not bias the channel. Finally, recall that the simulation model (5.19) is not an optimal simulation scheme for estimating the distribution of computation. Specifically, the partitioning model was derived by minimizing an upper bound of the variance of $P_\delta = \mathcal{P}($ node δ hypothesized.) Hence, it is not necessarily true that the partitioning simulation model is the optimal simulation scheme for estimating the distribution of computation. More importantly, it is reasonable to deduce that by using values of ε_k^o which are different from the ones predicted by the partitioning method analysis may lead to better computational efficiencies. It turns out that this last statement is in fact true. Specifically, suppose that instead of using ε_k^o as specified by the simulation model (5.19), we use another crossover probability ε_k^{**}. The value of ε_k^{**} is obtained in such a way that the computational efficiency for estimating $\mathcal{P}(C_j \geq M ; N^* = k)$ is maximized. By doing that, we have found that for small values of k, ε_k^{**} and ε_k^o are not equal in general. However, it turns out that as $k \uparrow \infty$, the optimal values of ε_k^{**} seem to be converging to ε_k^o. Thus, it does appear that the partitioning method is in a sense an asymptotically optimal simulation scheme for estimating the distribution of computation. Figures 5.5-5.7 illustrate this by showing the relative efficiency gains (erg) for the PM

estimates of $\mathcal{P}(C_j \geq M\,;\, N^* = m)$ as a function of the simulation crossover probability ε^*. In each of these figures m is kept fixed and the erg plots correspond to the estimates of $\mathcal{P}(C_j \geq M\,;\, N^* = m)$ for various values of M. In all of these figures, $\varepsilon = .01$ and the code is the constraint length 5 code. Notice that as m increases, the optimal values of the crossover probabilities that maximize the reg tend to converge to ε_o which is equal to .1373 in this case.

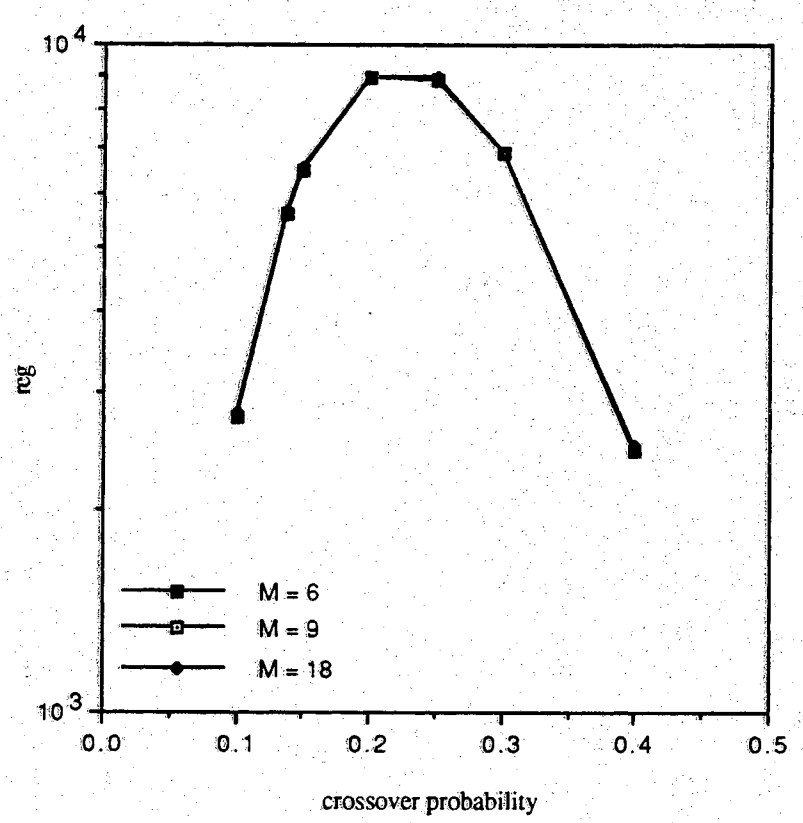


Figure 5.5: The relative efficiency gains (reg) for the PM estimates of $\mathcal{P}(C_j \geq M\,;\, N^* = 9)$ as a function of the simulation crossover probability.

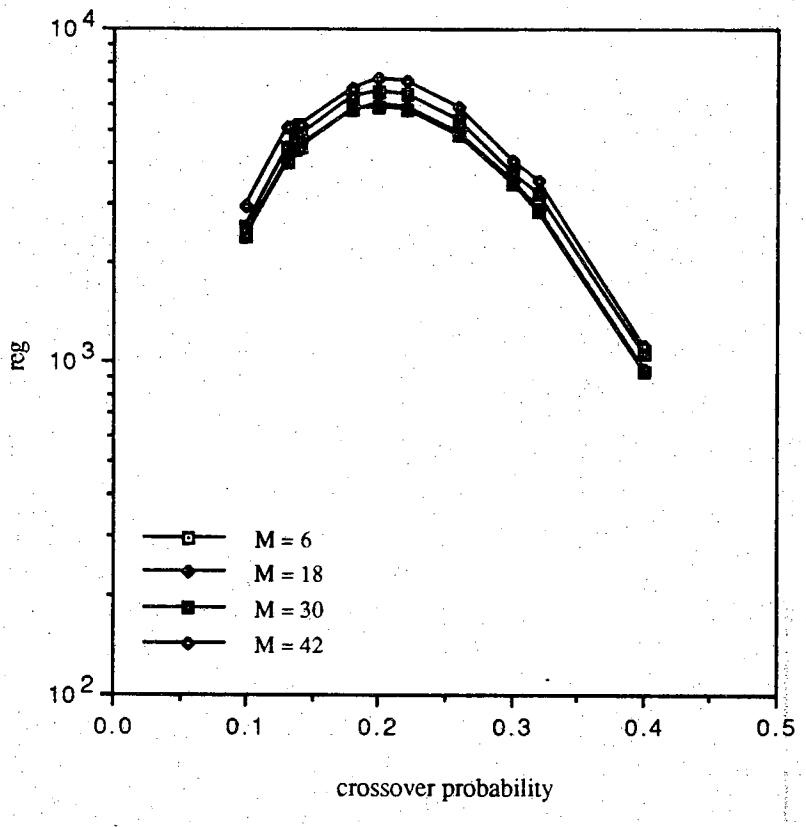


Figure 5.6: The relative efficiency gains (reg) for the PM estimates of $\mathcal{P}(C_j \geq M\,;\, N^* = 11)$ as a function of the simulation crossover probability.

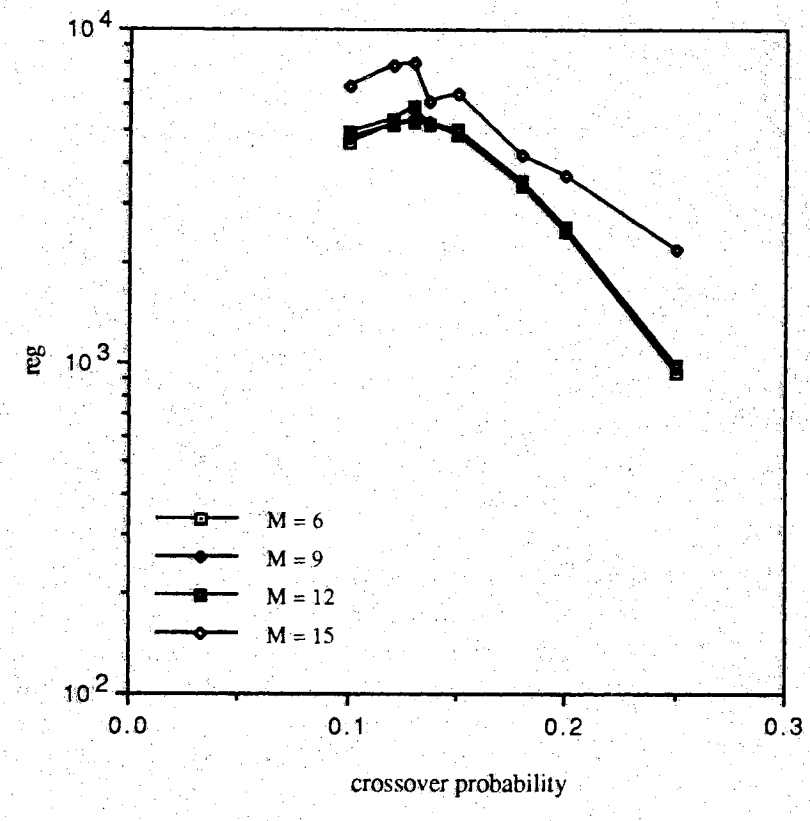


Figure 5.7: The relative efficiency gains (reg) for the PM estimates of $\mathcal{P}(C_j \geq M ; N^* = 20)$ as a function of the simulation crossover probability.

We are now ready to present the partitioning method simulation data. These results show that the partitioning method simulation technique is indeed an extremely efficient scheme for estimating the distribution of computation. We begin by considering the case where $\varepsilon = .01$. In this case, for a given $\{ N^* = m \}$, $\varepsilon_k^o = .1373$ for all $k \leq m$. The following table shows the distribution of computation for the constraint length 5 convolutional code. Again as in the reference path method, Monte Carlo simulation results are also included. We shall refer to the simulation model used to compute the estimates in Table 5.6 as the partitioning method model 1 (PM Model 1.) This model is characterized by

$$\varepsilon_k^o = \begin{cases} \varepsilon & m = 0 \\ .1373 & \text{if } k \leq m \\ \varepsilon & \text{otherwise} \end{cases} \tag{5.23}$$

The partitioning method estimates in Tables 5.6 and 5.8-5.10 are based on $L = 300{,}000$ simulation runs. Specifically, we use 20,000 simulation runs for each value of m ($m = 0, 1, ..., 15$.) On the other hand, recall that the Monte Carlo estimates are based on a 1 million simulation runs.

Table 5.6: The distribution of computation estimates for the constraint length 5 code operating on the BSC with $\varepsilon = .01$. $L = 300{,}000$ for PM Model 1 and $L = 1{,}000{,}000$ for MC.

	PM Model 1	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 6)$	$.689 \times 10^{-3}$	1.79 %	$.762 \times 10^{-3}$	3.62 %
$\mathcal{P}(C_j \geq 12)$	$.589 \times 10^{-3}$	2.05 %	$.644 \times 10^{-3}$	3.93 %
$\mathcal{P}(C_j \geq 21)$	$.346 \times 10^{-3}$	2.67 %	$.388 \times 10^{-3}$	5.08 %
$\mathcal{P}(C_j \geq 42)$	$.206 \times 10^{-4}$	3.86 %	$.16 \times 10^{-4}$	25.00 %
$\mathcal{P}(C_j \geq 57)$	$.924 \times 10^{-5}$	5.10 %	$.8 \times 10^{-5}$	35.33 %
$\mathcal{P}(C_j \geq 66)$	$.822 \times 10^{-5}$	5.63 %	$.7 \times 10^{-5}$	37.80 %
$\mathcal{P}(C_j \geq 72)$	$.579 \times 10^{-5}$	6.23 %	$.3 \times 10^{-5}$	57.74 %
$\mathcal{P}(C_j \geq 81)$	$.405 \times 10^{-5}$	5.81 %	$.2 \times 10^{-5}$	70.72 %
$\mathcal{P}(C_j \geq 87)$	$.374 \times 10^{-5}$	6.12 %	$.2 \times 10^{-5}$	70.72 %
$\mathcal{P}(C_j \geq 90)$	$.352 \times 10^{-5}$	6.42 %	$.2 \times 10^{-5}$	70.72 %

We shall now present some simulation results that have been obtained using another partitioning method simulation models. We shall refer to these models as the partitioning method model 2 (PM Model 2) and the partitioning method model 3 (PM Model 3.) These models are characterized in the following table.

Table 5.7: ε_k^* for the constraint length 5 code operating on the BSC with $\varepsilon = .01$.

ε_k^*		
k	PM Model 2	PM Model 3
0	ε	ε
1	.9	.55
2	.6	.55
3	.5	.5
4	.35	.4
5	.3	.3
6	.275	.3
7	.25	.3
8	.225	.2
9, 10, 11	.2	.2
12	.175	.18
13	.15	.18
14	.14	.15
15	.1373	.15

The values of ε_k^* in the above simulation models were obtained by trial an error in such a way that computational efficiency is maximized. It turns out that the performances of both of these models were very similar as seen from Table 5.8. However, we have found that overall the PM Model 3 yielded the most efficient

simulation in terms of computational efficiency gains.

Table 5.8: The distribution of computation estimates for the constraint length 5 code operating on the BSC with $\varepsilon = .01$. $L = 300{,}000$ for PM Model 2 and PM Model 3.

	PM Model 2	Accuracy	PM Model 3	Accuracy
$\mathcal{P}(C_j \geq 6)$	$.696 \times 10^{-3}$	0.84 %	$.705 \times 10^{-3}$	0.83 %
$\mathcal{P}(C_j \geq 12)$	$.591 \times 10^{-3}$	0.91 %	$.597 \times 10^{-3}$	0.89 %
$\mathcal{P}(C_j \geq 21)$	$.350 \times 10^{-3}$	1.14 %	$.344 \times 10^{-3}$	1.15 %
$\mathcal{P}(C_j \geq 42)$	$.201 \times 10^{-4}$	2.52 %	$.209 \times 10^{-4}$	2.58 %
$\mathcal{P}(C_j \geq 57)$	$.936 \times 10^{-5}$	3.29 %	$.897 \times 10^{-5}$	3.21 %
$\mathcal{P}(C_j \geq 66)$	$.811 \times 10^{-5}$	3.60 %	$.780 \times 10^{-5}$	3.52 %
$\mathcal{P}(C_j \geq 72)$	$.530 \times 10^{-5}$	4.12 %	$.502 \times 10^{-5}$	3.86 %
$\mathcal{P}(C_j \geq 81)$	$.369 \times 10^{-5}$	2.32 %	$.372 \times 10^{-5}$	2.19 %
$\mathcal{P}(C_j \geq 87)$	$.341 \times 10^{-5}$	2.33 %	$.339 \times 10^{-5}$	2.09 %
$\mathcal{P}(C_j \geq 90)$	$.317 \times 10^{-5}$	2.03 %	$.338 \times 10^{-5}$	2.08 %

Tables 5.9 and 5.10 also show some partitioning method simulation results for the constraints length 5 and 14 convolutional codes.

Table 5.9: The distribution of computation estimates for the constraint length 5 code operating on the BSC with $\varepsilon = .005$. $L = 300{,}000$ for the PM and $L = 1{,}000{,}000$ for MC.

	PM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 3)$	$.533 \times 10^{-2}$	1.14 %	$.547 \times 10^{-2}$	1.35 %
$\mathcal{P}(C_j \geq 6)$	$.156 \times 10^{-3}$	0.86 %	$.149 \times 10^{-3}$	8.19 %
$\mathcal{P}(C_j \geq 12)$	$.138 \times 10^{-3}$	0.93 %	$.136 \times 10^{-3}$	8.58 %
$\mathcal{P}(C_j \geq 30)$	$.277 \times 10^{-4}$	1.06 %	$.230 \times 10^{-4}$	20.96 %
$\mathcal{P}(C_j \geq 45)$	$.174 \times 10^{-5}$	3.74 %	$.300 \times 10^{-5}$	57.80 %
$\mathcal{P}(C_j \geq 54)$	$.976 \times 10^{-6}$	4.07 %	$.100 \times 10^{-5}$	100.0 %
$\mathcal{P}(C_j \geq 72)$	$.367 \times 10^{-6}$	2.01 %	$.100 \times 10^{-5}$	100.0 %
$\mathcal{P}(C_j \geq 78)$	$.338 \times 10^{-6}$	1.99 %	$.100 \times 10^{-5}$	100.0 %
$\mathcal{P}(C_j \geq 84)$	$.316 \times 10^{-6}$	1.98 %	$.100 \times 10^{-5}$	100.0 %
$\mathcal{P}(C_j \geq 90)$	$.301 \times 10^{-6}$	1.98 %	$.100 \times 10^{-5}$	100.0 %

Table 5.10: The distribution of computation estimates for the constraint length 14 code operating on the BSC with $\varepsilon = .01$. $L = 300{,}000$ for the PM and $L = 1{,}000{,}000$ for MC.

	PM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 3)$	$.113 \times 10^{-1}$	1.86 %	$.115 \times 10^{-1}$	0.93 %
$\mathcal{P}(C_j \geq 18)$	$.173 \times 10^{-3}$	0.86 %	$.197 \times 10^{-3}$	7.12 %
$\mathcal{P}(C_j \geq 24)$	$.152 \times 10^{-3}$	0.82 %	$.167 \times 10^{-3}$	7.74 %
$\mathcal{P}(C_j \geq 33)$	$.397 \times 10^{-4}$	2.27 %	$.39 \times 10^{-4}$	16.02 %
$\mathcal{P}(C_j \geq 42)$	$.245 \times 10^{-4}$	2.49 %	$.24 \times 10^{-4}$	20.41 %
$\mathcal{P}(C_j \geq 51)$	$.210 \times 10^{-4}$	2.86 %	$.19 \times 10^{-4}$	22.93 %
$\mathcal{P}(C_j \geq 66)$	$.148 \times 10^{-4}$	3.62 %	$.12 \times 10^{-4}$	28.93 %
$\mathcal{P}(C_j \geq 75)$	$.106 \times 10^{-4}$	3.62 %	$.70 \times 10^{-5}$	37.74 %
$\mathcal{P}(C_j \geq 81)$	$.844 \times 10^{-5}$	3.57 %	$.60 \times 10^{-5}$	40.82 %
$\mathcal{P}(C_j \geq 87)$	$.666 \times 10^{-5}$	3.82 %	$.50 \times 10^{-5}$	44.64 %

C. The M-Method

We shall now present and demonstrate the power and accuracy of the M-method (MM) for the rate 1/2 and constraint lengths 5 and 14 convolutional codes operating on the binary symmetric channel.

Recall that the M-method is basically a variation of the partitioning method discussed earlier. Specifically, in the M-method scheme we consider a discrete random variable M with probability mass function $p_m = \mathcal{P}(M = m)$ for $m = 1,2,...,J$. For $\ell = 1,2,...$, the random samples $M^{(\ell)}$ are generated from another probability mass function $p_m^* = \mathcal{P}^*(M = m)$ for $m = 1,2,...,J$. For each of these samples, the samples $\mathbf{V}^{(\ell)}$ are then generated from the conditional density $\mathbf{f}^*(\mathbf{v}|\mathbf{u}^c,m)$. Specifically, for a given m the samples $\mathbf{V}^{(\ell)}$ are generated from the density $\mathbf{f}^*(\mathbf{v}|\mathbf{u}^c,m)$ as in the partitioning simulation model. That is, for a given m, we will bias the channel model only up to depth m using a time varying biasing. As in the simulation model (5.19), recall that the MM simulation model also switches to the original channel density for depths $k \geq m$. In the sequel, we shall assume that $p_m^* \equiv p_m$. Then for any $N \geq 1$, the *M-method estimator* for $\mathcal{P}(C \geq N)$ is

$$\hat{P}_N = \frac{1}{L} \sum_{\ell=1}^{L} I_N(\mathbf{V}^{(\ell)})\, w(\mathbf{V}^{(\ell)}|\mathbf{u}^c,M^{(\ell)}) \tag{5.24}$$

where $I_N(.) \triangleq$ the indicator of the event $\{ C_j \geq N \}$ and

$$w(\mathbf{v}|\mathbf{u}^c,m) = \frac{\mathbf{f}(\mathbf{v}|\mathbf{u}^c)}{\mathbf{f}^*(\mathbf{v}|\mathbf{u}^c,m)} \tag{5.25}$$

$$= \prod_{k=1}^{n} \frac{\mathbf{f}_k(v_k|u_k^c)}{\mathbf{f}_k^*(v_k|u_k^c,m)}.$$

Observe that an identical computation to that given in the proof of the unbiasedness of the estimator (5.2) indicates that $\hat{P}_N$ is an unbiased estimator of $P_N = \mathcal{P}(C_j \geq N)$.

The following simulation results demonstrate the potential of the MM simulation technique. Throughout this chapter, we shall assume that the random variable M has the following probability mass function

$$p_m = e^{-c\,m} \quad \text{for } m = 1,2,...,J \tag{5.26}$$

where c is a positive constant that should be selected so that (5.26) is a valid probability mass function. The value of J was equal to 10 in all of the MM simulation results that are listed in this chapter (hence c = .6229.) The MM simulation model that we have used in the BSC case is characterized by the following crossover probability

$$\varepsilon_k^* = \begin{cases} .225 & \text{for } k \leq m \\ \varepsilon & \text{otherwise} \end{cases} \tag{5.27}$$

Tables 5.11 and 5.12 indicate that the above simulation model does indeed yield to accurate estimates of the distribution of computation along with high computational efficiency gains. Notice that only 250,000 simulation runs were used to obtain the MM estimates. The Monte Carlo simulation are based on 1,000,000 simulation runs and yet notice that for values of N (say greater than 30,) the Monte Carlo estimates of the $\mathcal{P}(C_j \geq n)$ are simply meaningless.

Table 5.11: The distribution of computation estimates for the constraint length 14 code operating on the BSC with $\varepsilon = .01$. $L = 250{,}000$ for the MM and $L = 1{,}000{,}000$ for MC.

	MM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 3)$	$.113 \times 10^{-1}$	0.30 %	$.115 \times 10^{-1}$	0.93 %
$\mathcal{P}(C_j \geq 18)$	$.163 \times 10^{-3}$	1.39 %	$.197 \times 10^{-3}$	7.12 %
$\mathcal{P}(C_j \geq 24)$	$.148 \times 10^{-3}$	1.35 %	$.167 \times 10^{-3}$	7.74 %
$\mathcal{P}(C_j \geq 33)$	$.373 \times 10^{-4}$	3.79 %	$.39 \times 10^{-4}$	16.02 %
$\mathcal{P}(C_j \geq 42)$	$.234 \times 10^{-4}$	4.58 %	$.24 \times 10^{-4}$	20.41 %
$\mathcal{P}(C_j \geq 51)$	$.202 \times 10^{-4}$	4.94 %	$.19 \times 10^{-4}$	22.93 %
$\mathcal{P}(C_j \geq 66)$	$.150 \times 10^{-4}$	5.73 %	$.12 \times 10^{-4}$	28.93 %
$\mathcal{P}(C_j \geq 75)$	$.108 \times 10^{-4}$	5.94 %	$.70 \times 10^{-5}$	37.74 %
$\mathcal{P}(C_j \geq 81)$	$.959 \times 10^{-5}$	6.47 %	$.60 \times 10^{-5}$	40.82 %
$\mathcal{P}(C_j \geq 87)$	$.774 \times 10^{-5}$	6.53 %	$.50 \times 10^{-5}$	44.64 %

Table 5.12 The distribution of computation estimates for the constraint length 5 code operating on the BSC with $\varepsilon = .005$. $L = 250{,}000$ for the MM and $L = 1{,}000{,}000$ for MC.

	MM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 3)$	$.547 \times 10^{-2}$	0.32 %	$.547 \times 10^{-2}$	1.35 %
$\mathcal{P}(C_j \geq 6)$	$.157 \times 10^{-3}$	1.30 %	$.149 \times 10^{-3}$	8.19 %
$\mathcal{P}(C_j \geq 12)$	$.142 \times 10^{-3}$	1.33 %	$.136 \times 10^{-3}$	8.58 %
$\mathcal{P}(C_j \geq 30)$	$.280 \times 10^{-4}$	1.20 %	$.230 \times 10^{-4}$	20.96 %
$\mathcal{P}(C_j \geq 45)$	$.197 \times 10^{-5}$	8.37 %	$.300 \times 10^{-5}$	57.80 %
$\mathcal{P}(C_j \geq 72)$	$.330 \times 10^{-6}$	4.19 %	$.100 \times 10^{-6}$	100.0 %
$\mathcal{P}(C_j \geq 78)$	$.315 \times 10^{-6}$	4.27 %	$.100 \times 10^{-6}$	100.0 %
$\mathcal{P}(C_j \geq 81)$	$.311 \times 10^{-6}$	4.30 %	$.100 \times 10^{-6}$	100.0 %
$\mathcal{P}(C_j \geq 84)$	$.304 \times 10^{-6}$	4.34 %	$.100 \times 10^{-6}$	100.0 %
$\mathcal{P}(C_j \geq 90)$	$.297 \times 10^{-6}$	4.37 %	$.100 \times 10^{-6}$	100.0 %

5.3.2 The AWGN Channel Case

We shall now present our importance sampling simulation results for the AWGN channel case. Recall that the correct path is assumed to be the all-zero path. That is, $u_k^c = (u_{k1}^c, u_{k2}^c) \equiv (-1,-1)$, and thus, if we define

$$g(v) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{v}{2\sigma^2}}. \tag{5.28}$$

Then the channel output symbol density is

$$f_k(v_k \mid u_k^c) = \prod_{i=1}^{2} f_{ki}(v_{ki} \mid u_{ki}^c) \tag{5.29}$$

where

$$f_{ki}(v_{ki} \mid u_{ki}^c) \equiv g(v_{ki} - 1) \quad \text{for i =1 or 2.} \tag{5.30}$$

We begin by presenting the simulation results for the reference path method.

A. The Reference Path Method

The biasing in the reference path method (RPM) for the AWGN channel and the BSC cases are essentially the same. In other words, for a given reference path of depth N, $\mathbf{u}_N^r = (u_1^r, u_2^r, ..., u_N^r)$, 1) we do not bias the channel when $u_{ki}^c = u_{ki}^r$; and 2) we use a uniform biasing for the remaining instances when $u_{ki}^c \neq u_{ki}^r$ (i=1 or 2.) Specifically, for the AWGN channel case, we use a channel output symbol simulation density which is normal with zero mean and variance σ^2 at the instances when $u_{ki}^c \neq u_{ki}^r$ (i=1,2.) Once the data is generated up to N, the depth of the reference path, no more biasing is done. Thus, the reference path method is based on a non-stationary memoryless model which is characterized by a non-stationary simulation density

$$f_{ki}^*(v_{ki} \mid u_{ki}^c) = \begin{cases} g(v_{ki}) & \text{if } u_{ki}^c \neq u_{ki}^r \\ f_{ki}(v_{ki} \mid u_{ki}^c) & \text{if } u_{ki}^c = u_{ki}^r \end{cases} \tag{5.31}$$

for i=1 or 2 and $k \leq N$. For $k > N$ we use $f_k^*(v_k \mid u_k^c) = f_k(v_k \mid u_k^c)$. Again, it should be clear that with roughly probability 1/2 the reference path will be examined up to depth N.

Some of the simulation results for the reference path method are summarized in Tables 5.13 and 5.14. The first two columns give the estimates of the distribution of computation and the accuracy using the reference path method (RPM). The last two columns give the estimates of the distribution of computation and the accuracy using ordinary Monte Carlo.

Table 5.13: The distribution of computation estimates for the constraint length 14 code operating on the AWGN channel with $\sigma = .6$. $L = 500{,}000$ for the RPM and $L = 1{,}000{,}000$ for MC.

	RPM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 3)$	$.129 \times 10^{-1}$	0.42 %	$.130 \times 10^{-1}$	0.87 %
$\mathcal{P}(C_j \geq 9)$	$.250 \times 10^{-2}$	0.68 %	$.252 \times 10^{-2}$	1.99 %
$\mathcal{P}(C_j \geq 15)$	$.101 \times 10^{-2}$	1.70 %	$.102 \times 10^{-2}$	3.13 %
$\mathcal{P}(C_j \geq 30)$	$.277 \times 10^{-3}$	3.58 %	$.274 \times 10^{-3}$	6.04 %
$\mathcal{P}(C_j \geq 48)$	$.118 \times 10^{-3}$	3.86 %	$.116 \times 10^{-3}$	9.29 %
$\mathcal{P}(C_j \geq 60)$	$.711 \times 10^{-4}$	4.76 %	$.710 \times 10^{-4}$	11.86 %
$\mathcal{P}(C_j \geq 72)$	$.446 \times 10^{-4}$	4.73 %	$.500 \times 10^{-4}$	14.14 %
$\mathcal{P}(C_j \geq 78)$	$.357 \times 10^{-4}$	4.32 %	$.400 \times 10^{-4}$	15.82 %
$\mathcal{P}(C_j \geq 81)$	$.334 \times 10^{-4}$	4.51 %	$.380 \times 10^{-4}$	16.23 %
$\mathcal{P}(C_j \geq 90)$	$.244 \times 10^{-4}$	5.02 %	$.350 \times 10^{-4}$	16.90 %

Table 5.14: The distribution of computation estimates for the constraint length 5 code operating on the AWGN channel with $\sigma = .55$. $L = 600{,}000$ for the RPM and $L = 1{,}000{,}000$ for MC.

	RPM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 3)$	$.648 \times 10^{-2}$	0.47 %	$.638 \times 10^{-2}$	1.25 %
$\mathcal{P}(C_j \geq 9)$	$.114 \times 10^{-2}$	1.04 %	$.114 \times 10^{-2}$	2.96 %
$\mathcal{P}(C_j \geq 15)$	$.415 \times 10^{-3}$	1.24 %	$.396 \times 10^{-3}$	5.02 %
$\mathcal{P}(C_j \geq 18)$	$.240 \times 10^{-3}$	1.36 %	$.233 \times 10^{-3}$	6.55 %
$\mathcal{P}(C_j \geq 21)$	$.167 \times 10^{-3}$	1.59 %	$.161 \times 10^{-3}$	7.88 %
$\mathcal{P}(C_j \geq 24)$	$.917 \times 10^{-4}$	2.07 %	$.850 \times 10^{-4}$	10.84 %
$\mathcal{P}(C_j \geq 30)$	$.424 \times 10^{-4}$	3.22 %	$.33 \times 10^{-4}$	17.39 %
$\mathcal{P}(C_j \geq 39)$	$.192 \times 10^{-4}$	5.01 %	$.140 \times 10^{-4}$	26.74 %
$\mathcal{P}(C_j \geq 45)$	$.117 \times 10^{-4}$	7.01 %	$.900 \times 10^{-5}$	33.33 %
$\mathcal{P}(C_j \geq 87)$	$.828 \times 10^{-6}$	7.28 %	**	**

Notice that Table 5.14 indicates that the event { $C_j \geq 87$ } has not been observed during the one million Monte Carlo simulation runs. We note that for 10 % accuracy, ordinary MC would require more than 120 Million simulation runs to estimate $\mathcal{P}(C_j \geq 87)$.

B. The Partitioning Method

Let N^* be the depth at which the correct path minimum (after depth j) occurs. Recall that in Section 4.5.4, we have shown that Given $N^* = m$, the partitioning method simulation model is

$$f_k^o(v_k \mid u_k^c) \propto \begin{cases} \exp\left[.5\,\Psi(u_k^c, u_k^i, v_k)\right] f_k(v_k \mid u_k^c) & \text{if } k \leq m \\ f_k(v_k \mid u_k^c) & \text{otherwise} \end{cases} \qquad (5.32)$$

where $\Psi(.,.,.)$ is an optimized twisting function which is given by (4.67). Furthermore, recall that we have also shown that for $k \leq m$, the above simulation model can be expressed as

$$f_k^o(v_k \mid u_k^c) = \begin{cases} q^- f^-(v_k) & \text{if } v_k \leq 0 \\ q^+ f^+(v_k) & \text{if } v_k > 0 \end{cases} \qquad (5.33)$$

where $f^-(.)$, $f^+(.)$, q^- and q^+ are defined in (4.72), (4.73), (4.75) and (4.76), respectively. A close observation of equations (4.72), (4.73), (4.75) and (4.76) indicates that the densities $f^-(.)$, $f^+(.)$ and the constants q^- and q^+ depend on only α^c. As in the BSC case, it turns out that for small values of m, the optimal value of α^c which is predicted by the partitioning method analysis does not lead to the most efficient simulation model. Hence, in order to optimize the computational efficiencies, it was necessary to find the optimal values of α^c by a trial and error.

Tables 5.15 and 5.16 show some partitioning method simulation results for the constraint lengths 5 and 14 convolutional codes. For both of these codes a total of 600,000 simulation runs were used. The values of α^c which we have used for the constraint length 5 convolutional code are shown below (the maximum values of m used in this case was 15.)

$$\alpha^c = \begin{cases} 0 & \text{if } k = 0 \\ -4.25 & \text{if } k = 1 \\ -4.65 & \text{if } k = 2 \\ -4.55 & \text{if } 3 \le k \le 15 \end{cases} \tag{5.34}$$

Likewise, the values of α^c which we have used for the constraint length 14 convolutional code are shown below (the maximum values of m used in this case was 12.)

$$\alpha^c = \begin{cases} 0 & \text{if } k = 0 \\ -4.25 & \text{if } k = 1 \\ -4.65 & \text{if } k = 2 \\ -4.00 & \text{if } k = 3 \\ -3.5 & \text{if } 4 \le k \le 5 \\ -3.25 & \text{if } 6 \le k \le 7 \\ -3.00 & \text{if } 8 \le k \le 12 \end{cases} \tag{5.35}$$

Table 5.15: The distribution of computation estimates for the constraint length 5 code operating on the AWGN channel with $\sigma = .6$. $L = 600{,}000$ for the PM and $L = 1{,}000{,}000$ for MC.

	PM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 3)$	$.127 \times 10^{-1}$	2.10 %	$.131 \times 10^{-1}$	0.87 %
$\mathcal{P}(C_j \geq 9)$	$.300 \times 10^{-2}$	3.39 %	$.303 \times 10^{-2}$	1.81 %
$\mathcal{P}(C_j \geq 15)$	$.144 \times 10^{-2}$	4.39 %	$.146 \times 10^{-2}$	2.61 %
$\mathcal{P}(C_j \geq 27)$	$.334 \times 10^{-3}$	6.42 %	$.338 \times 10^{-3}$	5.44 %
$\mathcal{P}(C_j \geq 33)$	$.206 \times 10^{-3}$	7.73 %	$.207 \times 10^{-3}$	6.95 %
$\mathcal{P}(C_j \geq 54)$	$.399 \times 10^{-4}$	7.47 %	$.590 \times 10^{-4}$	13.02 %
$\mathcal{P}(C_j \geq 60)$	$.290 \times 10^{-4}$	7.06 %	$.390 \times 10^{-4}$	16.03 %
$\mathcal{P}(C_j \geq 66)$	$.212 \times 10^{-4}$	7.87 %	$.340 \times 10^{-4}$	17.15 %
$\mathcal{P}(C_j \geq 87)$	$.899 \times 10^{-5}$	8.08 %	$.170 \times 10^{-4}$	24.25 %
$\mathcal{P}(C_j \geq 90)$	$.826 \times 10^{-5}$	8.64 %	$.140 \times 10^{-4}$	26.72 %

Table 5.16: The distribution of computation estimates for the constraint length 14 code operating on the AWGN channel with $\sigma = .6$. $L = 600{,}000$ for the PM and $L = 1{,}000{,}000$ for MC.

	PM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 3)$	$.124 \times 10^{-1}$	1.97 %	$.130 \times 10^{-1}$	0.87 %
$\mathcal{P}(C_j \geq 9)$	$.234 \times 10^{-2}$	2.99 %	$.252 \times 10^{-2}$	1.99 %
$\mathcal{P}(C_j \geq 24)$	$.369 \times 10^{-3}$	5.88 %	$.398 \times 10^{-3}$	5.01 %
$\mathcal{P}(C_j \geq 30)$	$.252 \times 10^{-3}$	6.61 %	$.274 \times 10^{-3}$	6.04 %
$\mathcal{P}(C_j \geq 48)$	$.105 \times 10^{-3}$	6.87 %	$.116 \times 10^{-3}$	9.29 %
$\mathcal{P}(C_j \geq 57)$	$.704 \times 10^{-4}$	6.92 %	$.860 \times 10^{-4}$	10.78 %
$\mathcal{P}(C_j \geq 63)$	$.584 \times 10^{-4}$	7.69 %	$.670 \times 10^{-4}$	12.22 %
$\mathcal{P}(C_j \geq 78)$	$.281 \times 10^{-4}$	7.40 %	$.400 \times 10^{-4}$	15.82 %
$\mathcal{P}(C_j \geq 81)$	$.271 \times 10^{-4}$	7.60 %	$.380 \times 10^{-4}$	16.23 %
$\mathcal{P}(C_j \geq 90)$	$.207 \times 10^{-4}$	8.14 %	$.350 \times 10^{-4}$	16.90 %

C. The M-Method

We shall now present and demonstrate the power and accuracy of the M-Method (MM) by showing some simulation results for the rate 1/2 and constraint length 5 and 14 convolutional codes operating on the AWGN channel.

As in the BSC case, the MM estimator is given by (5.24) and the probability mass function of the random variable M is given by (5.26). The MM simulation model that we have used in the AWGN channel case is given by

$$f_k^*(v_k \mid u_k^c, m) = \begin{cases} g(v_k) & \text{if } k = m \\ f_k(v_k \mid u_k^c) & \text{otherwise} \end{cases} \tag{5.36}$$

In other words, for a given m, the simulation density is normal with zero mean and variance σ^2 for all $k \leq m$. for $k > m$, the MM simulation model switches to the original channel model.

Some of the MM simulation results which we have obtained using the simulation model (5.36) are shown in Tables 5.17-5.19.

Table 5.17: The distribution of computation estimates for the constraint length 14 code operating on the AWGN channel with $\sigma = .6$. $L = 300,000$ for the MM and $L = 1,000,000$ for MC.

	MM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 3)$	$.118 \times 10^{-1}$	1.43 %	$.130 \times 10^{-1}$	0.87 %
$\mathcal{P}(C_j \geq 9)$	$.243 \times 10^{-2}$	2.45 %	$.252 \times 10^{-2}$	1.99 %
$\mathcal{P}(C_j \geq 18)$	$.627 \times 10^{-3}$	2.70 %	$.637 \times 10^{-3}$	3.96 %
$\mathcal{P}(C_j \geq 30)$	$.258 \times 10^{-3}$	3.45 %	$.274 \times 10^{-3}$	6.04 %
$\mathcal{P}(C_j \geq 48)$	$.109 \times 10^{-3}$	4.38 %	$.116 \times 10^{-3}$	9.29 %
$\mathcal{P}(C_j \geq 60)$	$.658 \times 10^{-4}$	4.71 %	$.710 \times 10^{-4}$	11.86 %
$\mathcal{P}(C_j \geq 66)$	$.509 \times 10^{-4}$	5.86 %	$.650 \times 10^{-4}$	12.40 %
$\mathcal{P}(C_j \geq 72)$	$.395 \times 10^{-4}$	5.79 %	$.500 \times 10^{-4}$	14.14 %
$\mathcal{P}(C_j \geq 87)$	$.233 \times 10^{-4}$	5.35 %	$.350 \times 10^{-4}$	16.90 %
$\mathcal{P}(C_j \geq 90)$	$.217 \times 10^{-4}$	5.86 %	$.350 \times 10^{-4}$	16.90 %

Table 5.18: The distribution of computation estimates for the constraint length 5 code operating on the AWGN channel with $\sigma = .55$. $L = 300{,}000$ for the MM and $L = 1{,}000{,}000$ for MC.

	MM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 9)$	$.110 \times 10^{-2}$	1.66 %	$.114 \times 10^{-2}$	2.96 %
$\mathcal{P}(C_j \geq 15)$	$.391 \times 10^{-3}$	2.19 %	$.396 \times 10^{-3}$	5.02 %
$\mathcal{P}(C_j \geq 18)$	$.237 \times 10^{-3}$	3.12 %	$.233 \times 10^{-3}$	6.55 %
$\mathcal{P}(C_j \geq 30)$	$.387 \times 10^{-4}$	3.52 %	$.330 \times 10^{-4}$	17.41 %
$\mathcal{P}(C_j \geq 42)$	$.103 \times 10^{-4}$	3.03 %	$.100 \times 10^{-4}$	31.62 %
$\mathcal{P}(C_j \geq 51)$	$.568 \times 10^{-5}$	3.99 %	$.700 \times 10^{-5}$	37.79 %
$\mathcal{P}(C_j \geq 72)$	$.127 \times 10^{-5}$	5.29 %	$.200 \times 10^{-5}$	70.72 %
$\mathcal{P}(C_j \geq 81)$	$.946 \times 10^{-6}$	5.60 %	$.100 \times 10^{-5}$	100 %
$\mathcal{P}(C_j \geq 84)$	$.765 \times 10^{-6}$	6.19 %	$.100 \times 10^{-5}$	100 %
$\mathcal{P}(C_j \geq 90)$	$.591 \times 10^{-6}$	7.19 %	**	**

Table 5.19: The distribution of computation estimates for the constraint length 14 code operating on the AWGN channel with $\sigma = .55$. $L = 100{,}000$ for the MM and $L = 1{,}000{,}000$ for MC.

	MM	Accuracy	MC	Accuracy
$\mathcal{P}(C_j \geq 12)$	$.346 \times 10^{-3}$	8.50 %	$.333 \times 10^{-3}$	5.48 %
$\mathcal{P}(C_j \geq 15)$	$.213 \times 10^{-3}$	3.75 %	$.233 \times 10^{-3}$	6.55 %
$\mathcal{P}(C_j \geq 18)$	$.109 \times 10^{-3}$	4.60 %	$.128 \times 10^{-3}$	8.84 %
$\mathcal{P}(C_j \geq 24)$	$.568 \times 10^{-4}$	7.18 %	$.660 \times 10^{-4}$	12.31 %
$\mathcal{P}(C_j \geq 36)$	$.262 \times 10^{-4}$	6.93 %	$.330 \times 10^{-4}$	17.41 %
$\mathcal{P}(C_j \geq 60)$	$.372 \times 10^{-5}$	5.26 %	$.700 \times 10^{-5}$	37.79 %
$\mathcal{P}(C_j \geq 66)$	$.284 \times 10^{-5}$	5.73 %	$.500 \times 10^{-5}$	44.72 %
$\mathcal{P}(C_j \geq 72)$	$.219 \times 10^{-5}$	6.77 %	$.100 \times 10^{-5}$	100 %
$\mathcal{P}(C_j \geq 81)$	$.170 \times 10^{-5}$	7.92 %	**	**
$\mathcal{P}(C_j \geq 90)$	$.116 \times 10^{-5}$	8.57 %	**	**

5.4 Discussion and Conclusion

In this section, we shall further discuss and compare the performance of the new importance sampling schemes that we have presented here. We begin by presenting some simulation results for the rate 1/2 and constraint lengths 5 and 14 convolutional codes operating on the BSC and the AWGN channel.

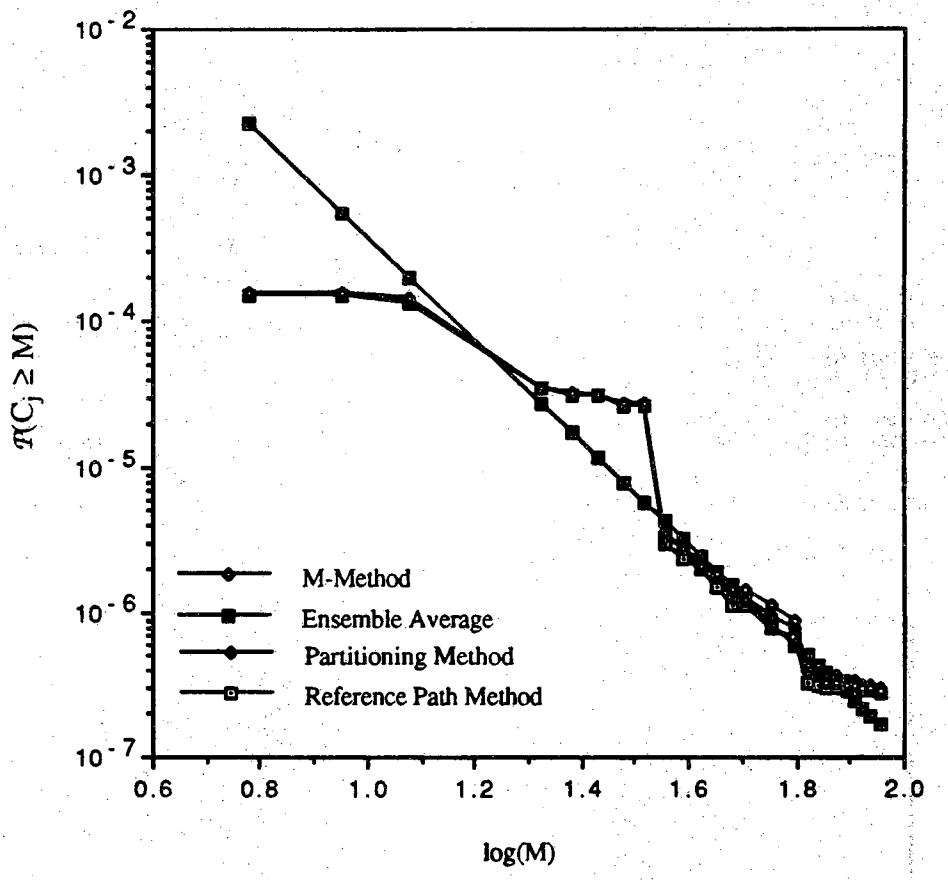


Figure 5.8: The distribution of computation estimates for the constraint length 5 code operating on the BSC with $\varepsilon = .005$.

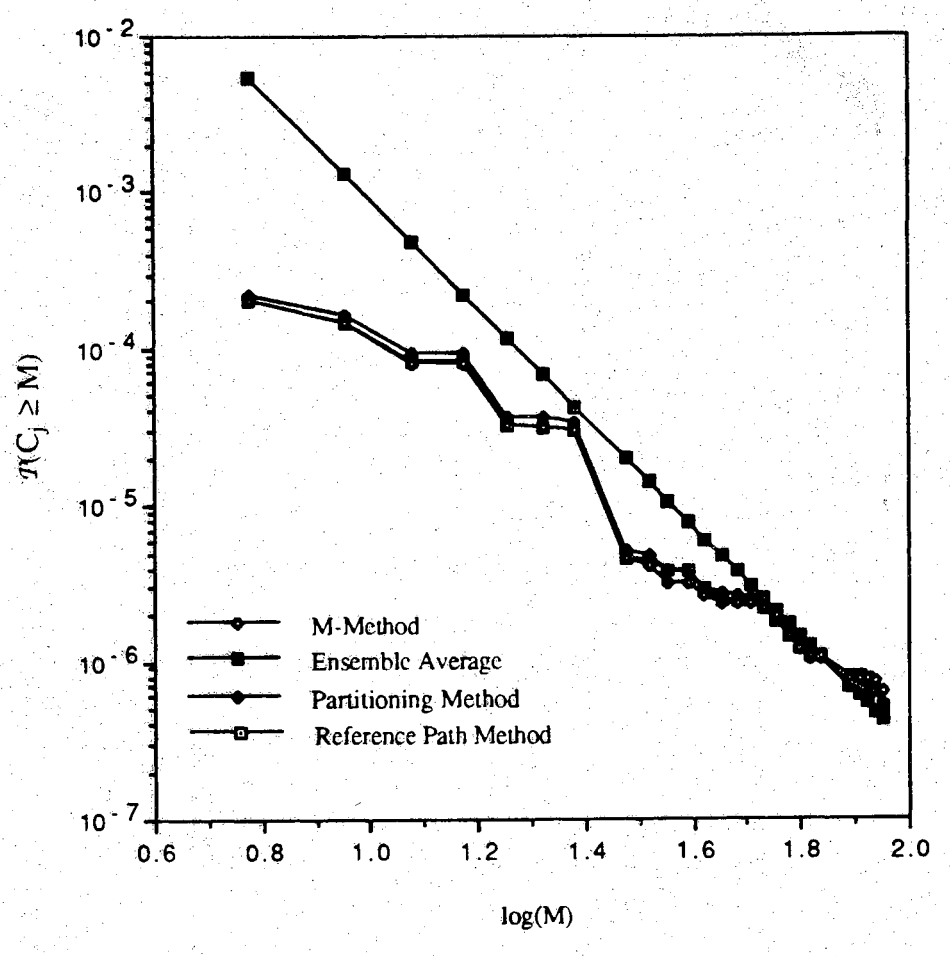


Figure 5.9: The distribution of computation estimates for the constraint length 14 code operating on the BSC with $\varepsilon = .005$.

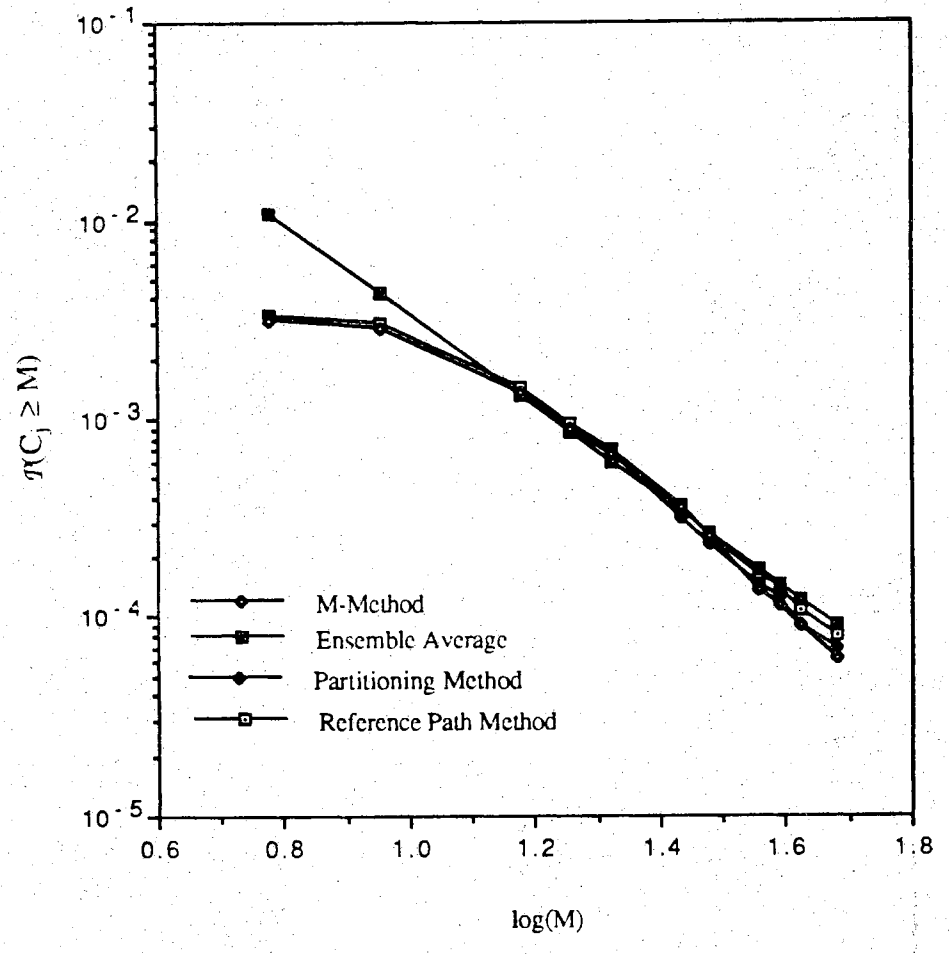


Figure 5.10: The distribution of computation estimates for the constraint length 5 code operating on the AWGN channel with $\sigma = .6$.

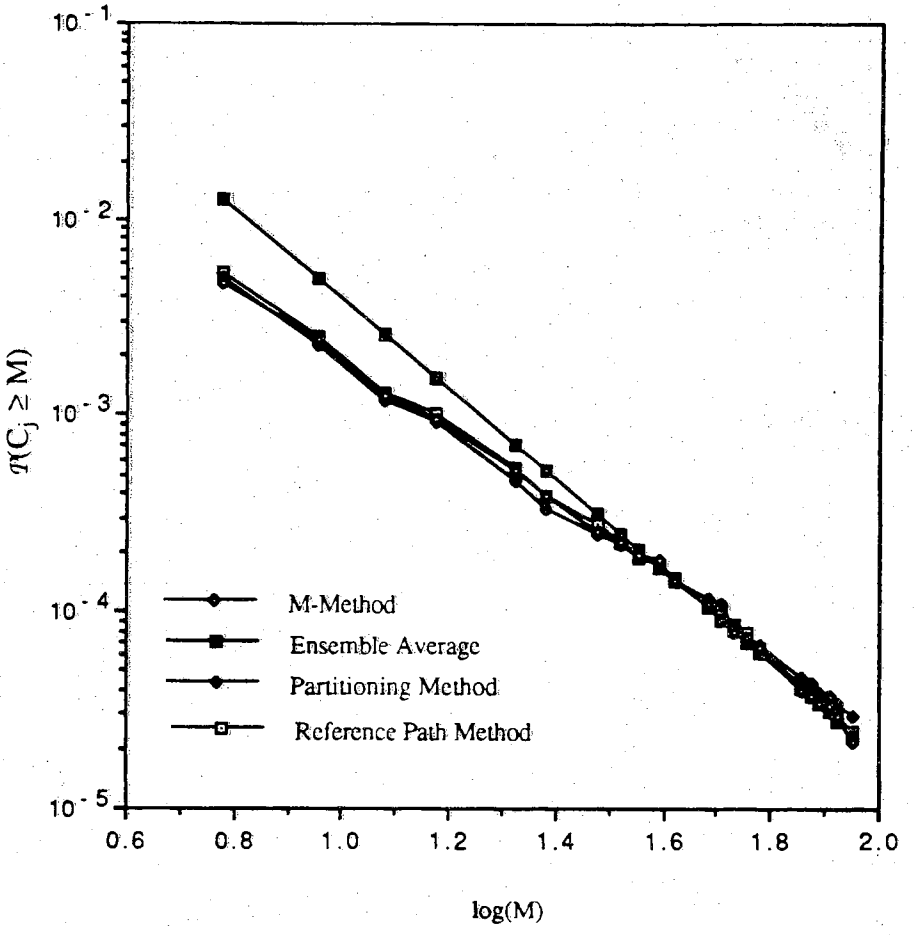


Figure 5.11: The distribution of computation estimates for the constraint length 14 code operating on the AWGN channel with $\sigma = .6$.

The data in figure 5.8-5.11 was obtained empirically using a total of 300,000 simulation runs for all the simulation schemes in the BSC case. In the AWGN case, a total of 600,000 simulation runs were used for the constraint length 5 convolutional code. For the constraint length 14 convolutional code, however, a total of 500,000 were used for the reference path method. For the partitioning method and the M-method simulation schemes a total of 300,000 simulation runs were used for both schemes. This is far more simulation runs than is required for estimating $\mathcal{P}(C_j \geq M)$ for $M = 1,..90$; however, it was our desire to ensure accurate answers.

Recall that the information theoretic ensemble average analysis indicates that the distribution $\mathcal{P}(C_j \geq M)$ should have a Pareto tail with exponent ρ. That is,

$$\mathcal{P}(C_j \geq M) \sim M^{-\rho} \tag{5.37}$$

or equivalently,

$$\log(\mathcal{P}(C_j \geq M)) \sim -\rho \log(M). \tag{5.38}$$

It turns out that for the BSC of Figures 5.8 and 5.9, the ensemble average exponent is $\rho = 3.5$ and for the AWGN channel of Figures 5.10 and 5.11, the exponent is $\rho = 2.3$. For comparison, Figures 5.8-5.11 also plot curves labeled "Ensemble Average." These curves are just $c\, M^{-\rho}$ where the constant c is fit for the tail of our importance sampling estimates. We note that the curves appear as straight lines because these figures are logarithmic scale plots of the probability estimates against log(M).

Table 5.20 compares the power and accuracy of the new importance sampling schemes that we have presented here. This table give some of the estimates of the distribution of computation for the constraint length 5 convolutional code operating on the BSC with $\varepsilon = .005$. Table 5.20 also lists estimates of the relative accuracy.

Table 5.20: Comparison of the RPM, the PM and the MM for the constraint length 5 code operating on the BSC with $\varepsilon = .005$. $L = 300{,}000$ for all schemes, and A= relative accuracy estimates.

	RPM	A	PM	A	MM	A
$\mathcal{P}(C_j \geq 6)$	$.15 \times 10^{-3}$	1.92 %	$.16 \times 10^{-3}$	0.86 %	$.16 \times 10^{-3}$	1.19 %
$\mathcal{P}(C_j \geq 12)$	$.14 \times 10^{-3}$	1.98 %	$.14 \times 10^{-3}$	0.93 %	$.14 \times 10^{-3}$	1.22 %
$\mathcal{P}(C_j \geq 21)$	$.34 \times 10^{-4}$	2.30 %	$.36 \times 10^{-4}$	1.03 %	$.35 \times 10^{-4}$	1.58 %
$\mathcal{P}(C_j \geq 30)$	$.27 \times 10^{-4}$	1.26 %	$.28 \times 10^{-4}$	1.06 %	$.28 \times 10^{-4}$	1.05 %
$\mathcal{P}(C_j \geq 33)$	$.27 \times 10^{-4}$	1.26 %	$.27 \times 10^{-4}$	1.06 %	$.28 \times 10^{-4}$	1.05 %
$\mathcal{P}(C_j \geq 72)$	$.30 \times 10^{-6}$	3.33 %	$.37 \times 10^{-6}$	2.01 %	$.33 \times 10^{-6}$	3.84 %
$\mathcal{P}(C_j \geq 84)$	$.284 \times 10^{-6}$	3.42 %	$.32 \times 10^{-6}$	1.98 %	$.30 \times 10^{-6}$	3.97 %
$\mathcal{P}(C_j \geq 90)$	$.28 \times 10^{-6}$	3.45 %	$.30 \times 10^{-6}$	1.98 %	$.29 \times 10^{-6}$	3.99 %

better than the other two schemes. In other words, it was found that the reference path method usually take less CPU time than the partitioning method and the M-method. This is reasonable because the partitioning method and the M-method will tend to "grow" big incorrect subtrees. As an example, we note that for the BSC example of Table 5.20, the reference path method took 11 minutes, the M-method took about 12 minutes, while the partitioning method took about 21 minutes. On the other hand, for the AWGN channel data of Figure 5.10, we note that the reference path method took 84 minutes, the partitioning method took 135 minutes, while the M-method took 486 minutes.

We conclude this section by comparing the modified stack algorithm simulation (MSAS) with the stack algorithm simulation. Recall that because we are only interested in estimating $\mathcal{P}(C_j \geq M)$ for a given M, it follows that only the search performed by the stack algorithm in the j'th incorrect subtree $\mathcal{N}_j$ is of interest to us. Keeping this in mind, we have designed the MSAS (see Chapter 3) which operates exactly like the stack algorithm simulation except that it 1) extends only the j'th incorrect subtree and 2) replaces every top-of-stack node which is on the correct path by only its direct descendent which is on the correct path. Because the search performed by the stack algorithm in $\mathcal{N}_j$ can be affected by the search performed by the stack algorithm in other incorrect subtrees, it is possible that the estimates obtained using the MSAS might be incorrect. It turns out, however, that this last statement is apparently not true! In other words, the MSAS gives estimates which are very close to the ones obtained when the stack algorithm is actually used. Tables 5.21 and 5.22 illustrate this by listing some simulation results which were obtained using the MSAS and the stack algorithm.

Table 5.21: Comparison of the modified stack algorithm simulation (MSAS) and the stack algorithm (SA) results for the constraint length 5 convolutional code operating on the BSC with $\varepsilon = .005$ using the reference path method. L = 300,000.

	MSAS	Accuracy	SA	Accuracy
$\mathcal{P}(C_j \geq 3)$	$.540 \times 10^{-2}$	0.52 %	$.541 \times 10^{-2}$	0.57 %
$\mathcal{P}(C_j \geq 30)$	$.267 \times 10^{-4}$	1.25 %	$.272 \times 10^{-4}$	1.41 %
$\mathcal{P}(C_j \geq 45)$	$.148 \times 10^{-5}$	6.48 %	$.148 \times 10^{-5}$	6.51 %
$\mathcal{P}(C_j \geq 75)$	$.297 \times 10^{-6}$	3.34 %	$.295 \times 10^{-6}$	3.36 %

Table 5.22: Comparison of the modified stack algorithm simulation (MSAS) and the stack algorithm (SA) results for the constraint length 14 convolutional code operating on the AWGN channel with $\sigma = .6$. L = 500,000 and the reference path method was used.

	MSAS	Accuracy	SA	Accuracy
$\mathcal{P}(C_j \geq 6)$	$.526 \times 10^{-2}$	0.68 %	$.523 \times 10^{-2}$	0.68 %
$\mathcal{P}(C_j \geq 27)$	$.346 \times 10^{-3}$	3.02 %	$.345 \times 10^{-3}$	3.07 %
$\mathcal{P}(C_j \geq 48)$	$.115 \times 10^{-3}$	4.67 %	$.112 \times 10^{-3}$	3.93 %
$\mathcal{P}(C_j \geq 78)$	$.369 \times 10^{-4}$	7.60 %	$.366 \times 10^{-4}$	6.95 %

By comparing the data in Tables 5.21 and 5.22, it does appear that both the MSAS and the stack algorithm simulation give very close estimates and accuracy. In terms of CPU time we have found out that our simulation methods take about twice less CPU time whenever the MSA is used instead of the stack algorithm.

CHAPTER 6
SEQUENTIAL DECODERS
ERROR EVENTS SIMULATION

6.1 Introduction

In the previous chapters, we have discussed and presented new importance sampling techniques for simulating (stack algorithm) sequential decoders decision processes. Recall that in these schemes, we did not consider decision error probabilities. In other words, we have assumed that the decoder ultimately chooses the correct transmitted path. In this chapter, we shall consider the simulation of the error events associated with the stack algorithm sequential decoders. In particular, we will use the *error event simulation method* to estimate bit error rates for such decoders. The error event simulation method is an importance sampling technique which has been developed by Sadowsky [24] to simulate the Viterbi decoder. In this section we shall apply this technique to the simulation of the stack algorithm sequential decoders. The basic ideas here are the same, however, there is one significant difference between the stack algorithm and the Viterbi algorithm operation which complicates the simulation issue: In contrast to the Viterbi algorithm, the *time of a correct decode* for the stack algorithm is not a *stopping time*. In this application, we avoid this problem by using a termination strategy which we shall refer to as the Δ–stopping rule.

In this chapter, we will only consider the problem of estimating bit error rates associated with (stack algorithm) sequential decoders. However, we shall note that the error event simulation technique can be used to estimate many key performance parameters such as error burst length, error burst length distribution, and so on [24].

We shall begin this chapter by briefly reviewing the concept of error events or bursts of decoding errors.

6.2 Error Events and Performance Parameters

Let $\mathbf{u}^f = (u_1^f, u_2^f, ..)$ denotes the final path hypothesized by the stack algorithm decoder. Likewise let $\mathbf{u}^c = (u_1^c, u_2^c, \ldots, u_k^c, ...)$ denotes the correct (that is, the transmitted) path. In the error event simulation technique, the index k is associated with the operation of the encoder. That is, k is the time index on the trellis diagram, not for the input bit stream. Thus for a rate b/n convolutional code, the passage from time k to k+1 corresponds to b information bits input into the encoder and n bits being shifted out. Now recall that decoding errors occur when the decoder output sequence diverges from the correct path. That is, when $\mathbf{u}^f$ diverges from $\mathbf{u}^c$. We call such a divergent path an *error event,* or a *burst of decoding errors,* or simply, a *burst.* Specifically, a burst is a partial sequence of all incorrect decodes which is immediately preceded by and immediately followed by correct decodes. The *burst length* is simply the number of incorrect decodes in a burst. Note that the minimum burst length is K-1 (recall that Kb is the code constraint length for a rate b/n convolutional code.) This is true because once an incorrect state is entered into the encoder, it takes K-1 consecutive correct information symbols to flush out the encoder shift register. For example, Figure 6.1 shows a burst of length 5. Notice that the hamming distance between the code sequences of the error event and the correct path in Figure 6.1 is 7.

Furthermore, note that this error event differs in only one input bit from the correct path. Hence, the number of bit errors that will be caused by decoding the error burst shown in Figure 6.1 instead of the correct path will be 1 (recall that a dotted line denotes the output generated by the information bit 1 and a solid line denotes the output generated by the information bit 0.)

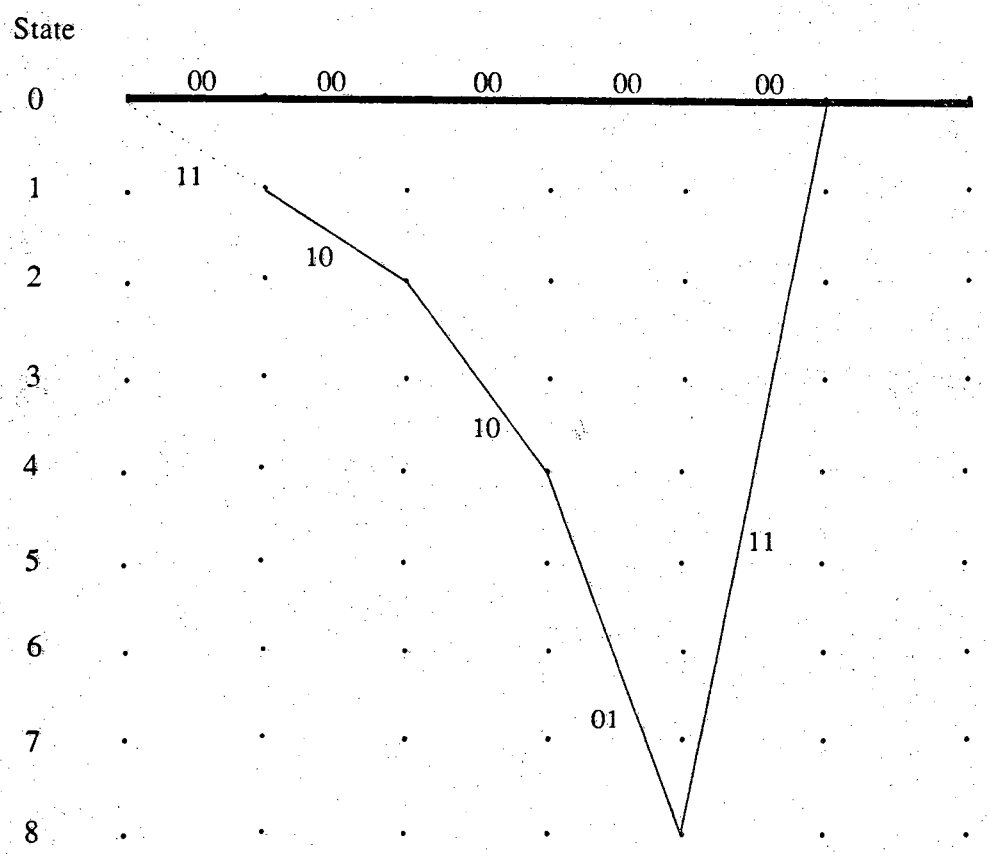


Figure 6.1: An illustration of a burst of length 5. The bold line is the correct path.

Now, suppose that the decoder has made a correct decode at a given time j. That is, the decoded branch u_{j+1}^{f} emanates from the correct path node at time j on the trellis diagram. On this event, define the random variable

$$N_b(j) \triangleq \begin{array}{l}\text{the total number of bit errors due the error event} \\ \text{immediately following the correct decode at time j}\end{array} \tag{6.1}$$

In the above definition, we must allow the possibility of a correct decode at time j+1. In this case, the burst is said to be *trivial,* and causes no bit erros. That is, $N_b(j) = 0$.

Next consider the expectation

$$\overline{N}_b \triangleq E[\, N_b(j) \mid \mathbf{u}^c \text{ transmitted and correct decode at time j }]. \tag{6.2}$$

In the case of binary input-output channels, it turns out that $\overline{N}_b$ does not depend on j and $\mathbf{u}^c$ [57, Chapter 4]. Thus, hereafter we can arbitrarily set $\mathbf{u}^c$ to be the *all-zero path,* and time $j = 0$. In addition, notice that since $\overline{N}_b$ does not depend on j and $\mathbf{u}^c$, it follows that

$$\overline{N}_b \triangleq \text{the expected number of bit errors per correct decode.} \tag{6.3}$$

Since each transition from time k to time k+1 represents the encoding of b bits, it follows that

$$P_b \triangleq \frac{\overline{N}_b}{b} \tag{6.4}$$

is the *average number of bit errors per correct decode.* P_b is the key parameter which we shall estimate via importance sampling using the error event simulation technique.

We shall now define

$$\mathcal{E}_0 \triangleq \begin{array}{l}\text{the collection of all error events emanating} \\ \text{from a correct node at time } j = 0\end{array} \tag{6.5}$$

That is, $\mathcal{E}_0$ is the collection of all bursts immediately following the correct decode at time $j=0$ (including the trivial burst.) An error event in $\mathcal{E}_0$ will be represented by its divergent branches $\mathbf{u}'=(u'^{f}_1,\ldots,u'^{f}_m)$ where m is the burst length. Let $n_b(\mathbf{u}^c,\mathbf{u}')$ denote the number of bit errors caused by decoding $\mathbf{u}'$ instead of $\mathbf{u}^c$ when $\mathbf{u}^c$ was transmitted. $P(\mathbf{u}'|\mathbf{u}^c)$ shall denote the conditional probability of decoding $\mathbf{u}'$ instead of $\mathbf{u}^c$ given that $\mathbf{u}^c$ is transmitted and given that the decoder has made a correct decode at time $j=0$. Then

$$\bar{N}_b = \sum_{\mathbf{u}'\in\mathcal{E}_0} n_b(\mathbf{u}^c,\mathbf{u}')\,P(\mathbf{u}'|\mathbf{u}^c). \tag{6.6}$$

6.3 The Error Event Simulation Method

The error event simulation technique is an importance sampling technique which is based on the sum (6.6). Our desire is to estimate the average number of bit errors per correct decode, P_b, via importance sampling. From (6.4) we can see that this is equivalent to the problem of estimating $\bar{N}_b$. It turns out that only few terms dominate the sum (6.6). Thus the error event simulation method must emphasize those error events because these are precisely the "important" error events in $\mathcal{E}_0$.

The basic principle of the *error event simulation technique* is that each simulation run will produce precisely one simulated error event. The data sequence $\mathbf{V}^{(\ell)}=(V_1^{(\ell)},V_2^{(\ell)},\ldots)$ for the ℓ'th simulation run is generated by sequentially encoding the data sequence to produce a sequence of branching decisions $\mathbf{u}'=(u'_1,u'_2,\ldots)$. Each simulation is conditioned on a correct decision at time $j=0$. As soon as the encoded path $\mathbf{u}'$ merges with the correct path on the trellis diagram at some time $J>0$, the simulation is terminated. Consequently, the length of an error event simulation is a random variable. Define

$$T^{(0)} \triangleq \text{time of the first correct decode.} \tag{6.7}$$

That is, $T^{(0)}$ is the first time that the decoded path *remerges* with correct path on the trellis diagram (after the correct decode at time $j = 0$.) Then each simulation run needs to generate the simulation data $\mathbf{V}^{(0)}$ only up to time $T^{(0)}$. As a result $T^{(0)}$ is the length of the simulation data $\mathbf{V}^{(0)}$.

Now recall that in the previous chapter, we have presented the DELTA stopping rule for the stack algorithm simulation. Recall that the basic idea behind this stopping rule is to discard the inactive subtrees as the search progresses and concentrate on only the active ones which contain the TOS node. In this chapter, we slightly modify this rule in order to apply it to the error event simulation scheme. We shall refer to the termination strategy in the error event simulation technique as the Δ–stopping rule. This rule is given below.

The Δ-Stopping Rule (For one simulation):

Initialize: Start the search at node j.

DO i= 1, ...

Simulate until only one subtree is active, call it S_{β_i}.

IF β_i is a correct node **THEN**

STOP

ELSE

Delete inactive subtrees from the stack.

Reassociate the remaining stack nodes to subtrees S_δ with $\delta \in D_{\beta_i}$.

Compute $\overline{M}_\delta(t)$ for $\delta \in D_{\beta_i}$.

END IF

CONTINUE

Observe that by following the above termination strategy, the terminal path that the stack algorithm finally chooses is simply $(j,\beta_1,\beta_2,...)$. Thus for the ℓ'th simulation, $T^{(\ell)}$ is simply the time when the simulation stops. In other words, it is the first time that one of the node β_i for $i \geq 1$ is found to be a correct node. As a consequence, the simulation termination time $T^{(\ell)}$ is a stopping time. That is, given the infinite sequence of channel outputs $\mathbf{V}^{(\ell)}= (V_1^{(\ell)}, V_2^{(\ell)},...)$, the event $\{T^{(\ell)} = t\}$ is determined to be true or false by only the data up to time t, that is, $(V_1^{(\ell)}, \ldots, V_t^{(\ell)})$.

6.4 Importance Sampling and Error Event Simulation

Let $I_{\mathbf{u}'}(\mathbf{v})$ be the indicator function for decoding the error event $\mathbf{u}'$. That is, $I_{\mathbf{u}'}(\mathbf{v}) = 1$ if the channel output sequence $\mathbf{V}$ causes $\mathbf{u}'$ to be detected, otherwise, $I_{\mathbf{u}'}(\mathbf{v}) = 0$. Then the *importance sampling estimator* for $P(\mathbf{u}' \mid \mathbf{u}^c)$ is

$$\hat{P}(\mathbf{u}' \mid \mathbf{u}^c) = \frac{1}{L} \sum_{\ell=1}^{L} w(\mathbf{V}^{(\ell)} \mid \mathbf{u}^c)\, I_{\mathbf{u}'}(\mathbf{V}^{(\ell)}) \tag{6.8}$$

where

$$w(\mathbf{v}^{(\ell)} \mid \mathbf{u}^c) = \prod_{k=1}^{T^{(\ell)}} \frac{f_k(v_k \mid u_k^c)}{f_k^*(v_k \mid u_k^c)}. \tag{6.9}$$

The likelihood (6.9) is the *importance sampling weight.* Notice that (6.9) implies that the importance sampling model is *memoryless,* but possibly *non-stationary* with transition probability $f_k^*(v_k \mid u_k^c)$ at time k. Furthermore, note that we must have

$$f_k^*(v_k \mid u_k^c) > 0 \quad \text{whenever} \quad f_k(v_k \mid u_k^c) > 0 \tag{6.10}$$

so that the importance sampling weight (6.9) is well defined.

A similar computations to the ones used in [24] indicates that the importance sampling estimator $\hat{P}(\mathbf{u}' \mid \mathbf{u}^c)$ as specified by (6.8) is unbiased. Indeed, if we let $I_{t,\mathbf{u}'}(\mathbf{V}^{(\ell)})$ be the indicator of the event

$$\{\, T^{(\ell)} = t \text{ and path } \mathbf{u}' \text{ decoded } \}.$$

Then because $T^{(\ell)}$ is a stopping time, it follows that $I_{t,\mathbf{u}'}(\mathbf{V}^{(\ell)})$ depends on only $(V_1^{(\ell)}, \ldots, V_t^{(\ell)})$. As a result, we can write $I_{\mathbf{u}'}(\mathbf{V}^{(\ell)})$ as

$$I_{\mathbf{u}'}(\mathbf{V}^{(\ell)}) = \sum_{t=1}^{\infty} I_{t,\mathbf{u}'}(v_1^{(\ell)}, \ldots, v_t^{(\ell)}).$$

Consequently, we get

$$
\begin{aligned}
E^*[\, w(V \mid \mathbf{u}^c)\, I_{\mathbf{u}'}(V)\,] &= \sum_{t=1}^{\infty} E^*[\, w(V \mid \mathbf{u}^c)\, I_{t,\mathbf{u}'}(V_1,\ldots,V_t)\,] \\
&= \sum_{t=1}^{\infty} \int\!\!\ldots\!\int \prod_{k=1}^{t} \frac{f_k(v_k \mid u_k^c)}{f_k^*(v_k \mid u_k^c)}\, I_{t,\mathbf{u}'}(v_1,\ldots,v_t) \\
&\qquad \times \prod_{k=1}^{t} f_k^*(v_k \mid u_k^c)\, dv_1, \ldots, dv_t \\
&= \sum_{t=1}^{\infty} \int\!\!\ldots\!\int I_{t,\mathbf{u}'}(v_1,\ldots,v_t) \prod_{k=1}^{t} f_k(v_k \mid u_k^c)\, dv_1, \ldots, dv_t \\
&= \sum_{t=1}^{\infty} E[\, I_{t,\mathbf{u}'}\,] \\
&= E[\, I_{\mathbf{u}'}\,] \\
&= P(\mathbf{u}' \mid \mathbf{u}^c).
\end{aligned}
$$

Thus the importance sampling estimator (6.8) is unbiased (recall that the simulation data is i.i.d. and thus, $\hat{P}(\mathbf{u}' \mid \mathbf{u}^c)$ is unbiased if and only if it is unbiased for $L = 1$.)

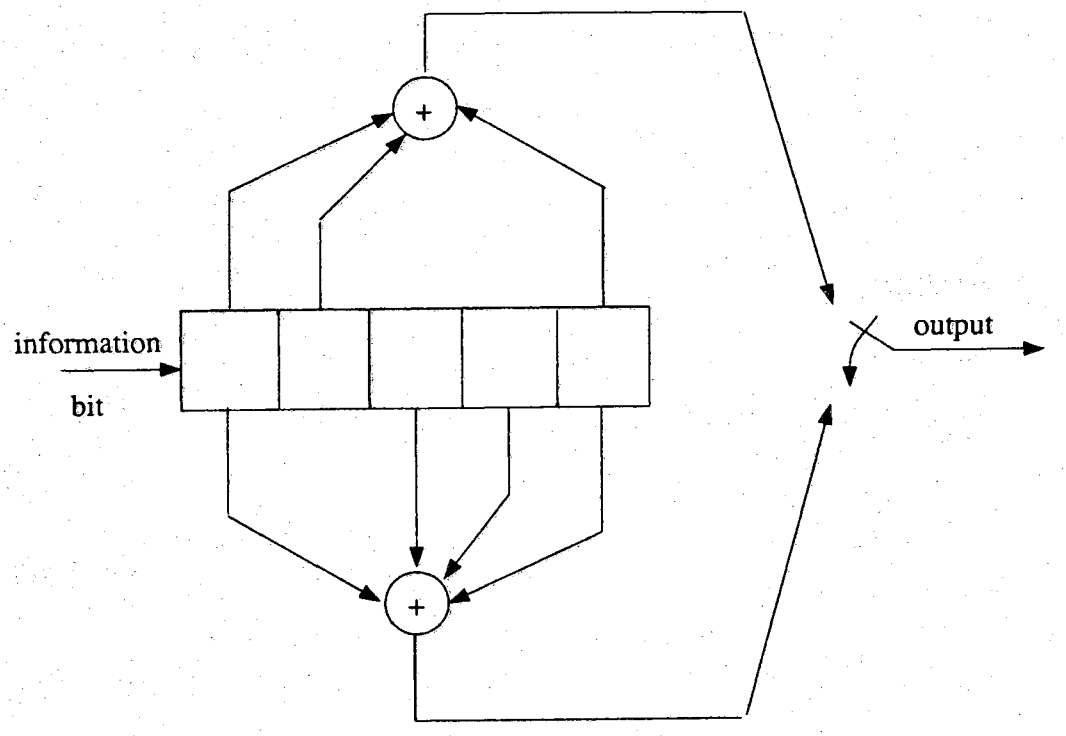
Finally, notice that because $\hat{P}(\mathbf{u}' \mid \mathbf{u}^c)$ is unbiased, it follows that importance sampling estimators for $\overline{N}_b$ and P_b are also unbiased.

6.5 Numerical Examples

In this section, we shall present some simulation results that illustrate the power and accuracy of the error event simulation method. Throughout this section, we shall consider rate 1/2 convolutional codes that operate on the binary symmetric channel. All of these codes will have the same constraint length 5. However, the generators for these codes will be different. The first of these codes is the constraint length 5 code presented in the previous chapters. This code will be referred to as code 1. The second and third codes will be refered to as code 2 and code 3 respectively. The

convolutional encoders that generate these codes are shown in Figures 6.2 and 6.3.

Now for a given error burst $\mathbf{u}'$, we shall let $d(\mathbf{u}^c,\mathbf{u}')$ denote the hamming distance between $\mathbf{u}^c$ and $\mathbf{u}'$. Furthermore, we shall define the *information weight* of a given error burst $\mathbf{u}'$ as the number of information bit errors $n_b(\mathbf{u}^c,\mathbf{u}')$. Tables 6.1-6.3 show all the error events with hamming distances that are smaller than 10 that are associated with code 1 , code 2, and code 3. These tables also list the hamming distances and the information weights associated with these codes.

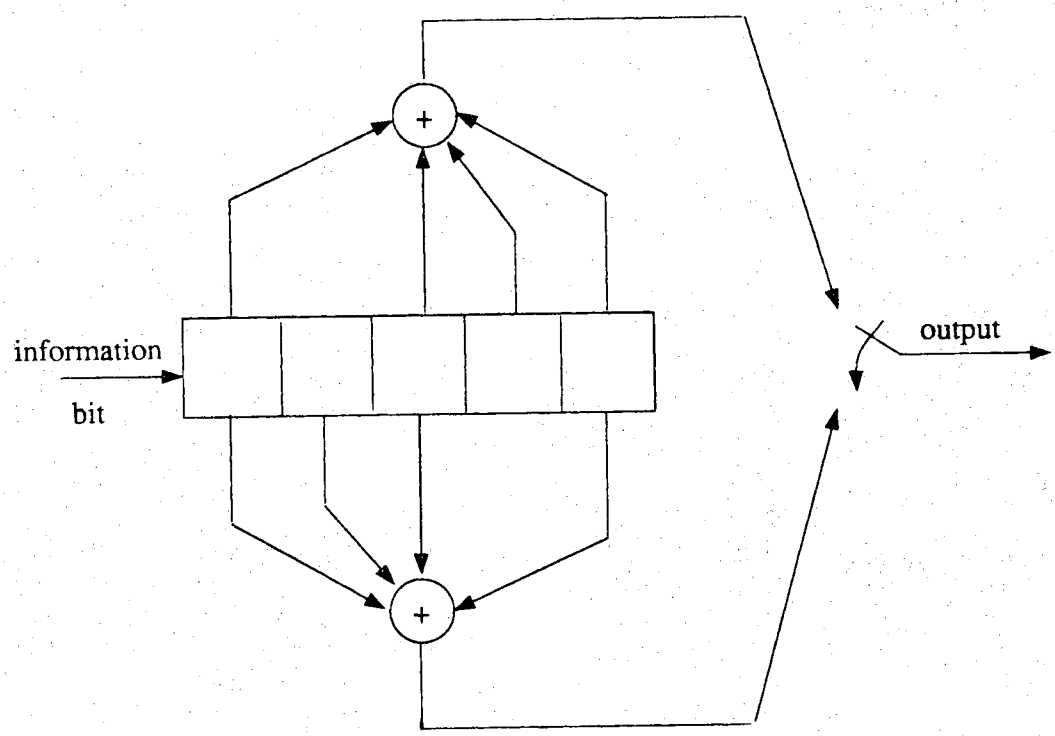


$g_1 = 31$ (octal)

$g_2 = 27$ (octal)

states = 16

Figure 6.2: The convolutional encoder for code 2.



$g_1 = 23$ (octal)

$g_2 = 35$ (octal)

states = 16

Figure 6.3: The convolutional encoder for code 3.

Table 6.1: A list of all the error bursts with hamming distances less than 10 for code 1.

Weight	Distance	Error Bursts
1	7	1110100111
3	7	1101000000010111
2	8	110100111011
4	8	1110010000001001011
6	8	11010000001000000001001011
3	9	11011101001011
5	9	11010000110001001011
5	9	11100100001010001011
7	9	11010000001000001010001011
2	10	11100111010111
2	10	1110101001100111
4	10	1101111010001011
2	10	1110100100100111
4	10	11100100110000010111
4	10	11010000001011010111
6	10	11010000110010010001011
4	10	11101010100000000010111
4	10	11010000000011001100111
6	10	11100100001001100001011
4	10	11101001000100000010111
4	10	11010000000101001010100111
6	10	11010000001000110000010111
8	10	11010000001000001001100001011
6	10	11010000000110100000000010111
6	10	11010000000101000100000010111

Table 6.2: A list of all the error bursts with hamming distances less than 10 for code 2.

Weight	Distance	Error bursts
1	7	1110010111
3	7	1110100000001011
2	8	110111001011
4	8	110100100000100111
6	8	11010010000010000001011
3	9	11010010111011
5	9	11010001010000100111
5	9	11010010001100001011
7	9	110100010100000100000001011
2	10	11101011100111
2	10	1110011001010111
4	10	1101000101111011
2	10	111001010010010111
4	10	111010000011001001111
4	10	11101011010000001011
6	10	11010001010011000001011
4	10	11101000000010101010111
4	10	11100110011000000001011
6	10	11010001100100000100111
4	10	11101000000100010010111
4	10	11100101001010000000001011
6	10	11101000001100010000001011
8	10	110100011001000001000000001011
6	10	11101000000001011000000001011
6	10	111010000000100010100000001011

Table 6.3: A list of all the error bursts with hamming distances less than 10 for code 3.

Weight	Distance	Error bursts
1	7	1101011011
3	7	1110000000101011
2	8	111000110111
4	8	110110000010000111
6	8	11100000000100001000111
3	9	11101110000111
5	9	11100000110010000111
5	9	11011000000101000111
7	9	111000000001000000101000111
2	10	11011011101011
2	10	1101010110011011
4	10	1110110101000111
2	10	110101100001011011
4	10	1101100011000010101 1
4	10	1110000000111101011
6	10	11100000110001010001 11
4	10	1101010101000000101011
4	10	1110000000100110011011
6	10	1101100000011001000111
4	10	1101011000100000000101011
4	10	1110000000101000010110 11
6	10	1110000000010011000010101 1
8	10	11100000000100000110010001 11
6	10	1110000000100101000000101011
6	10	11100000001010001000000010101 1

Tables 6.4-6.6 show some simulation results for code 1, code 2 and code 3, respectively. The first two columns of these tables give the code's information weights and hamming distances. The third column gives the importance sampling estimates $\hat{P}(\mathbf{u}'|\mathbf{u}^c)$. Finally, columns 4 and 5 give the accuracy of the estimates $\hat{P}(\mathbf{u}'|\mathbf{u}^c)$ and the relative efficiency gains (reg) respectively. The estimates given in Tables 6.4-6.6 are based on 9,000 simulation runs. Specifically, we use $L = 1000$ simulation runs per error burst. For each of the codes that were simulated in this chapter, the corresponding error bursts used in the simulation are the first 9 error events listed in Tables 6.1-6.3.

The simulation model that we have used to obtain the simulation simulation results in this chapter was a non-stationary memoryless BSC model with a time varying crossover probability ε_{ki}^* for i = 1 or 2. Specifically,

$$\varepsilon_{ki}^* = \begin{cases} 1/2 & \text{if } u_{ki}^c \neq u_{ki}' \\ \varepsilon & \text{if } u_{ki}^c = u_{ki}' \end{cases} \tag{6.11}$$

for i=1 or 2. It is noted that in this context, the index k corresponds to the code symbol transmission time and the index i corresponds to the code symbol bit.

We should finally mention that we have also included in Table 6.7 some simulation results which list some importance sampling simulation estimates that were obtained using the Viterbi decoder instead of the stack algorithm sequential decoder[1]. These simulation results use the same error bursts and the same number of simulation runs as in table 6.4. It is noted that in this case, the performance of the stack algorithm

1. We are grateful to J. C. Chen for providing the Viterbi decoder simulation results. Mr. Chen is a Research Assistant of Professor Sadowsky at Purdue University.

decoder and the Viterbi decoder should be very similar. Consequently, the Viterbi decoder simulation results can be used as a verification of the accuracy of the estimates obtained using the stack algorithm decoder.

Table 6.4: Bit error probability estimates for code 1 with $\varepsilon = .04$, and $L = 1000$ simulation runs per error burst. W= information weight, D= hamming distance, A= relative accuracy estimates, and reg= relative efficiency gain estimates.

W	D	$\hat{P}(\mathbf{u}'\|\mathbf{u}^c)$	A	reg
1	7	$.951 \times 10^{-8}$	4.89 %	$.4 \times 10^{8}$
3	7	$.841 \times 10^{-8}$	5.33 %	$.4 \times 10^{8}$
2	8	$.895 \times 10^{-8}$	7.85 %	$.2 \times 10^{8}$
4	8	$.870 \times 10^{-8}$	7.98 %	$.2 \times 10^{8}$
6	8	$.961 \times 10^{-8}$	7.53 %	$.2 \times 10^{8}$
3	9	$.113 \times 10^{-9}$	5.95 %	$.3 \times 10^{10}$
5	9	$.135 \times 10^{-9}$	5.29 %	$.3 \times 10^{10}$
5	9	$.119 \times 10^{-9}$	5.77 %	$.3 \times 10^{10}$
7	9	$.115 \times 10^{-9}$	5.88 %	$.3 \times 10^{10}$

Table 6.5: Bit error probability estimates for code 1 with $\varepsilon = .04$, and $L = 1000$ simulation runs per error burst. W= information weight, D= hamming distance, A= relative accuracy estimates, and reg= relative efficiency gain estimates.

W	D	$\hat{P}(\mathbf{u}' \mid \mathbf{u}^c)$	A	reg
1	7	$.812 \times 10^{-8}$	5.46 %	$.4 \times 10^{8}$
3	7	$.867 \times 10^{-8}$	5.18 %	$.4 \times 10^{8}$
2	8	$.895 \times 10^{-8}$	7.85 %	$.2 \times 10^{8}$
4	8	$.793 \times 10^{-8}$	8.41 %	$.2 \times 10^{8}$
6	8	$.883 \times 10^{-8}$	7.91 %	$.2 \times 10^{8}$
3	9	$.106 \times 10^{-9}$	6.22 %	$.2 \times 10^{10}$
5	9	$.129 \times 10^{-9}$	5.46 %	$.3 \times 10^{10}$
5	9	$.125 \times 10^{-9}$	5.59 %	$.3 \times 10^{10}$
7	9	$.241 \times 10^{-9}$	53.2 %	$.2 \times 10^{8}$

Table 6.6: Bit error probability estimates for code 1 with $\varepsilon = .04$, and $L = 1000$ simulation runs per error burst. W= information weight, D= hamming distance, A= relative accuracy estimates, and reg= relative efficiency gain estimates.

W	D	$\hat{P}(u' \mid u^c)$	A	reg
1	7	$.964 \times 10^{-8}$	4.85 %	$.4 \times 10^{8}$
3	7	$.928 \times 10^{-8}$	4.98 %	$.4 \times 10^{8}$
2	8	$.870 \times 10^{-8}$	7.97 %	$.2 \times 10^{8}$
4	8	$.876 \times 10^{-8}$	7.95 %	$.2 \times 10^{8}$
6	8	$.805 \times 10^{-8}$	8.34 %	$.2 \times 10^{8}$
3	9	$.103 \times 10^{-9}$	6.34 %	$.2 \times 10^{10}$
5	9	$.122 \times 10^{-9}$	5.67 %	$.3 \times 10^{10}$
5	9	$.113 \times 10^{-9}$	5.97 %	$.3 \times 10^{10}$
7	9	$.121 \times 10^{-9}$	5.72 %	$.3 \times 10^{10}$

Table 6.7: Bit error probability estimates for code 1 using the Viterbi decoder with $\varepsilon = .04$ and $L = 1000$ simulation runs per error event. W= information weight, D= hamming distance, and A= relative accuracy estimates.

W	D	$\hat{P}(\mathbf{u}' \mid \mathbf{u}^c)$	A
1	7	$.808 \times 10^{-8}$	5.47 %
3	7	$.786 \times 10^{-8}$	5.58 %
2	8	$.876 \times 10^{-8}$	7.95 %
4	8	$.967 \times 10^{-8}$	7.50 %
6	8	$.735 \times 10^{-8}$	8.77 %
3	9	$.114 \times 10^{-9}$	5.91 %
5	9	$.132 \times 10^{-9}$	5.37 %
5	9	$.110 \times 10^{-9}$	6.07 %
7	9	$.123 \times 10^{-9}$	5.66 %

Now recall that given the estimates of $P(\mathbf{u}' \mid \mathbf{u}^c)$, then the estimates of the expected number of bit errors per correct decode $\overline{N}_b$ can be also estimated using the sum (6.6). Table 6.8 below lists the estimates of $\overline{N}_b$ for the three convolutional codes discussed earlier by using the simulation data listed in Tables 6.4-6.7, along with the sum (6.6).

Table 6.8: The expected number of bit errors per correct decode estimates for code 1, code 2, and code 3 with $\varepsilon = .04$, $L = 1000$ simulation runs per error event. A= relative accuracy estimates.

$\hat{\hat{N}}_b$					
Code 1	A	Code 2	A	Code 3	A
$.15 \times 10^{-6}$	3.7 %	$.14 \times 10^{-6}$	3.9 %	$.14 \times 10^{-6}$	3.8 %

Table 6.9 below compares the expected number per correct decode estimates for code 1 that were obtained from the simulation of the stack algorithm and the Viterbi decoders. Note that the difference between both estimates is insignificant.

Table 6.9: The expected number of bit errors per correct decode estimates for code 1 with $\varepsilon = .04$, and $L = 1000$ simulation runs per error event. A= relative accuracy estimates, SA denotes stack algorithm, and VD denotes Viterbi Decoder.

$\hat{\hat{N}}_b$			
SA	A	VD	A
$.15 \times 10^{-6}$	3.7 %	$.13 \times 10^{-6}$	3.9 %

In conclusion, this chapter has demonstrated that the error event simulation method used in conjection with importance sampling could prove to be an extremely powerful tool for performance evaluation of sequential decoders. This presentation has considered only the estimation of the bit error probabilities. However, as stated earlier, we should note that the error event simulation model can actually efficiently estimate

key parameters such as the error burst length, the error burst length distribution and so on.

CHAPTER 7
SEQUENTIAL EDGE LINKING SIMULATION

7.1 Introduction

In the previous chapters, we have demonstrated that the importance sampling techniques which we have presented were very efficient for estimating the key quantities that characterize the stack algorithm sequential decoders. In this chapter, we shall consider the application of importance sampling to a different problem. Specifically, we consider the simulation of the *sequential edge linking (SEL) algorithm*. This is a stack algorithm technique for detecting edges in images that has been proposed by Eichel and Delp [84-86].

We shall begin this chapter by presenting some background on sequential edge detection. We shall then develop various definitions which are needed for the discussions to follow. This development is then followed by the presentation of our importance sampling technique for simulating the SEL algorithm. The concluding section in this chapter discusses some of the key issues which are related to this application and the potential application of our technique to performance evaluation of the SEL algorithm.

7.2 The Edge Detection Problem

7.2.1 Introduction

Edge detection represents one of the first processing steps in image processing and computer vision. Research into methods of finding edges in noisy images has been an active field of investigation for many years. Reflecting this importance, the literature devoted to this problem is enormous [67-86] and many different approaches have been proposed.

The general edge detection problem is illustrated in Figure 7.1. Here the input is a two-dimensional *digital image*. By digital, we mean that the image intensity function is not continuous, but rather, defined on an array of points. The values of the intensity function at these points represent the brightness or gray level of the digital image. These image elements are called *pixels* (for "picture elements"). *Edges* of interest in real scenes are intuitively defined as picture elements which lie on the boundary between regions of different intensities or gray levels; that is, edges are represented in an image as a discontinuity in intensity. Hence the task of an *edge detector* becomes one of identifying intensity discontinuities. Because edge elements are associated with a rapid change in gray level as a function of the spatial domain the edge detector is usually implemented as some form of differential operator or high-pass filter which emphasizes high spatial frequency components and suppresses areas with little change in intensity. The interested reader is referred to [71], [78], [82] and references therein for in depth discussion of edge operators and the general edge detection problem.

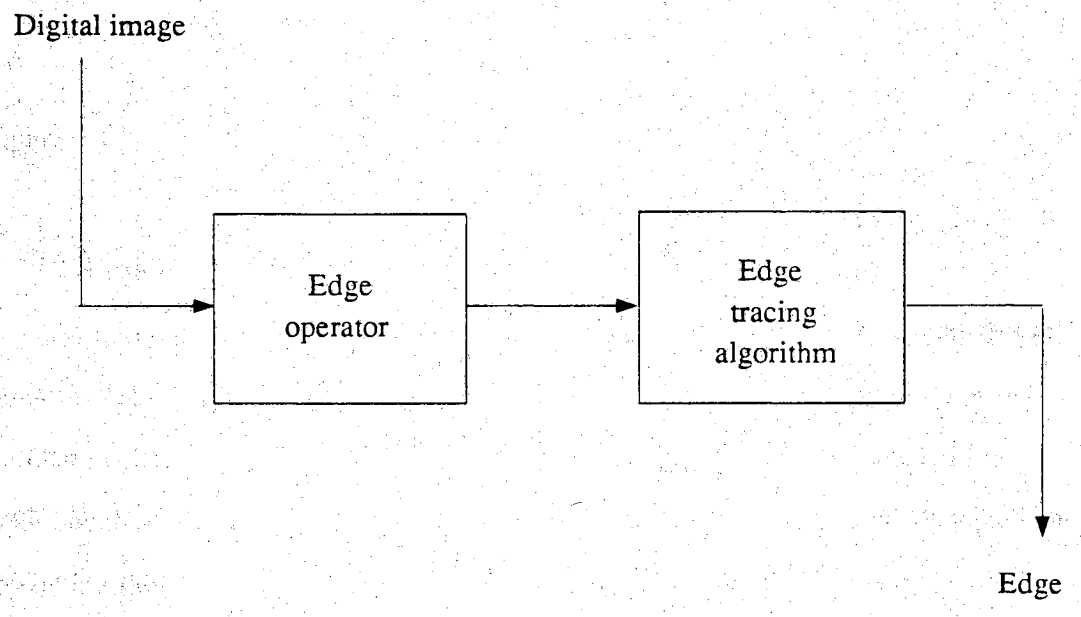


Figure 7.1: The general edge detection problem.

7.2.2 Digital Images and Random fields

A digital image shall refer to a sample function of a two-dimensional discrete random field [71]. Such sample functions will consist of a rectangular array of numbers called *pixels*. These pixels may represent the brightness or the gray level at each point of the digital image. The points of the rectangular lattice at which the pixels are defined are called *pixel locations* or *pixel indices,* their spacing is uniform and equal in both directions; furthermore, they are indexed by N^2 where N is the set of integers. Due to the rectangular nature of the lattice, each pixel has a unique set of eight *neighbors*. For a given random field, we shall let Y_{ij} denotes the pixel or observation at the pixel location $\underline{i} \triangleq (i , j)$. Furthermore, we shall assume the existence

of two conditional densities on the random field. The first is the conditional density for the pixels on the edge, namely

$$p_1 (Y_{ij} = y) \triangleq \mathcal{P}(Y_{ij} = y \mid \underline{i} \text{ is on an edge }). \tag{7.1}$$

The second is the conditional density for the pixels not on the edge, namely

$$p_0 (Y_{ij} = y) \triangleq \mathcal{P}(Y_{ij} = y \mid \underline{i} \text{ is not on an edge }). \tag{7.2}$$

Again, we note that the term *"density"* whenever used will mean either a probability distribution function or a probability mass function depending on the context.

7.2.3 Image Paths

An *image path* will be defined as a sequence of successively connected pixel locations such that for any subset of three pixel locations in this sequence, the directions defined by the first two pixel locations and the second two pixel locations differ by $\pi/4$ or less.

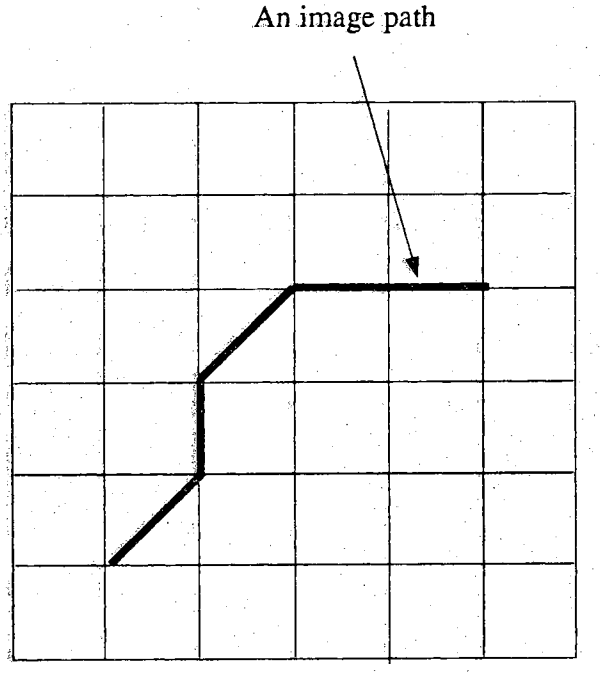


Figure 7.2: An example of an image path.

An image path **m** of length n can be specified in one of two ways. The first is simply the ordered sequence of pixel locations comprising the image path:

$$\mathbf{m} = [\underline{\mathbf{i}}_0, \underline{\mathbf{i}}_1, \cdots, \underline{\mathbf{i}}_n].$$

The second is by specifying a *root pixel location* $\underline{\mathbf{i}}_0$, a *start direction* $\vec{\mathbf{d}}_0$, and an ordered set of letters $a_1, a_2, \cdots, a_n$:

$$\mathbf{m} = \underline{\mathbf{i}}_0 \times \vec{\mathbf{d}}_0 \times [\, a_1, a_2, \cdots, a_n \,] \quad ; \quad a_i \in \{L,S,R\} \tag{7.3}$$

where the letters L,S, and R stand for left, straight, and right respectively.

The location of each observation in the rectangular lattice can be obtained in a recursive fashion from the letters a_i, namely each pixel location $\underline{i}_n$ in a given image path can be obtained from the pixel location $\underline{i}_{n-1}$ by moving in the image array in the direction $\vec{d}_{n-1}$. The direction to the n + 1 pixel location is then obtained from $\vec{d}_n$ and a_{n+1} according to the following rule:

$$\vec{d}_{n+1} = \begin{cases} \vec{d}_n + \pi/4 & \text{if } a_{n+1} = L \\ \vec{d}_n & \text{if } a_{n+1} = S \\ \vec{d}_n - \pi/4 & \text{if } a_{n+1} = R \end{cases}$$

7.3 Sequential Edge Detection

The edge detection problem can be formulated as a *tree searching* problem. Given a sequence of turns $(a_1, a_2, \cdots)$ on the digital image, then these relative directions are equivalent to branching possibilities in a ternary tree with each tree branch being an L, S, or R (see Figure 7.3.)

In real applications, the size of a typical digital image is large and hence the resulting decision tree is enormous. Consequently, an exhaustive search approach which examines every possible candidate edge contour and chooses the "best" according to some predetermined criterion is not possible. Sequential tree searching algorithms provide a practical alternative through a structured search strategy in which the paths are extended sequentially, with the current most probable path extended by one observation at each iteration.

For this approach to succeed in finding edges, a means of comparing all paths hypothesized must be provided. This comparison is accomplished by associating

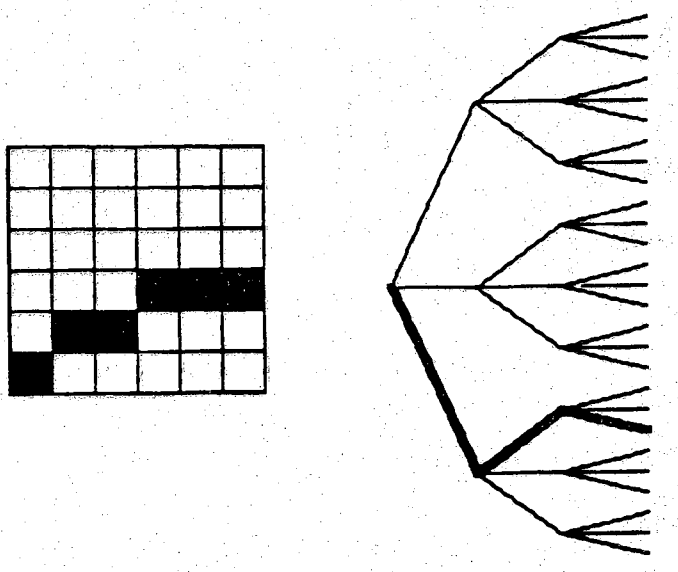
with each path a statistic called a *path metric* which is an indicator of the likelihood that the corresponding path coincides with the true edge. Consequently, only the most probable paths which presumably should include the true edge path are extended by the searching algorithm.

There are many sequential edge detection algorithms which have been proposed in the literature. Examples include the work of Chien and Fu [70] and Martelli [68], [72]. Specifically, Chien and Fu proposed the use of what is known as depth-first tree search to extract the heart boundary from digitized chest X rays. In a similar fashion, Martelli formulated the edge detection problem as a graph search and uses the A^* algorithm described by Nilson [67] to minimize a cost function determined by the heuristics of the problem at hand. Both of the above methods use a cost function which is highly specialized to the type of image under consideration and thus limit their applications.

Many other investigators have attempted to employ sequential tree or graph searching algorithms in the context of edge detection. Ashkar and Modestino [74] seem to have come the closest to a truly sequential search. However, their metric suffers from two problems. Actually, the metric used in their search technique is purely ad hoc and hence no analytical treatment of the search dynamics can be undertaken. More seriously, their technique makes explicit use of experimentally determined parameters and look-up tables and requires a nominal or a "prototype" contour to guide the search. This represent very high quality a-priori information and thus limit the applicability of this technique.

The interested reader is referred to [71], [82] and references therein for more background and in depth discussions of sequential edge detection.

Now notice that in order to relate the edge detection problem to tree searching, we must first model the collection of possible edges as a tree. Given two successive edge pixel locations, then the edge can be expressed as a sequence of relative direction changes, and the relative directions are equivalent to branching possibilities in a tree. For example, Figure 7.3a illustrates an edge in a 6×6 noiseless digital image for which all changes of directions are of $\pi/4$ or less, namely the relative directions are 45 degrees to the left, straight, or 45 degrees to the right (recall section 7.2.3). Figure 7.3b is a tree for which the starting point is the first two pixel locations in the lower left corner of Figure 7.3a. The bold line in Figure 7.3b represents the edge in Figure 7.3a. Observe that knowing two successive pixel locations is equivalent to knowing the 2'nd pixel location and the direction from the 1'st to the 2'nd pixel location, i.e, two successive pixel locations on the edge define a "from-to" direction.



(a) (b)

Figure 7.3: (a) an edge in 6×6 noiseless image; (b) the corresponding tree representation of the edge in Figure 7.3 (a).

With this background on sequential edge detection, it becomes clear that one of the critical issues in this problem is the selection of a good initial point; as well as, a good initial direction. The performance of the entire technique will depend upon the identification of both of these quantities. Since edges are associated with a rapid change in intensity or gray level as a function of the spatial domain, it follows that the larger the magnitude of the gradient at a given point in the image, the higher the probability that such a point actually lies on an edge. Thus root pixel locations can be easily obtained by imposing a high threshold on the gradient magnitude output of the

pre-processing stage which proceeds the search in general. Likewise, because edges are searched in a direction perpendicular to the gradient direction, the root direction $\vec{d}_0$ at the root pixel location $\underline{i}_0$ can be also obtained from the output of the gradient operator at the pre-processing phase. In summary, the pre-processing of the image provides $\underline{i}_0$, and $\vec{d}_0$. The interested reader is referred to [85] and references therein for more discussion about the root pixel location and the start direction selection problem.

7.4 Sequential Edge Linking

In this section we shall describe a sequential edge detection scheme known as *sequential edge linking* or *SEL* which has been proposed by Eichel and Delp [84-86]. The SEL algorithm is a sequential tree searching technique which is inspired by the sequential decoding of convolutional codes. The main difference between the SEL algorithm and the other sequential techniques discussed earlier is that the SEL path metric used is based on a dynamic model of the edge behavior.

The SEL algorithm is based on the stack algorithm. Due to the random field model on which paths are based, a slight modification of the stack algorithm must be accommodated. Actually, the root pixel location $\underline{i}_0$ and the root direction $\vec{d}_0$ must be specified before starting the stack algorithm search. This modification is needed in order to specify the initial search direction. As noted earlier, these quantities can be provided by the pre-processing phase which proceeds the search.

In the SEL algorithm, the edge sequence $\{ a_1, a_2, \cdots \}$ in the digital image is modeled as a K'th order Markov Chain [63]. In this model, the state of the process X_n for $n \geq 0$ is defined to be the last K transition letters. That is,

$$X_n \triangleq (a_n, \cdots , a_{n-K+1}) \tag{7.4}$$

so that when the process enters the state at $n+1$ from that at n, it outputs a letter $a_{n+1} \in \{ L,S,R \}$. In this model, the initial state X_0 is assumed to be fixed and given. By letting $\mathcal{F}_n \triangleq \sigma(X_0, X_1, \cdots, X_n)$, then by the Markov assumption it follows that

$$\mathcal{P}(X_{n+1} = x_{n+1} \mid X_n = x_n, \mathcal{F}_n) = \mathcal{P}(X_{n+1} = x_{n+1} \mid X_n = x_n) \tag{7.5}$$

or equivalently,

$$\mathcal{P}(a_{n+1} \mid \mathcal{F}_n) = \mathcal{P}(a_{n+1} \mid X_n).$$

Consequently, it follows that the a-priori probability of an edge path **m** of length n is

$$\begin{aligned} \mathcal{P}(\mathbf{m} \mid X_0) &= \mathcal{P}(a_1, a_2, \cdots, a_n) \\ &= \prod_{i=1}^{n} \mathcal{P}(X_i \mid X_{i-1}). \end{aligned} \tag{7.6}$$

As has been pointed out earlier, most of the sequential edge detection schemes proposed to date involve nodes metrics which tend to be heuristic. As stated previously, the SEL algorithm path metric is based on a dynamic model of the edge behavior. This path metric is defined as follows: if we let γ be some node at depth n in the tree, and $\mathbf{m} = [\underline{i}_0, \underline{i}_1, \cdots, \underline{i}_n]$ be its corresponding image path on the image, then the metric at node γ is

$$\mathbf{M}_\gamma \triangleq \sum_{k=1}^{n} [\ln(\frac{p_1(y_k)}{p_0(y_k)}) + \ln(\mathcal{P}(X_k \mid X_{k-1}))] \tag{7.7}$$

where y_k is the pixel value or observation at $\underline{i}_k$, $p_0(.)$ and $p_1(.)$ are the conditional densities defined on the random field, and X_k is the state in the SEL Markov model. Observe that this metric involve two different components which play different roles. The first is a function of the data in the real image, and the second component is just the a-priori probability of the hypothesized path.

7.5 Error Segments and Remerging

Because of the two-dimensional lattice structure of the random field, it is possible for incorrect image paths to merge with the correct edge path. These "remerged paths" correspond to incorrect paths in the tree which behave exactly like the correct path after the point of remerging. On the image, however, a remerged path corresponds to a hypothesized edge which briefly diverges from the correct edge path. Consequently, the hypothesized image path contains an *error segment* of some length n (Recall that the same phenomenon occurs in the sequential decoding of convolutional codes.) For example, Figure 7.4a below shows a hypothesized image path **m** which coincides with the edge path **e** up to some pixel location **i**, branches off at **i**, and then remerges with **e** at some pixel location **j**. In this case, **m** contains an error segment of length 3.

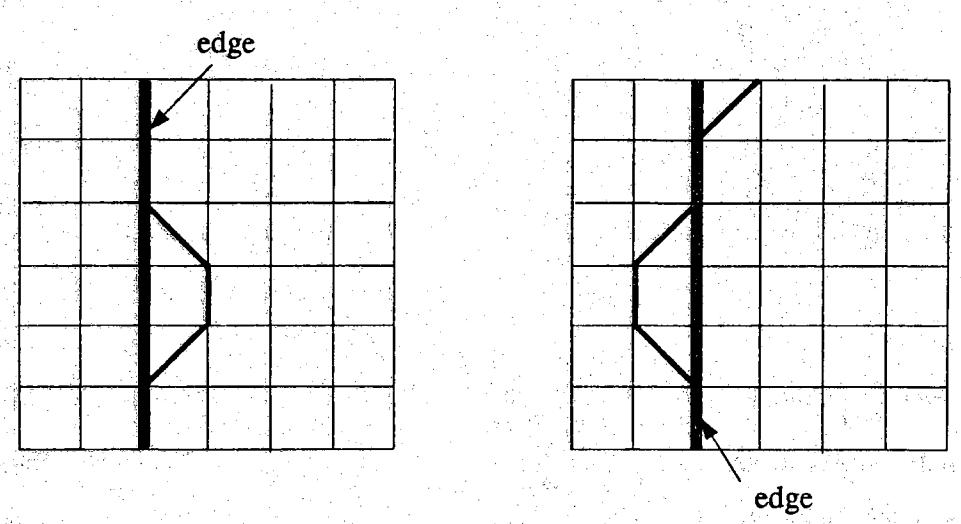


Figure 7.4: An illustration of the remerging phenomenon: (a) a remerging image path which terminates at the edge and (b) a remerging image path which eventually diverges from the edge.

Recall that the remerging phenomenon corresponds to incorrect paths in the tree which behave exactly like the correct path after the point of remerging. It follows that the notion of a node in the tree being "correct" becomes somewhat ambiguous. Thus, a precise definition of "correctness" in this context is needed.

Let γ be some node at depth n in the corresponding tree of a digital image. Because of the many to one correspondence between nodes in the tree and nodes in the image, it follows that node γ is uniquely represented by an image path of depth n $\mathbf{m} = [\, \underline{i}_0^{\gamma}, \underline{i}_1^{\gamma}, \cdots, \underline{i}_n^{\gamma} \,]$. For such a node, we define

$\underline{i}_n^\gamma \triangleq$ leading pixel location of node γ,

and

$\underline{i}_n^\gamma, \cdots, \underline{i}_{n-k+1}^\gamma \triangleq$ k leading pixel locations of node γ.

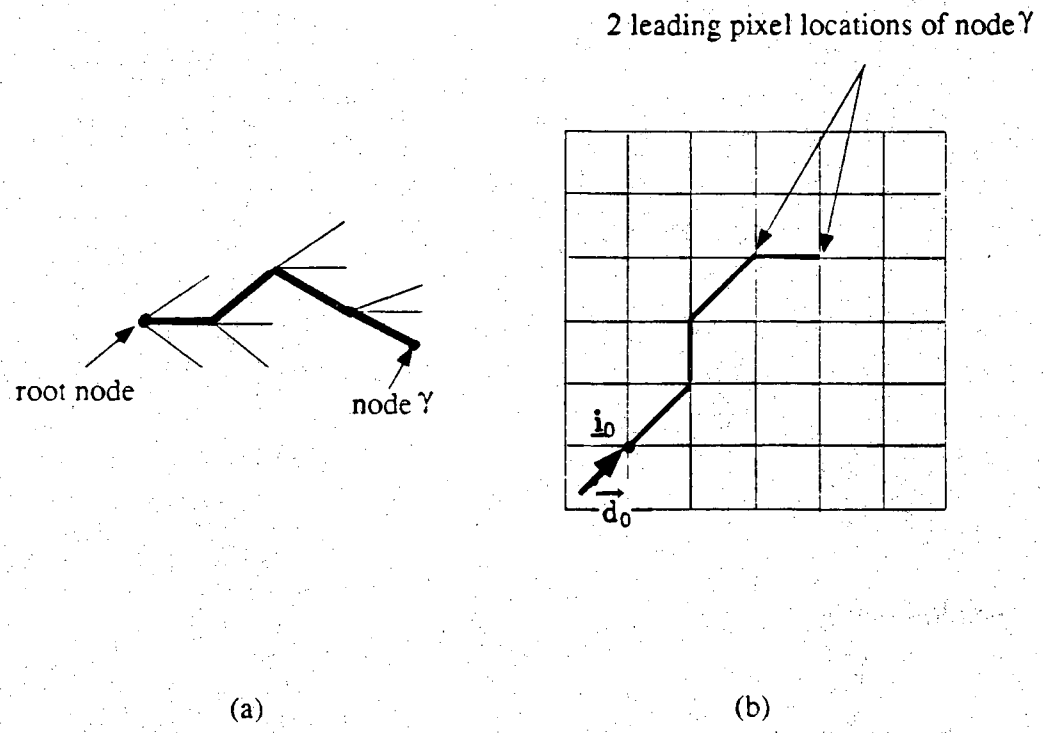


Figure 7.5: An illustration of the relationship between a node γ in the tree and its corresponding image path. (a) shows node γ, and (b) shows its corresponding image path.

Observe that for the node shown in Figure 7.5, the corresponding SEL Markov state model is simply the last two turns or transition letters; i.e, (R,R) (assuming a 2'nd order Markov model). This is determined by the 4 leading pixel locations of node γ.

In general, the SEL Markov state model which corresponds to a given node is determined by its K+2 leading pixel locations where K is the Markov Chain order.

We are now ready to define the notion of "correctness" in the context of sequential edge detection.

Definition: Let γ be a given node in the tree. Then node γ is said to be *k-correct* if its k leading pixel locations are on the edge.

Now let γ be a k-correct node at depth n in the tree. Next suppose that a correct decision at node γ has been made.

Define

$$P_\gamma = (\gamma, \beta_1, \beta_2, \cdots)$$

$\triangleq$ The terminal path emanating node γ (final selected path)

and

$$\ell_\gamma \triangleq \min \{ i : \beta_i \text{ is k–correct} \}.$$

Then,

$$L_\gamma = \begin{cases} \ell_\gamma - (k-1) & \text{if } \beta_1 \text{ is k–correct} \\ 0 & \text{if } \beta_1 \text{ is not k–correct} \end{cases} \tag{7.8}$$

$\triangleq$ length of branching errors following

the correct decision at node γ.

Consequently, $(\gamma, \beta_1, \cdots, \beta_{L_\gamma})$ is an *error burst* of length L_γ, or simply a *burst of* L_γ *branching errors* which is immediately proceeded by a k-correct node and immediately followed by the detection of a k-correct node. Of course, we must allow

the possibility of a correct branch decode immediately following the correct decode at node γ. In this case, we say the error event is *trivial,* and hence $L_\gamma = 0$. Finally, if we define

$$\bar{L}_b = E[\, L_\gamma \mid \gamma \text{ is m–correct} \,] \tag{7.9}$$

then, $\bar{L}_b$ is precisely *the expected number of branching errors per correct decision.*

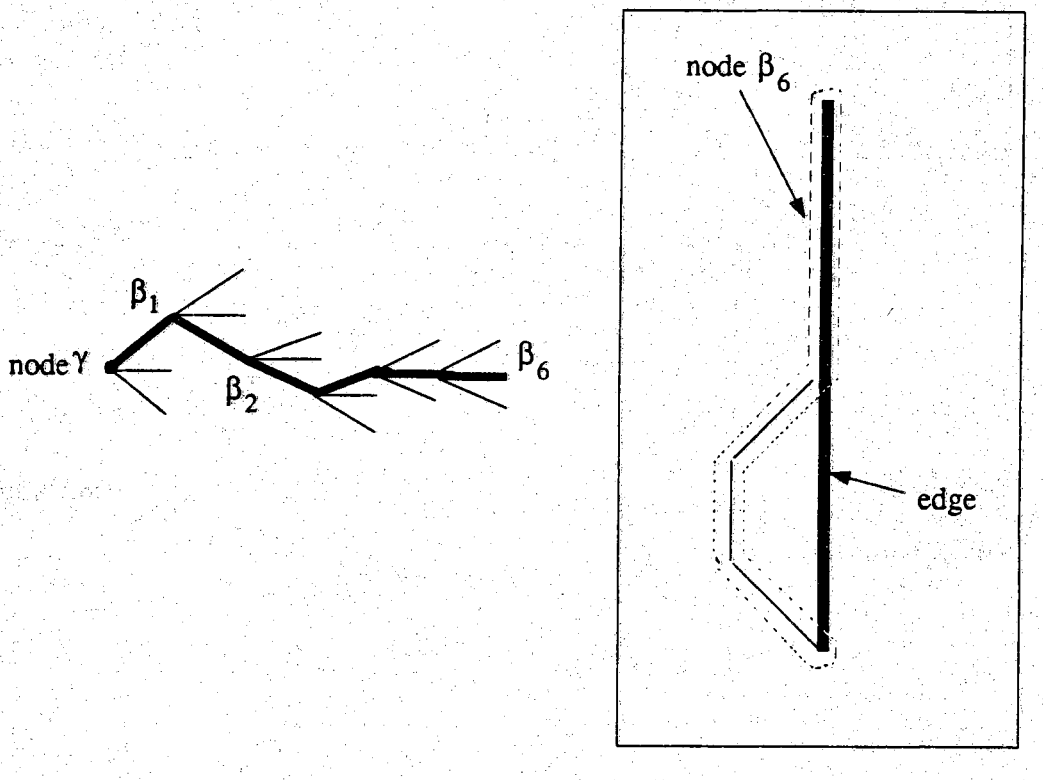


Figure 7.6: An illustration of an error burst of length 3 assuming k = K+2 where K is the order of SEL Markov model (K = 2): (a) shows the error burst and (b) shows its corresponding error segment on the digital image.

7.6 Sequential Edge Linking Simulation

Our main objective in this section is to develop a simulation technique for simulating the SEL algorithm which exploits the importance sampling principle. We begin by developing various definitions which are needed for the discussions to follow.

7.6.1 Preliminaries

Consider the random field defined in section 7.2.2 and let Y_{ij} denotes the pixel value or observation at a given pixel location (i,j) in the field. Next let

$\mathbf{D}^{(\ell)}$ = pixel index set for the ℓ'th simulation (a random set)

$\mathbf{Y}^{(\ell)} = \{ Y_{ij}^{(\ell)} : (i,j) \in \mathbf{D}^{(\ell)} \}$ (i.e., the data record for the ℓ'th simulation)

In the sequel, we shall consider the problem of estimating expectations of random variables that are $\mathcal{G}_\gamma$–measurable for some node γ. That is, we shall consider the basic problem discussed in Section 3.4.2. Recall that such a problem can be stated as follows:

The Basic Problem: Given the event E_γ, estimate

$$E[X \mid E_\gamma]$$

where X is a $\mathcal{G}_\gamma$-measurable random variable.

Notice that because the SEL algorithm is based on the stack algorithm, it follows that the problem of estimating most of the key parameters associated with the SEL algorithm can be formulated as in the basic problem. In addition, note that from Theorem 4.1, it follows that for any $\mathcal{G}_\gamma$–measurable random variable which is associated with the SEL algorithm we have,

$$E[\, X \mid E_\gamma \,] = E[\, X \mid \gamma \text{ is the root node }]$$

7.6.2 The Importance Sampling Estimator

Let $\mathbf{D}^{(0)}$, $\mathbf{Y}^{(0)}$, and Y_{ij} be defined as in section 7.6.1. Likewise, let p_0 and p_1 be the two conditional densities defined on the random field. Recall that these densities were defined as follows:

$$p_0(\, y \,) = \mathcal{P}(\, Y_{ij} = y \mid (i,j) \text{ is not on an edge }) \tag{7.10}$$

and

$$p_1(\, y \,) = \mathcal{P}(\, Y_{ij} = y \mid (i,j) \text{ is on an edge }). \tag{7.11}$$

Next for any finite set of pixel locations D, define

$$f_Y(\, y \,;\, D \,) = \prod_{(i,j)\in D} f_{ij}(\, Y_{ij} \,) \tag{7.12}$$

where

$$f_{ij}(\, y \,) = \begin{cases} p_1(\, y \,) & \text{if } (i,j) \text{ is on edge} \\ p_0(\, y \,) & \text{if } (i,j) \text{ is not on an edge} \end{cases} \tag{7.13}$$

and Y is the simulation data record associated with the pixel locations in D. Observe that $f_Y(.\,;\,.)$ is simply a finite order distribution on the random field. Furthermore, note that conceptually $f_Y(\, y \,;\, D \,)$ is just the joint probability that the random variables $Y_{ij} \in dy_{ij}$ for all (i,j) in D.

Now let us consider the problem of estimating[1]

$$\alpha = E[\, X \,] \tag{7.14}$$

1. For simplicity, we shall drop the conditioning on the event E_γ in our notation. Hence, hereafter we will write $E[\, X \,]$ instead of $E[\, X \mid E_\gamma \,]$.

where X is $\mathcal{G}_Y$–measurable. As we have seen previously, importance sampling is applied by simulating using a different joint density $f^*_Y(.\,;.)$ instead of $f_Y(.\,;.)$. By letting p^*_0 and p^*_1 be the *importance sampling densities* defined on the random field, then for any finite set of pixel locations D,

$$f^*_Y(\,y\,;D\,) = \prod_{(i,j)\,\in\,D} f^*_{ij}(\,Y_{ij}\,) \tag{7.15}$$

where

$$f^*_{ij}(\,y\,) \triangleq \begin{cases} p^*_1(\,y\,) & \text{if (i,j) is on an edge} \\ p^*_0(\,y\,) & \text{if (i,j) is not on an edge} \end{cases} \tag{7.16}$$

by letting

$$W^{(\ell)} \triangleq \prod_{(i,j)\,\in\,D^{(\ell)}} \frac{f_{ij}(\,Y_{ij}\,)}{f^*_{ij}(\,Y_{ij}\,)} \tag{7.17}$$

be the importance sampling weight for the ℓ'th simulation, then the importance sampling estimator for α is

$$\hat{\alpha}^* = \frac{1}{L}\sum_{\ell=1}^{L} X^{(\ell)}\,W^{(\ell)} \tag{7.18}$$

where $X^{(1)}, \ldots, X^{(L)}$ are independent simulation data records which are generated from the simulation density $f^*_Y(.;.)$.

Claim 4.1: $E^*[\,X^{(\ell)}\,W^{(\ell)}\,] = E[\,X\,]$.

Proof: Let $\mathcal{D}$ be the set of all pixel locations, $\hat{D}$ be some subset of $\mathcal{D}$, and $\hat{Y}$ be the simulation data record which is associated with $\hat{D}$. Next note that on the event $D^{(\ell)} = \hat{D}$, there exists some function $g(.;\hat{D})$ such that $X^{(\ell)} = g(Y^{(\ell)};\hat{D})$. Consequently,

for $\ell = 1,2, \cdots, L^2$

$$E^*[\, X^{(\ell)} W^{(\ell)} \,] = \sum_{\hat{D} \in \mathcal{D}} E^*[\, g(\hat{Y}^{(\ell)}; \hat{D})\, W^{(\ell)} ; \mathbf{D}^{(\ell)} = \hat{D} \,]$$

$$= \sum_{\hat{D} \in \mathcal{D}} \iint g(\hat{y}; \hat{D})\, W^{(\ell)} \prod_{(i,j) \in \hat{D}} f^*_{ij}(Y_{ij})\, I_{\{\mathbf{D}^{(\omega)} = \hat{D}\}}(\hat{y})\, d\hat{y}$$

$$= \sum_{\hat{D} \in \mathcal{D}} \iint g(\hat{y}; \hat{D}) \frac{\prod_{(i,j) \in \hat{D}} f_{ij}(Y_{ij})}{\prod_{(i,j) \in \hat{D}} f^*_{ij}(Y_{ij})} \prod_{(i,j) \in \hat{D}} f^*_{ij}(Y_{ij})\, I_{\{\mathbf{D}^{(\omega)} = \hat{D}\}}(\hat{y})\, d\hat{y}$$

$$= \sum_{\hat{D} \in \mathcal{D}} \iint g(\hat{y}; \hat{D}) \prod_{(i,j) \in \hat{D}} f_{ij}(Y_{ij})\, I_{\{\mathbf{D}^{(\omega)} = \hat{D}\}}(\hat{y})\, d\hat{y}$$

$$= E[\, X \,].$$

□

Claim 4.2: $\hat{\alpha}^*$ is an unbiased estimator of α.

Notice that the proof of Claim 4.2 follows from Claim 4.1 because the simulation data $X^{(1)}, \cdots, X^{(L)}$ are independent random data records that are generated from the importance sampling distribution $f^*_Y(.;.)$.

---

2. The double integral sign $\iint$ in the next equations and elsewhere indicates a multi-dimensional integration.

7.6.3 Termination of the Simulation

The SEL simulation termination rule which we shall use for the SEL algorithm simulation is basically the Δ–stopping rule presented in the previous chapter. The Δ–stopping rule was slightly modified here in order to take into account the new definition of "correctness" that we have introduced in the sequential edge detection context. Our new termination strategy is called the Δ'–stopping rule.

Assume that γ is the root node and suppose that the ℓ'th simulation starts at $t = 0$. Then, the Δ'–stopping rule is

The Δ'–Stopping Rule (For the ℓ'th simulation):

Initialize: Start the search at node γ.

DO j=1, $\cdots$

Simulate until only one subtree is active, call it S_{β_j}.

IF β_j is k-correct **THEN**

STOP

ELSE

Delete inactive subtrees from the stack.

Reassociate the remaining stack nodes to subtrees S_δ with $\delta \in D_{\beta_j}$.

Compute $\overline{M}_\delta(t)$ for $\delta \in D_{\beta_j}$.

END IF

CONTINUE

Note that the basic idea behind the above rule is again to discard the inactive trees as the search progresses and concentrate on only the active ones which contain the TOS node.

7.6.4 Examples and Discussions

In this chapter, we have presented a new technique for the simulation of the SEL algorithm using importance sampling. when properly implemented, this technique can be used to efficiently estimate most of the key parameters which are associated the SEL algorithm. In fact, in addition to the basic estimation of error probabilities, this technique can be applied to estimate

1) The expected number of metric computations;
2) The distribution of computation;
3) The number of branching errors caused by an error burst;
4) The error burst length; and
5) The error burst length distribution, etc.

Estimating these parameters using conventional Monte Carlo simulations would most likely require a lot of simulation runs, especially when the image is not too noisy.

Certainly, the most important issue in this importance sampling technique is the design of importance sampling simulation models. Actually, as it should be clear from the previous chapters the critical element of importance sampling is the choice of the importance sampling density $f_Y^*(.;.)$. This density should be selected in such a way that it will approximate the unconstrained optimal importance sampling density (2.8), and at the same time, be the optimal density within the class of all candidate simulation densities which can be practically implemented. The knowledge of the system to be

simulated shall also be exploited when the selection of a suitable candidate simulation density is to be made. In fact, importance sampling works because it allows the simulation designer to apply his knowledge of the system to be simulated in order to emphasize the "important" events in the simulation.

In this application, our knowledge of the algorithm behavior and the "important" events to be simulated supports the use of a *non-stationary simulation model.* In fact, our experience indicates that whenever incorrect paths are hypothesized, either they will eventually become inactive as their lengths get longer and longer, or their corresponding image paths eventually merge with the edge path and once they do, they start behaving like the correct path. The practical implication of this is that as we get far away from the root node, large excursions from the edge path become scarce. In other words, large excursions from the correct path will probably occur around the root node. As these excursions get larger and larger, more opportunities for the incorrect image paths to merge with the edge path are created and again once they do, the excursions from the edge path become smaller as we get far away from the root node. As a consequence, if we assume that γ is the root node and let X be a $\mathcal{G}_\gamma$–measurable random variable, then it follows that most the pixels which will determine X will be around the root pixel location $\underline{i}_0$. Consequently, a non-stationary density $f^*_Y(.;.)$ which 1) makes the image around $\underline{i}_0$ more noisy than the actual operating conditions; and 2) decays to the true distribution $f_Y(.;.)$ in some fashion as we get farther and farther from $\underline{i}_0$ would be more efficient than a stationary density which will end up simulating unimportant events. Figure 7.8 illustrate this by showing an example of the pixel locations visited by the SEL algorithm when (a) the simulation density $f^*_Y(.;.)$ is a non-stationary density which decays to $f_Y(.;.)$ as the search gets farther and farther from $\underline{i}_0$; and (b) the simulation density $f^*_Y(.;.)$ is stationary.

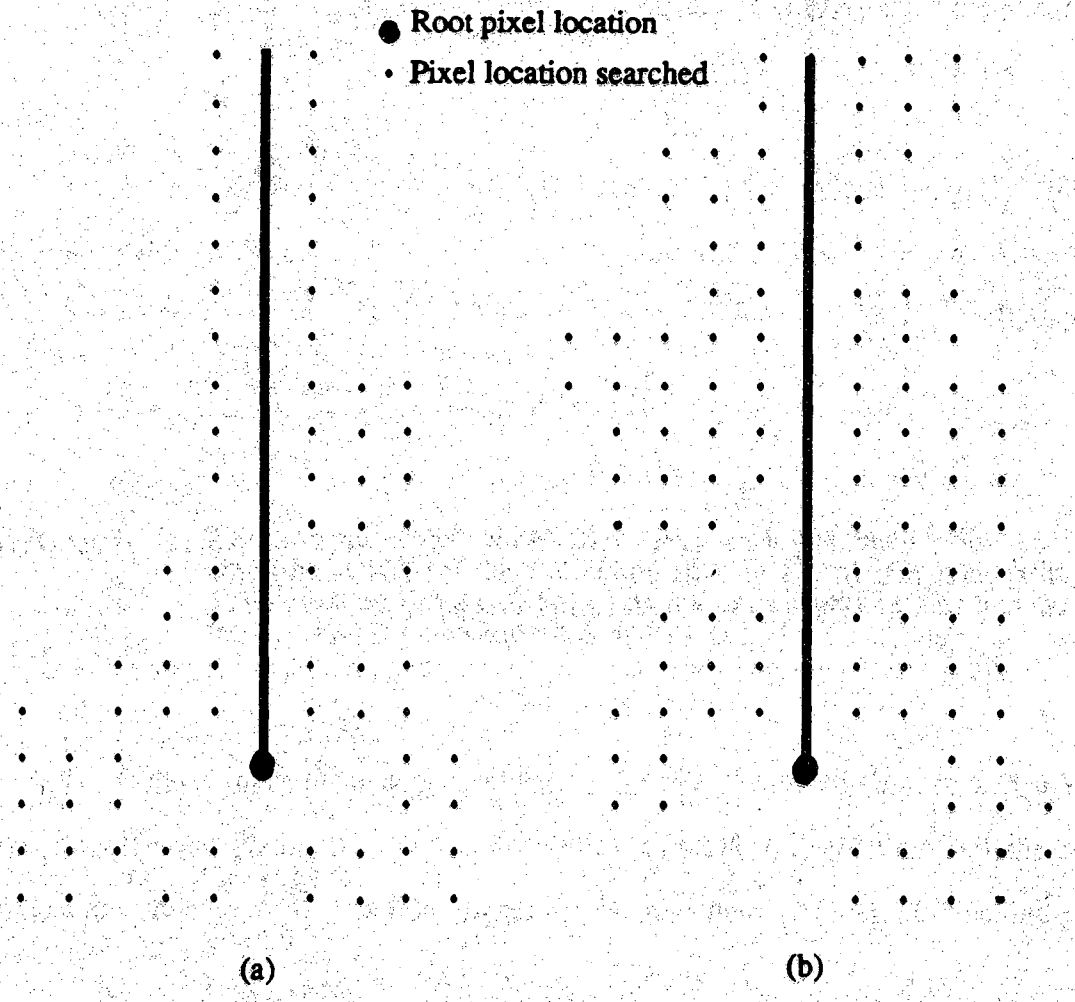


Figure 7.7: An example which shows the pixel locations visited by the SEL algorithm when (a) the simulation density $f^*_Y(.;.)$ is a non-stationary density which decays to $f_Y(.;.)$ as the search gets farther and farther from $\underline{i}_0$; and (b) the simulation density $f^*_Y(.;.)$ is stationary.

To further understand this problem, consider the following *binary image* example. That is, the digital image under consideration is an image for which the pixel values are either 0 or 1. For this image, the random field conditional densities $p_0(.)$

and $p_1(.)$ are defined as follows: For some a_0 and $a_1 \in [0, 1]$,

$$p_0(y_{ij}) \triangleq \begin{cases} a_0 & \text{if } y_{ij} = 1 \\ (1 - a_0) & \text{if } y_{ij} = 0 \end{cases} \tag{7.19}$$

and

$$p_1(y_{ij}) \triangleq \begin{cases} a_1 & \text{if } y_{ij} = 1 \\ (1 - a_1) & \text{if } y_{ij} = 0 \end{cases} \tag{7.20}$$

Now, let d_{ij} denotes the distance between $\underline{i}_0$ and some pixel location (i,j) on the image. Next, for some b_0 and $b_1 \in [0, 1]$ and for some $\alpha > 0$, define

$$p_0^*(y_{ij}) \triangleq p_0(y_{ij}) \left(1 - e^{-\alpha d_{ij}} \right) + b_0 \, e^{-\alpha d_{ij}} \tag{7.21}$$

and

$$p_1^*(y_{ij}) \triangleq p_1(y_{ij}) \left(1 - e^{-\alpha d_{ij}} \right) + b_1 \, e^{-\alpha d_{ij}}. \tag{7.22}$$

That is, the importance sampling densities $p_0^*(.)$ and $p_1^*(.)$ decay exponentially to $p_0(.)$ and $p_1(.)$ as $d_{ij} \uparrow \infty$.

To illustrate the differences between this non-stationary model and a stationary simulation model, consider the following simple stationary simulation model

$$p_0^*(y_{ij}) \triangleq \begin{cases} c_0 & \text{if } y_{ij} = 1 \\ (1 - c_0) & \text{if } y_{ij} = 0 \end{cases} \tag{7.23}$$

and

$$p_1^*(y_{ij}) \triangleq \begin{cases} c_1 & \text{if } y_{ij} = 1 \\ (1 - c_1) & \text{if } y_{ij} = 0 \end{cases} \tag{7.24}$$

where $a_0 < c_0 < 1$ and $0 < c_1 < a_1$. Next consider the problem of estimating the expected number of metric computations per correct decision; i.e., we are interested in estimating $E[C_\gamma]$. In this case, recall that the set of pixel indices which determine C_γ

is $\mathbf{D}^{(\emptyset)} \cap \tilde{\mathbf{S}}_\gamma$ where

$$\tilde{\mathbf{S}}_\gamma \triangleq \bigcup_{\delta \in D_\gamma - \beta_1} \mathbf{S}_\delta,$$

and β_1 is the direct descendent node of γ which lies on the terminal path. As a consequence, it follows that the stationary simulation model will most likely end up simulating unimportant events because this model makes errors occur everywhere and hence, it will extend a lot of incorrect subtrees which are not needed. In other words, if such model is used then the algorithm will be forced to waste computation time on the exploration of incorrect paths which do not determine C_γ. On the other hand, the non-stationary simulation model can be selected so that only the pixels of interest are searched most of the time. Specifically, the non-stationary simulation model can be chosen so that the image around $\underline{i}_0$ is noisier than the actual operating conditions. As consequence, the algorithm will be forced, most of the time, to search the image paths whose pixel locations are in $\mathbf{D}^{(\emptyset)} \cap \tilde{\mathbf{S}}_\gamma$. In other words, the non-stationary simulation model makes the "important events" occur very often without forcing the algorithm to waste a lot of computation time as in the case of the stationary simulation model. Figure 7.8 and Figure 7.9 show actual simulations of the SEL algorithm with the stationary simulation model specified by (7.23)-(7.24); as well as, the non-stationary model that as specified by (7.21)-(7.22).

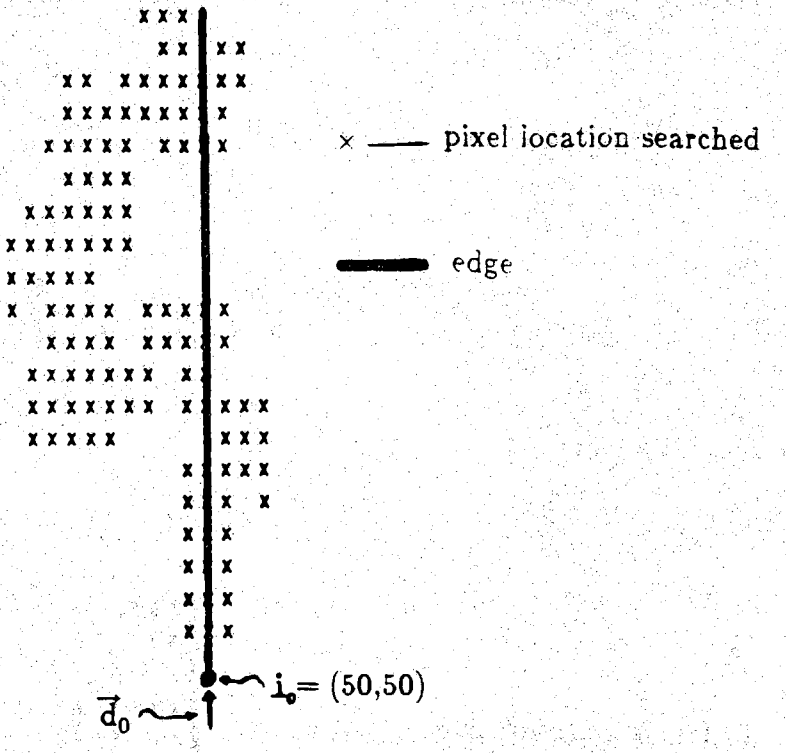


Figure 7.8: An illustration of an actual simulation of the SEL algorithm using a stationary simulation model for the binary image example.

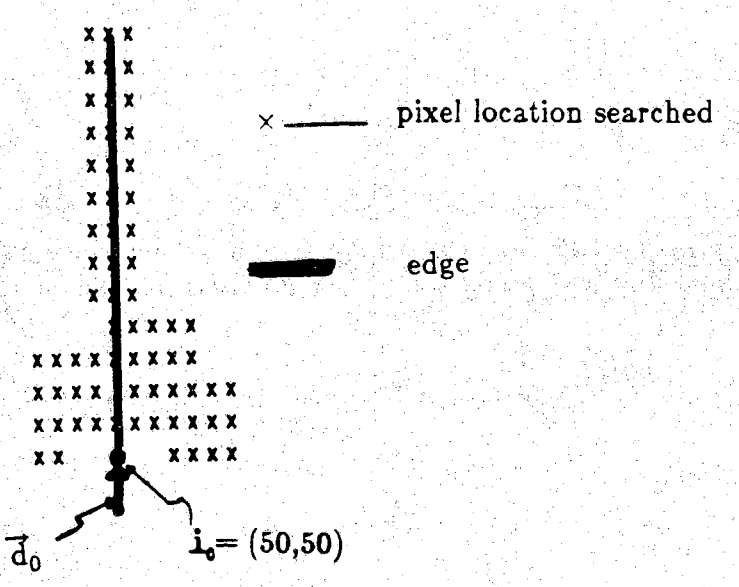


Figure 7.9: An illustration of an actual simulation of the SEL algorithm using a non-stationary simulation model for the binary image example.

To further illustrate the potential of applying importance sampling to the simulation of the SEL algorithm, consider the binary image example and suppose that

$$p_0(y_{ij}) = \begin{cases} .008 & \text{if } y_{ij} = 1 \\ .992 & \text{if } y_{ij} = 0 \end{cases} \tag{7.25}$$

and

$$p_1(y_{ij}) \triangleq \begin{cases} .99 & \text{if } y_{ij} = 1 \\ .01 & \text{if } y_{ij} = 0 \end{cases} \tag{7.26}$$

Next Suppose that we are interested in estimating the probability of error following the correct decision at node γ. That is, we would like to estimate

$$\alpha \triangleq E[\, I_E \mid \gamma \text{ is on the terminal path }] \tag{7.27}$$

where $E = \text{event } \{\beta_1 \neq \alpha_1\}$. In this example, we note that the correct path and the terminal path in S_γ are respectively denoted by $(\gamma, \alpha_1, \alpha_2, ...)$ and $(\gamma, \beta_1, \beta_2, ...)$.

Let $\hat{\alpha}^*$ be the importance sampling estimator for the above probability of error. Table 7.1 list some simulation results which were obtained using the non-stationary simulation model specified by (7.21) and (7.22) with $b_0 = .2$, $b_1 = .85$ and $\alpha = .5$.

Table 7.1: The probability of error estimate for the binary image example.

$\hat{\alpha}^*$	Frequency	Accuracy	reg
$.1596 \times 10^{-3}$	182517	.31 %	1298

We note that a total of $L = 500{,}000$ simulation runs were used to compute the above estimates. Furthermore, we note that the second column in the above table lists

the relative frequency of the error event E during the L simulations. The substantial increase of the error event relative frequency under the simulation model (in comparison to ordinary Monte Carlo) indicates one of the key characteristics of importance sampling. Finally, we note that the last two columns in the above table give respectively the estimates of the accuracy and the relative efficiency gain. These quantities were estimated using sample variances estimates as in the previous chapters. We omit the details for brevity.

In conclusion, it is clear that the idea of applying importance sampling to the simulation of the SEL algorithm is still in its infancy. It is also true that this preliminary work has clearly posed more questions than answers. However, it has certainly presented several challenges for future research. The problem of selecting suitable importance sampling densities in the context of sequential edge linking will obviously be the major emphasis in this work. The extremely good accuracy and high relative efficiency gain obtained by using the simple ad hoc non-stationary model in the simple binary image example indicate that there is indeed a great potential for applying the principles of importance sampling to the SEL algorithm simulation and that substantial efficiency increases in comparison to ordinary Monte Carlo simulations are possible. By applying the ideas discussed in this section, along with the leverage obtained from the importance sampling simulation of sequential decoders, we feel that it is possible to design efficient importance sampling models for simulating the SEL algorithm.

CHAPTER 8
CONCLUSIONS

This thesis has demonstrated that when properly implemented, importance sampling could prove to be an extremely powerful technique for improving the computational efficiency gains of conventional Monte Carlo simulations. The presentation in the first few chapters of this thesis has considered mainly the estimation of the distribution of computation of stack algorithm decoders. However, it is noted that the new simulation methods that we have presented here, can be also used to estimate other key parameters that characterize the performance of the stack algorithm. A typical example of such parameters is the average number of metric computations per correct decision. Another key quantity that characterize the stack algorithm performance is the bit error probability. In Chapter 6, we have shown that the error probabilities associated with stack algorithm decoders can be efficiently estimated using importance sampling. Finally in Chapter 7, recall that we have shifted our attention to the simulation of the SEL algorithm. The presentation in this chapter has mainly considered the development of a new importance sampling technique for simulating the SEL algorithm; as well as, the basic theory which is relevant to this application.

To demonstrate the power and accuracy of our new simulation techniques, we have presented in Chapter 5 and 6 numerical results for some convolutional codes that

are operating on the binary symmetric channel and the additive white gaussian noise channel. These simulation results indicate that very good accuracies, along with astronomical computational efficiency gains can be achieved when our simulation techniques are used. In Chapter 7, we have illustrated the potential of applying importance sampling to the simulation of the SEL algorithm by presenting some simulation results for a binary image. These results indicate that importance sampling holds the promise of offering substantial improvements in computational cost in this particular application.

The design of efficient and practical importance sampling simulation distributions is often ad hoc. Most of the important sampling analysis begins by proposing a family of candidate simulation distributions and then optimizing some parameters of the family. For example, in this thesis the reference path method was an ad hoc simulation technique obtained by an application of a specific knowledge about the stack algorithm decoder operation. The partitioning method on the other hand was inspired by a branch of probability theory known as large deviations theory. Because large deviations theory is not limited to ideal channels, the partitioning method promises to be a powerful tool for performance evaluation of sequential decoders for non-ideal channels.

There are many aspects of the work presented here that offer avenues for future research. Indeed, using the leverage obtained from the simulation techniques presented in this thesis, it is expected that efficient importance sampling can be developed for the performance evaluation and design verification of coded communication systems. In particular, the simulation of communication channels which are corrupted by one or more non-ideal characteristics such as intersymbol interference, nonlinearity, synchronization errors, etc, will provide a large class of

challenging and practical problems. As an example, consider the simulation of optical communication systems. For such systems, a typical receiver consists of a photodetector, an amplifier, and some signal processing circuitry [87-91]. It turns out that the noise at the output of an optical receiver can be modeled as a filtered doubly stochastic Poisson process [90]. As a consequence, the probability density function of an optical receiver output signal is exceedingly complex [90], [91]. This makes a general analytical performance analysis of optical communication systems very difficult, if not impossible [3], [91]. It is noted that in many previous work, the error probabilities of optical systems were derived assuming limiting hypothesis on the statistics of the receiver noise [87]. Nontheless, for performance evaluation of optical communication systems, computer simulation are often used because of the analytical intractability associated with such systems [3]. Consequently, importance sampling could prove to be an extremely powerful and appealing tool for performance evaluation of optical communication systems. This technique holds the promise of offering vast improvements in computational cost in comparison to the ordinary Monte Carlo method. We should note that the idea of applying importance to the simulation of optical communication systems is not new. Indeed, Balaban [3] has successfully developed an importance sampling scheme to evaluate the error rate for the fiberguide repeaters. It is noted, however, that the application of importance sampling to the simulation of optical communication systems is still in its infancy. Consequently, this is a major area for both practical and theoretical research.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Kahn, H. and A. W. Marshall, "Methods of reducing sample size in Monte Carlo computations," *Journal of the operations research society of america,* VOL. 1, pp. 263-278 (1953)

[2] J. M. Hammersley and D. C. Handscomb, "Monte Carlo methods," *New York: Chapman and Hall* (1964)

[3] P. Balaban, "Statistical evaluation of the error rate of the fiberguide repeater using importance sampling," *Bell System Technical Journal,* VOL. 55, No. 6, pp. 745-766 (1976)

[4] D. Siegmund, "Importance sampling in the Monte Carlo study of sequential tests," *Ann. Statis.,* VOL. 4, pp. 673-684 (1976)

[5] K. S. Shanmugan and P. Balaban, "A modified Monte Carlo simulation technique for the evaluation of error rate in digital communication systems," *IEEE Trans. on Communication,* VOL. COM-28, pp. 1916-1924 (1980)

[6] R. L. Mitchel, "Importance sampling applied to simulation of false alarm statistics," *IEEE Trans. Aerospace Electron. Syst.,* VOL. AES-17, pp. 15-24 (1981)

[7] R. Y. Rubinstein, "Simulation and the Monte Carlo method," *New York: Wiley,* (1981)

[8] G. W. Lank, "Theoretical aspects of importance sampling applied to false alarms," *IEEE Trans. Information Theory,* IT-29, pp. 73-82 (1983)

[9] M. Cottrell, J. C. Fort, and G. Malgouyres,, "Large deviations and rare events in the study of stochastic algorithms," *IEEE Trans. Automatic Control,* VOL. AC-28, pp. 907-920 (1983)

[10] M. C. Jeruchim, "Techniques for estimating the bit error rate in the simulation of digital communications systems," *IEEE Journal Select. Areas Communication,* VOL. SAC-2, pp. 153-170 (1984)

[11] M. C. Jeruchim, "On the application of importance sampling to the simulation of digital satellite and multihop links," *IEEE Trans. Communication,* VOL. COM-32 (1984)

[12] B. R. Davis, "An improved importance sampling method for digital communications system simulations," *IEEE Trans. Communication,* VOL. COM-34, pp. 715-719 (1986)

[13] G. Orsak and B. Aazhang, "On the application of importance sampling to the analysis of detection systems," *Proc. 25th Annual Allerton Conf. on Communication, Control, and Computing,* University of Illinois, Monticello, IL, pp. 135-144 (1987)

[14] P. W. Glynn and D. L. Iglehart, "Importance sampling for stochastic simulations," *Technical Report No. 49, Dept. of Operations Research,* Stanford University, Stanford, CA (1987)

[15] Ripley, "Stochastic simulations," *Willey, New York,* (1987)

[16] Wong, Q and V. Bhargava, "On the application of importance sampling to BER estimation in the simulation of digital communication systems," *IEEE Trans. Communication,* VOL. COM-35, pp. 1231-1233 (1987)

[17] K. K. Parhi and R. S. Berkowitz, "On optimizing importance sampling simulations," *IEEE Trans. Circuit & Systems, VOL. CAS-34, pp. 1558-1563 (1987)*

[18] P. Hahn and M. Jeruchim, "Developments in the theory and applications of importance sampling," *IEEE Trans. Communication,* VOL. COM-35, pp. 706-714 (1987)

[19] D. Lu and K. Yao, "Bounds on the variances of importance sampling simulations in digital communication systems," *Proc. 25th Annual Allerton Conf. on Communication, Control, and Computing,* University of Illinois, Monticello, IL, pp. 135-144 (1987)

[20] P. Bratley, B. L. Fox and L. E. Schrag, A Guide to Simulation. *New York: Springer,* (1987)

[21] D. Lu and K. Yao, "Improved importance sampling technique for efficient simulation of digital communications systems," *IEEE Journal Select. Areas Communication,* VOL. SAC-6, pp. 67-75 (1988)

[22] G. Orsak and B. Aazhang, "A comparison of two importance sampling methods for the analysis of detection systems," *Proc. 22nd Annual Conf. on Information Sciences & Systems,* Princeton University, Princeton, NJ, pp. 314-318 (1988)

[23] J. S. Sadowsky and J. A. Bucklew, "Importance sampling and Viterbi decoder simulation," *Proc. 22nd Annual Conf. on Information Sciences & Systems,* Princeton University, Princeton, NJ, pp. 319-324 (1988)

[24] J. S. Sadowsky "A new method for Viterbi decoder simulation using importance sampling," *To appear as a regular paper in IEEE Trans. on Communication*

[25] V. Hunkel and J. Bucklew, "Fast simulation for functionals of markov chains," *Proc. 22nd Annual Conf. on Information Science & Systems,* Princeton University, Princeton, NJ, pp. 330-335 (1988)

[26] J. S. Sadowsky and J. A. Bucklew, "On large deviations theory and asymptotically efficient Monte Carlo estimation," *To appear as a regular paper in IEEE Trans. on Information Theory,*

[27] S. Parekh and J. Walrand, "A quick simulation method for excessive backlogs in networks of queues," *IEEE Trans. Automatic Control,* VOL. AC-34, pp. 54-66 (1989)

[28] K. B. Letaief and J. S. Sadowsky, "Some new methods for simulating sequential decoders using importance sampling," *Proc. 27th Annual Allerton Conf. on Communication, Control, and Computing,* University of Illinois, Monticello, IL (1989)

[29] M. C. Jeruchim, P. M. Hahn, K. P. Smyntek and R. T. Ray, "An experimental investigation of conventional and efficient importance sampling," *IEEE Trans. Communication,* VOL. COM-37, pp. 578-587 (1989)

[30] K. B. Letaief and J. S. Sadowsky, "New importance sampling methods for estimating sequential decoders performance," *Submitted to IEEE Trans. on Information Theory* (1990)

[31] J. A. Bucklew, P. Ney, and J. S. Sadowsky, "Monte Carlo simulation and large deviations for uniformly recurrent markov chains," *To appear as a regular paper in Journ. Advanc. Appl. Probab.* (1990)

[32] P. Elias, "Coding for noisy channels," *IRE Conv. Rec.,* pt. 4, pp. 37-46 (1955)

[33] J.M. Wozencraft, "Sequential decoding for reliable communication," *IRE Natl. Conv. Rec.,* VOL. 5, pt. 2, pp. 11-25 (1957)

[34] B. Reiffen, "Sequential encoding and decoding for the discrete memoryless channel," *MIT Research Lab. of Electronics,* Tech. Rept. 374 (1960)

[35] J.M. Wozencraft and B. Reiffen. "Sequential decoding," *MIT Press.*, Cambridge, Mass. (1961)

[36] R.M Fano. "A hueristic discussion of probabilistic decoding," *IEEE Trans. Information Theory,* IT-9, pp. 64-74 (April 1963).

[37] K. Zigangirov. "Some sequential decoding procedures," *problemy Peredachi informatsii,* VOL. 2, pp. 13-25 (1966).

[38] J.E. Savage, "Sequential decoding-the computational problem," *Bell System Technical Journal,* VOL. 45, pp. 149-175 (Jan. 1966)

[39] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Information theory, IT-13, pp. 260-269 (1967)*

[40] I. M. Jacobs, "Sequential decoding for efficient communication from deep space," *IEEE Trans. Communication Technology,* VOL. COM-15, pp. 492-501 (1967)

[41] I.M. Jacobs and E. R. Berlekamp. "A lower bound to the distribution of Computation for sequential decoding," *IEEE Trans. Information Theory,* IT-13, pp. 167-174 (April 1967).

[42] P. Hart, N. Nilsson, and B. raphael. "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Systems Sci. Cybernet.,* VOL. SSC-4, pp. 100-107 (July 1968).

[43] R. G. Gallager, "Information theory and reliable communication," *MCGraw-Hill,* New York (1968)

[44] F. Jelinek. "A fast sequential decoding algorithm using a stack," *IBM Journ. Res. and Dev.,* VOL. 13, pp. 675-685 (Nov. 1969).

[45] F. Jelinek. "An upper bound on moments of sequential decoding effort," *IEEE Trans. Information Theory,* IT-15, pp. 140-149 (Jan. 1969).

[46] D. J. Costello, "Construction of convolutional codes for sequential decoding," *PhD Dissertation,* Dept. Elec. Eng., Univ. Notre Dame, Notre Dame, IN (1969)

[47] A. J. Viterbi, "Convolutional codes and their performance in communication systems," *IEEE Trans. Commun. Technology,* VOL. COM-19, pp. 751-772 (1971)

[48] J. L. Massey and D. J. Costello, "Nonsystematic convolutional codes for sequential decoding in space applications," *IEEE Trans. Commun. Technol.*, COM-19, pp. 806-813 (1971)

[49] J. W. Layland and W. A. Lushbaugh, "A flexible high-speed sequential decoder for deep space channels," *IEEE Trans. Commun. Technol.*, VOL. COM-19, No. 5 (1971)

[50] G. D. Forney and E. K. Bower, "A High speed sequential decoder: Prototype design and test," *IEEE Trans. Commun. Technol.*, VOL. COM-19, pp. 821-835 (1971)

[51] J. L. Massey. "Variable-length codes and the Fano metric," *IEEE Trans. Information Theory*, IT-18, pp. 196-198 (January 1972)

[52] G. D. Forney, Jr. "The Viterbi algorithm," *Proc. IEEE*, VOL. 61, pp. 268-278 (March 1973)

[53] J. M. Geist, "Search properties of some sequential decoding algorithms," *IEEE Trans. Information Theory*, IT-19, pp. 519-526 (July 1973)

[54] G. D. Forney, Jr. "Convolutional codes III: Sequential decoding," *Inf. Control*, VOL. 25, pp. 267-297 (July 1974).

[55] D. Haccoun and M. J. Ferguson. "Generalized stack algorithms for decoding convolutional codes," *IEEE Trans. Information Theory*, IT-21, pp. 638-651 (Nov. 1975).

[56] P. R. Chevillat and D. J. Costello, Jr. "A multiple stack algorithm for erasurefree decoding of convolutional codes," *IEEE Trans. Communication*, COM-25, pp. 1460-1470 (Dec. 1977).

[57] A. J. Viterbi and J. K. Omura. "Principles of digital communication and coding," *McGraw Hill, New York*, (1979).

[58] R. Johannesson, "On the distribution of computation for sequential decoding using the stack algorithm," *IEEE Trans. Information Theory*, VOL. IT-25, No. 3 (1979)

[59] S. Lin and D. J. Costello, "Error control coding: Fundamentals and applications," *Prentice-Hall*, Englewood Cliffs, New Jersey (1983)

[60] S. Mohan and J.B. Anderson, "Computationally optimal metric-first code tree search algorithms," *IEEE Trans. Communication*, VOL. COM-32, pp. 710-717 (June 1984)

[61] K. B. Letaief and J. S. Sadowsky "A Large deviations analysis of the Stack Algorithm" *Proc. 22nd Annual Conf. on Information Science & System,* Princeton university, Princeton, NJ, pp. 1027-1032 (March 1988)

[62] E. Wong, "Stochastic processes in information and dynamical systems," *Huntington, N.Y.,* Krieger (1971)

[63] E. Cinlar, "Introduction to stochastic processes," *Prentice-Hall, Englewood Cliffs, NJ,* (1975)

[64] I. Iscoe, P. Ney, and E. Nummelin. "Large deviations of uniformly recurrent Markov additive processes," *Adv. in Appl. Math.,* 6, pp. 373-412 (1985)

[65] P. Ney and E. Nummelin. "Some limit theorems for Markov additive processes," *in semi-Markov Models, ed. Janssen, Plenum,* (1986)

[66] P. Ney and E. Nummelin. "Markov additive processes I: eigenvalue properties and limit theorems, parts I and II," *Ann. Probab.,* 15, pp. 561-609 (1987)

[67] N. Nilson, "Problem-Solving Methods in Artificial Intelligence," *N.Y.: McGraw-Hill,* (1971)

[68] A. Martelli, "Edge detection using heuristic search method," *CGIP 1,* pp. 169-182 (1972)

[69] M. Hueckel, "A Local visual operator which recognizes edges and lines," *J. ACM,* VOL. 20, pp. 634-647 (1973)

[70] Y. Chien and K. Fu, "A decision function method for boundary detection," *CGIP, VOL. 3,* pp. 125-140 (1974)

[71] A. Rosenfeld and A. Kak, "Digital Picture Processing," N.Y.: Academic Press, (1976)

[72] A. Martelli, "An application of heuristic search methods to edge and contour detection," *Commun. ACM,* VOL. 19, pp. 73-83 (1976)

[73] W. Pratt, "Digital image processing," *N.Y.: Wiley,* (1978)

[74] G. Ashkar and J. Modestino, "The contour extraction problem with biomedical applications," *CGIP, VOL. 7,* pp. 331-355 (1978)

[75] I. Addou, "Quantitative methods of edge detection," *Los Angeles, CA: Image Processing Inst.*, Univ. Southern California (1978)

[76] I. Addou and W. Pratt, "Quantitative design and evaluation of enhancement/thresholding edge detectors," *Proc. of IEEE,* VOL. 67, pp. 753-763 (1979)

[77] D. Cooper, "Maximum likelihood estimation of Markov-process boundaries in noisy images," *IEEE Trans. Pattern Anal. Machine Intell.,* VOL. PAMI-1, pp. 372-384 (1979)

[78] D. Marr and E. Hildreth, "Theory of edge detection," *Proc. R. Soc. Lond.,* VOL. B 207, pp. 187-217 (1980)

[79] R. Machuca and A. Gilbert, "Finding edges in noisy scenes," *IEEE Trans. Pattern Anal. Machine Intell.,* VOL. PAMI-3, pp. 103-111 (1981)

[80] M. Basseville, B. Espiau, and J. Gasnier, "Edge detection using sequential methods for change in level - part I: a sequential edge detection algorithm," *IEEE Trans. Acoustics, Speech & Sig. Proc.,* ASSP-29, pp. 24-31, (1981)

[81] R. Suciu and A. Reeves, "A comparison of differential and moment based edge detectors," *Proc. IEEE Comp. Soc. Conf. on Pattern Recog. and Image Proc.,* pp. 97-102 (1982)

[82] M. Kunt, "Edge detection: a tutorial review," *Proc. IEEE Intern. Conf. on Acoustics, Speech and Signal Proc.,* VOL. 2 (1982)

[83] J. Canny, "Finding edges and lines in images," *MIT AI-TR-720,* (1983)

[84] P. H. Eichel, "Sequential detection of linear features in two-dimensional random fields," *Ph.D dissertation, Computer, Information, and Control Engineering Program,* University of Michigan, Ann Arbor, MI 48109 (May 1985)

[85] P. H. Eichel and E.J. Delp, "Sequential edge detection in correlated random fields," *Proc. IEEE Computer Vision and Pattern Recognition Conf.,* San Francisco, CA, pp. 14-21 (June 1985)

[86] P. H. Eichel, E. J. Delp, K. Koral, and J. Buda, "A method for a fully automatic definition of coronary arterial edges from cineangiograms," *IEEE Trans. on Medical Imaging,* VOL. 7, NO. 4, pp. 313-320 (December 1988)

[87] S. D. Personick, "Receiver design for digital fiber optical communication systems, I," *Bell System Technical Journal,* VOL. 52, pp. 843-874 (1973)

[88] W. Hauk, F. Bross, and M. Ottka, "The calculation of error rates for optical fiber systems," *IEEE Trans. Communication,* VOL. COM-26, No. 7 (1978)

[89] N. Sorenson and R. Gagliardi, "Performance of optical receivers with avalanche photodetection," *IEEE Trans. Communication,* VOL. COM-27, No. 9 (1979)

[90] R. Dogliotti, A. Luvison, and G. Pirani, "Error probability in optical fiber transmission systems," *IEEE Trans. Information Theory,* VOL. IT-25, No. 2 (1979)

[91] R. Gagliardi and G. Prati, "On Gaussian error probabilities in optical receivers," *IEEE Trans. Communication,* VOL. COM-28, No. 9 (1980)

VITA

VITA

Khaled Ben Letaief was born in Nabeul, Tunisia on January 7, 1962. He attended Ecole Reussite Primary School from 1968 to 1974 and Lycee Technique High School from 1974 to 1981 in Nabeul, Tunisia. He received the Baccalaureat degree with distinction from Lycee Technique in June 1981. He received his BS with distinction in Electrical Engineering from Purdue University, West Lafayette, Indiana, USA, in December 1984. He has also received his MS and Ph.D degrees in Electrical Engineering from Purdue University, in August 1986, and May 1990, respectively. Since January, 1985 he has been employed as a teaching assistant and a research assistant in the School of Electrical Engineering at Purdue University.

His research interests include statistical communications, information & coding theory, digital communications, digital signal and image processing.

Khaled is a member of the Tunisian Scientific Society and IEEE.