

# TRELLIS AND TURBO CODING

by

Christian Schlegel and Lance Pérez  
IEEE Press, 2002



# Contents

<b>7</b>	<b>Decoding Strategies</b>	<b>1</b>
7.1	Background and Introduction . . . . .	1
7.2	Tree Decoders . . . . .	3
7.3	The Stack Algorithm . . . . .	6
7.4	The Fano Algorithm . . . . .	7
7.5	The $M$ -Algorithm . . . . .	9
7.6	Maximum Likelihood Decoding . . . . .	21
7.7	A Posteriori Probability Symbol Decoding . . . . .	25
7.8	Log-APP, Max-Log-APP, and Approximations . . . . .	32
	7.8.1 The APP in the Logarithm Domain (Log-APP) . . . . .	32
	7.8.2 Max-Log-APP . . . . .	34
	7.8.3 Approximations . . . . .	36
7.9	Random Coding Analysis of Sequential Decoding . . . . .	37
7.10	Some Final Remarks . . . . .	43



## Chapter 7

# Decoding Strategies

### 7.1 Background and Introduction

There a great variety of decoding algorithms for trellis, some heuristic, and some derived from well-defined optimality criteria. Until very recently, the main objective of a decoding algorithm was the successful identification of the transmitted symbol sequence, accomplished by so called sequence decoders. These sequence decoders fall into two main groups: The tree decoders and the trellis decoders. Tree decoders explore the code tree, to be defined below, and their most well-known representatives are the sequential algorithms and limited-size breadth first algorithms, such as the  $M$ -algorithm. Trellis decoders make use of the more structured trellis of a code, and its main algorithms is the maximum-likelihood Viterbi algorithm.

Recently, and in conjunction with the emergence of Turbo coding, symbol probability algorithms have become prominent. They calculate the reliability of individual transmitted or information symbols, rather than decoding sequences. Symbol probability algorithms are essential for the iterative algorithms used to decode large concatenated codes, such as Turbo codes, and their importance will eclipse that of sequence decoders. Their most popular and widely used representative is the A posteriori Probability (APP) algorithm, also known as the BCJR algorithm, or the forward-backward algorithm. The APP algorithm works with the trellis of the code, and is discussed in detail in Section ??.

Let us now set the stage for the discussion of these decoding algorithms. In Chapters 2 and 3 we have discussed how a trellis encoder generates a sequence  $\underline{x}^{(i)} = (x_{-l}^{(i)}, \dots, x_l^{(i)})$  of correlated complex symbols  $x_r^{(i)}$  for message  $i$ , and how this sequence is modulated, using the pulse waveform  $p(t)$ , into

the (baseband) output signal

$$s^{(i)}(t) = \sum_{r=-l}^l x_r^{(i)} p(t - rT). \quad (7.1)$$

From Chapter 2 we also know the structure of the optimal decoder for such a system. We have to build a matched filter for each possible signal  $s^{(i)}(t)$  and select the message which corresponds to the signal which produces the largest sampled output value.

The matched filter for  $s^{(i)}(t)$  is given by

$$s^{(i)}(-t) = \sum_{r=-l}^l x_r^{(i)} p(-t - rT), \quad (7.2)$$

and, if  $r(t)$  is the received signal, the sampled response of the matched filter (7.2) to  $r(t)$  is given by (see also (2.21))

$$\begin{aligned} \underline{r} \cdot \underline{s}^{(i)} &= \int_{-\infty}^{\infty} r(t) s^{(i)}(t) dt \\ &= \sum_{r=-l}^l x_r^{(i)} y_r = \underline{x}^{(i)} \cdot \underline{y}, \end{aligned} \quad (7.3)$$

where  $y_r = \int_{-\infty}^{\infty} r(\alpha) p(\alpha - rT) d\alpha$  is the output of the filter matched to the pulse  $p(t)$  sampled at time  $t = rT$  as discussed in Section 2.5 (equation (2.24)), and  $\underline{y} = (y_{-l}, \dots, y_l)$  is the vector of sampled signals  $y_r$ .

If time-orthogonal pulses (e.g., Nyquist pulses)  $p(t)$  with unit energy ( $\int_{-\infty}^{\infty} p^2(t) dt = 1$ ) are used, the energy of the signal  $s^{(i)}(t)$  is given by

$$\begin{aligned} |\underline{s}^{(i)}|^2 &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} s^{(i)}(\alpha) s^{(i)}(\beta) d\alpha d\beta \\ &= \sum_{r=-l}^l |x_r^{(i)}|^2, \end{aligned} \quad (7.4)$$

and, from (2.21), the maximum likelihood receiver will select the sequence  $\underline{x}^{(i)}$  which maximizes

$$J^{(i)} = 2 \sum_{r=-l}^l \operatorname{Re} \left\{ x_r^{(i)} v_r^* \right\} - \sum_{r=-l}^l |x_r^{(i)}|^2. \quad (7.5)$$

$J^{(i)}$  in equation (7.5) is called the *metric* of the sequence  $\underline{x}^{(i)}$ , and this metric is to be maximized over all allowable choices of  $\underline{x}^{(i)}$ .

## 7.2 Tree Decoders

From (7.5) we define the partial metric at time  $n$  as

$$J_n^{(i)} = 2 \sum_{r=-l}^n \operatorname{Re} \left\{ x_r^{(i)} v_r^* \right\} - \sum_{r=-l}^n |x_r^{(i)}|^2, \quad (7.6)$$

which allows us to rewrite (7.5) in the recursive form

$$J_n^{(i)} = J_{n-1}^{(i)} + 2\operatorname{Re} \left\{ x_n^{(i)} v_n^* \right\} - |x_n^{(i)}|^2. \quad (7.7)$$

Equation (7.7) implies a tree structure which can be used to evaluate the metrics for all the allowable signal sequences as illustrated in Figure 7.1 below for the trellis code from Figure 3.1, Chapter 3. This tree has in general,  $2^k$  branches leaving each node, since there are  $2^k$  possible different choices of the signal  $x_n$  at time  $n$ . Each node is labeled with the hypothesized partial sequence<sup>1</sup>  $\tilde{\underline{x}}^{(i)} = (x_{-l}^{(i)}, \dots, x_n^{(i)})$  which leads to it. The intermediate metric  $J_n^{(i)}$  is also associated with each node. A tree decoder starts at time  $n = -l$  at the single root node with  $J_{-l} = 0$ , and extends through the tree evaluating and storing (7.7) until time unit  $n = l$ , at which time the largest accumulated metric identifies the most likely sequence  $\underline{x}^{(i)}$ .

It becomes obvious that the size of this tree grows very quickly. In fact, its final width is  $k^{2l+1}$ , which is an outlandish number even for small values of  $l$ , *i.e.*, short encoded sequences. We therefore need to reduce the complexity of decoding in some appropriate way, and this can be done by performing only a partial search of the tree.

There are a number of different approaches to tree decoding and we will discuss the more fundamental types in the subsequent sections. Before we tackle these decoding algorithms, however, we wish to modify the metric such that it can take into account the different lengths of paths, since we will come up against the problem of comparing paths of different lengths.

Consider then the set  $\mathcal{X}_M$  of  $M$  partial sequences  $\tilde{\underline{x}}^{(i)}$  with lengths  $\{n_i\}$ , and let  $n_{\max} = \max\{n_1, \dots, n_M\}$  be the maximum length among the  $M$  partial sequences. The decoder must make its likelihood ranking of the paths based on the partial received sequence  $\tilde{\underline{y}}$  of length  $n_{\max}$ .

---

<sup>1</sup>We denote partial sequences by tildes to distinguish them from complete sequences or codewords.

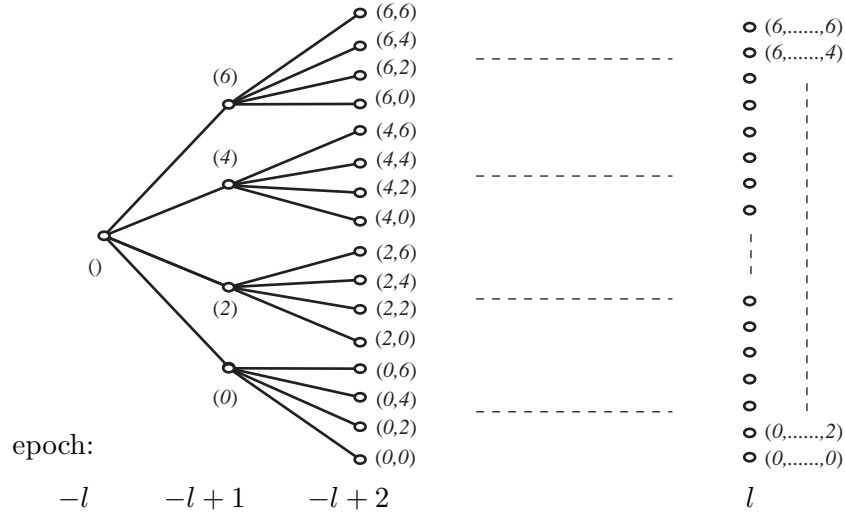


Figure 7.1: Code tree extending from time  $-l$  to time  $l$  for the code from Figure 3.1, Chapter 3.

From (2.10) we know that an optimum receiver would choose the  $\tilde{\underline{x}}^{(i)}$  which maximizes

$$P[\tilde{\underline{x}}^{(i)}|\tilde{\underline{y}}] = P[\tilde{\underline{x}}^{(i)}] \frac{\prod_{r=-l}^{-l+n_i+1} p_n(y_r - x_r) \prod_{r=-l+n_i}^{-l+n_{\max}} p(y_r)}{p(\tilde{\underline{y}})}, \quad (7.8)$$

where the second product reflects the fact that we have no hypotheses  $x_r^{(i)}$  for  $r > n_i$ , since  $\tilde{\underline{x}}^{(i)}$  extends only up to  $-l + n_i$ . We therefore have to use the a priori probabilities  $p(y_r|x_r^{(i)}) = p(y_r)$  for  $r > n_i$ . Using  $p(\tilde{\underline{y}}) = \prod_{r=-l}^{-l+n_{\max}} p(y_r)$  equation (7.8) can be rewritten as

$$P[\tilde{\underline{x}}^{(i)}|\tilde{\underline{y}}] = P[\tilde{\underline{x}}^{(i)}] \prod_{r=-l}^{-l+n_i} \frac{p_n(y_r - x_r^{(i)})}{p(y_r)}, \quad (7.9)$$

and we see that we need not be concerned with the tail samples not affected by  $\tilde{\underline{x}}_i$ . Taking logarithms gives the “additive” metric

$$L(\tilde{\underline{x}}^{(i)}, \tilde{\underline{y}}) = \sum_{r=-l}^{-l+n_i} \log \frac{p_n(y_r - x_r^{(i)})}{p(y_r)} - \log \frac{1}{P[\tilde{\underline{x}}^{(i)}]}. \quad (7.10)$$



Since  $P[\underline{\tilde{x}}^{(i)}] = (2^{-k})^{n_i}$  is the a priori the probability of the partial sequence  $\underline{\tilde{x}}^{(i)}$ , assuming that all the inputs to the trellis encoder have equal probability, (7.10) becomes

$$L(\underline{\tilde{x}}^{(i)}, \underline{\tilde{y}}) = L(\underline{\tilde{x}}^{(i)}, \underline{y}) = \sum_{r=-l}^{-l+n_i} \left[ \log \frac{p_n(y_r - x_r^{(i)})}{p(y_r)} - k \right], \quad (7.11)$$

where we have extended  $\underline{\tilde{y}} \rightarrow \underline{y}$  since (7.11) ignores the tail samples  $y_r; r > n_i$  anyhow. The metric (7.11) was introduced for decoding tree codes by Fano [15] in 1963, and was analytically derived by Massey [29] in 1972.

Since equation (2.11) explicitly gives the conditional probability distribution  $p_n(y_r - x_r)$ , the metric in (7.11) can be specialized for additive white Gaussian noise channels to

$$\begin{aligned} L(\underline{\tilde{x}}^{(i)}, \underline{y}) &= \sum_{r=-l}^{-l+n_i} \left[ \log \frac{\exp(-|x_r^{(i)} - y_r|^2/N_0)}{\sum_{x \in \mathcal{A}} p(x) \exp(-|x - y_r|^2/N_0)} - k \right] \\ &= - \sum_{r=-l}^{-l+n_i} \frac{|x_r^{(i)} - y_r|^2}{N_0} - c_r(y_r), \end{aligned} \quad (7.12)$$

where  $c_r(y_r) = \log(\sum_{x \in \mathcal{A}} p(x) \exp(-|x - y_r|^2/N_0)) + k$  is a term independent of  $x_r^{(i)}$  which is subtracted from all the metrics at time  $r$ . Note that  $c_r$  can be positive or negative, which causes some of the problems with *sequential decoding*, as we will see later.

It is worth noting here that if the paths examined are of the same length, say  $n$ , they all contain the same cumulative constant  $-\sum_{r=-l}^{-l+n} c_r$  in their metrics, which therefore may be discarded from all the metrics. This allows us to simplify (7.12) to

$$L(\underline{\tilde{x}}^{(i)}, \underline{y}) \equiv \sum_{r=-l}^{-l+n} 2\text{Re} \left\{ x_r^{(i)} v_r^* \right\} - |x_r^{(i)}|^2 = J_n^{(i)}, \quad (7.13)$$

by neglecting terms common to all the metrics. The metric (7.13) is equivalent to the accumulated Euclidean distance between the the received partial sequence  $\underline{\tilde{y}}$  and the hypothesized symbols on the  $i$ -th path to up to length  $n$ . The restriction to path of equal length makes this metric much simpler than the general metric (7.11) (and (7.12)) and finds application in the so called *breadth-first* decoding algorithms which we will discuss in subsequent sections.

### 7.3 The Stack Algorithm

The stack algorithm is one of the many variants of what has become known as *sequential decoding* of trellis codes. Sequential decoding was introduced by Wozencraft [40] for convolutional codes and has subsequently experienced many changes and additions. Sequential decoding describes any algorithm for decoding trellis codes which successively explores the code tree by moving to new nodes from an already explored node.

From the introductory discussion in the preceding section, one way of sequential decoding becomes apparent. We start exploring the tree and store the metric (7.11) (or (7.12)) for every node explored. At each stage now we simply extend the node with the largest such metric. This, in essence, is the *stack algorithm* first proposed by Zigangirov [45] and Jelinek [25]. This basic algorithm is:

- Step 1:** Initialize an empty stack  $S$  of visited nodes and their metrics. Deposit the empty partial sequence  $()$  at the top of the stack with its metric  $L((), \underline{y}) = 0$ .
- Step 2:** Extend the node corresponding to the top entry  $\{\tilde{\underline{x}}_{\text{top}}, L(\tilde{\underline{x}}_{\text{top}}, \underline{y})\}$  by forming  $L(\tilde{\underline{x}}_{\text{top}}, \underline{y}) - |x_r - y_r|^2/N_0 - c_r$  for all  $2^k$  extensions of  $\tilde{\underline{x}}_{\text{top}} \rightarrow (\tilde{\underline{x}}_{\text{top}}, x_r) = \tilde{\underline{x}}^{(i)}$ . Delete  $\{\tilde{\underline{x}}_{\text{top}}, L(\tilde{\underline{x}}_{\text{top}}, \underline{y})\}$  from the stack.
- Step 3:** Place the new entries  $\{\tilde{\underline{x}}^{(i)}, L(\tilde{\underline{x}}^{(i)}, \underline{y})\}$  from Step 2 into the stack such that the stack remains ordered with the entry with the largest metric at the top of the stack.
- Step 4:** If the top entry of the stack is a path to one of the terminal nodes at depth  $l$ , stop and select  $\underline{x}_{\text{top}}$  as the transmitted symbol sequence. Otherwise, go to Step 2.

There are some practical problems associated with the stack algorithm. Firstly, the number of computations which the algorithm performs is very dependent on the quality of the channel. If we have a very noisy channel, the received sample value  $y_r$  will be very unreliable and a large number of possible paths will have similar metrics. These paths all have to be stored in the stack and explored further. This causes a computational speed problem, since the incoming symbols have to be stored in a buffer while the algorithm performs the decoding operation. This buffer is now likely to overflow if the channel is very noisy and the decoder will have to declare a decoding failure. This phenomenon is explored further in Section 6.6. In practice,

the transmitted data will be framed and the decoder will declare a frame erasure if it experiences input buffer overflow.

A second problem with the stack algorithm is the increasing complexity of Step 2, *i.e.*, of reordering the stack. This sorting operation depends on the size of the stack, which, again, for very noisy channels becomes large. This problem is addressed in all practical applications by ignoring small differences in the metric and collecting all stack entries with metrics within a specified “quantization interval” in the same bucket. Bucket  $j$  contains all stack entries with metrics

$$j\Delta \leq L(\tilde{\mathbf{x}}^{(i)}, \mathbf{r}) \leq (j+1)\Delta, \quad (7.14)$$

where  $\Delta$  is a variable quantization parameter. Incoming paths are now sorted only into the correct bucket, avoiding the sorting complexity of the large stack. The depositing and removal of the paths from the buckets can occur on a “last in, first out” basis.

There are a number of variations of this basic theme. If  $\Delta$  is a fixed value, the number of buckets can also grow to be large, and the sorting problem, originally avoided, reappears. An alternative is to let the buckets vary in size, rather than in metric range. In that way, the critical dependence on the stack size can be avoided.

An associated problem with the stack is that of stack overflow. This is less severe and the remedy is simply to drop the last path in the stack from future consideration. The probability of actually losing the correct path is very small, a much smaller problem than that of a frame erasure. A large number of variants of this algorithm are feasible and have been explored in the literature. Further discussion of the details of implementation of these algorithms are found in [20, 2, 3].

## 7.4 The Fano Algorithm

Unlike the stack algorithm, the Fano algorithm is a depth-first tree search procedure in its purest form. Introduced by Fano [15] in 1963, this algorithm stores only one path, and thus, essentially, requires no storage. Its drawback is a certain loss in speed compared to the stack algorithm for higher rates [21], but for moderate rates the Fano algorithm decodes faster than the stack algorithm [22]. It seems that the Fano algorithm is the preferred choice for practical implementations of sequential decoding algorithms.

Since the Fano algorithm only stores one path, it must allow for backtracking. Also, there can be no jumping between non-connected nodes, *i.e.*,

the algorithm only moves between adjacent nodes which are connected in the code tree. The algorithm starts at the initial node and moves in the tree by proceeding from one node to a successor node with a suitably large metric. If no such node can be found, the algorithm backtracks and looks for other branches leading off from previously visited nodes. The metrics of all these adjacent nodes can be computed by adding or subtracting the metric of the connecting branch and no costly storing of metrics is required. If a node is visited more than once, its metric is recomputed. This is part of the computation/storage tradeoff of sequential decoding.

The algorithm proceeds along a chosen path as long as the metric continues to increase. It does that by continually tightening a metric threshold to the current node metric as it visits nodes for the first time. If new nodes along the path have a metric smaller than the threshold, the algorithm backs up and looks at other node extensions. If no other extensions with a metric above the threshold can be found, the value of the threshold is decreased and the forward search is resumed. In this fashion each node visited in the forward direction more than once is reached with a progressively lower threshold each time. This prevents the algorithm from getting caught in an infinite loop. Eventually this procedure reaches a terminal node at the end of the tree and a decoded symbol sequence can be output.

Figure 7.2 depicts an example of the search behavior of the Fano algorithm. Assume that there are two competing paths, where the solid path is the most likely sequence and the dashed path is a competitor. The vertical height of the nodes in Figure 7.2 is used to illustrate the values of the metrics for each node. Also assume that the paths shown are those with the best metrics, *i.e.*, all other branches leading off from the nodes lead to nodes with smaller metrics. Initially, the algorithm will proceed to node A, at which time it will start to backtrack since the metric of node D is smaller than that of node A. After exploring alternatives and successively lowering the threshold to  $t_1$ , and then to  $t_2$ , it will reach node O again and proceed along the dashed path to node B and node C. Now it will start to backtrack again, lowering its threshold to  $t_3$  and then to  $t_4$ . It will now again explore the solid path beyond node D to node E, since the lower threshold will allow that. From there on the path metrics pick up again and the algorithm proceeds along the solid path. If the threshold decrement  $\Delta$  had been twice as large, the algorithm would have moved back to node O faster, but would also have been able to move beyond the metric dip at node F, and would have chosen the erroneous path.

It becomes obvious that at some point the metric threshold  $t$  will have to be lowered to the lowest metric value which the maximum likelihood

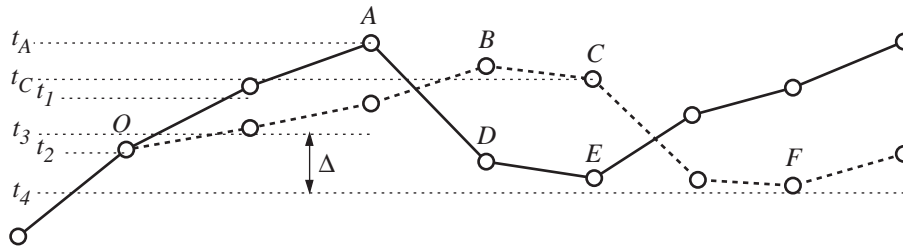


Figure 7.2: Illustration of the operation of the Fano algorithm when choosing between two competing paths.

path assumes, and, consequently, a large decrement  $\Delta$  allows the decoder to achieve this low threshold faster. Conversely, if the decrement  $\Delta$  is too large,  $t$  may drop to a value which allows several erroneous path to be potentially decoded before the maximum metric path. The optimal value of the metric threshold is best determined by experience and simulations. Figure 7.3 shows the flowchart of the Fano algorithm.

## 7.5 The $M$ -Algorithm

This section deals with a purely breadth-first algorithm. The  $M$ -algorithm is a synchronous algorithm which moves from time unit to time unit. It keeps  $M$  candidate paths at each iteration and deletes all others from further consideration. At each time unit the algorithm extends all  $M$  currently held nodes to form  $2^k M$  new nodes, from among which those  $M$  with the best metrics are retained. Due to the breadth-first nature of the algorithm, the metric in (7.13) can be used. The algorithm is very simple:

- Step 1:** Initialize an empty list  $L$  of candidate paths and their metrics. Deposit the zero-length path  $()$  with its metric  $L((), \underline{y}) = 0$  in the list. Set  $n = -l$ .
- Step 2:** Extend  $2^k$  partial paths  $\tilde{\underline{x}}^{(i)} \rightarrow (\tilde{\underline{x}}^{(i)}, x_r^{(i)})$  from each of the at most  $M$  paths  $\tilde{\underline{x}}^{(i)}$  in the list. Delete the entries in the original list.
- Step 3:** Find the at most  $M$  partial paths with the best metrics among the extensions<sup>2</sup> and save them in the list  $L$ . Delete the rest of the

<sup>2</sup>Note that from two or more extensions leading to the same state (see Section 6.5) all but the one with the best metric may be discarded. This will improve performance slightly by eliminating some paths which cannot be the ML path.

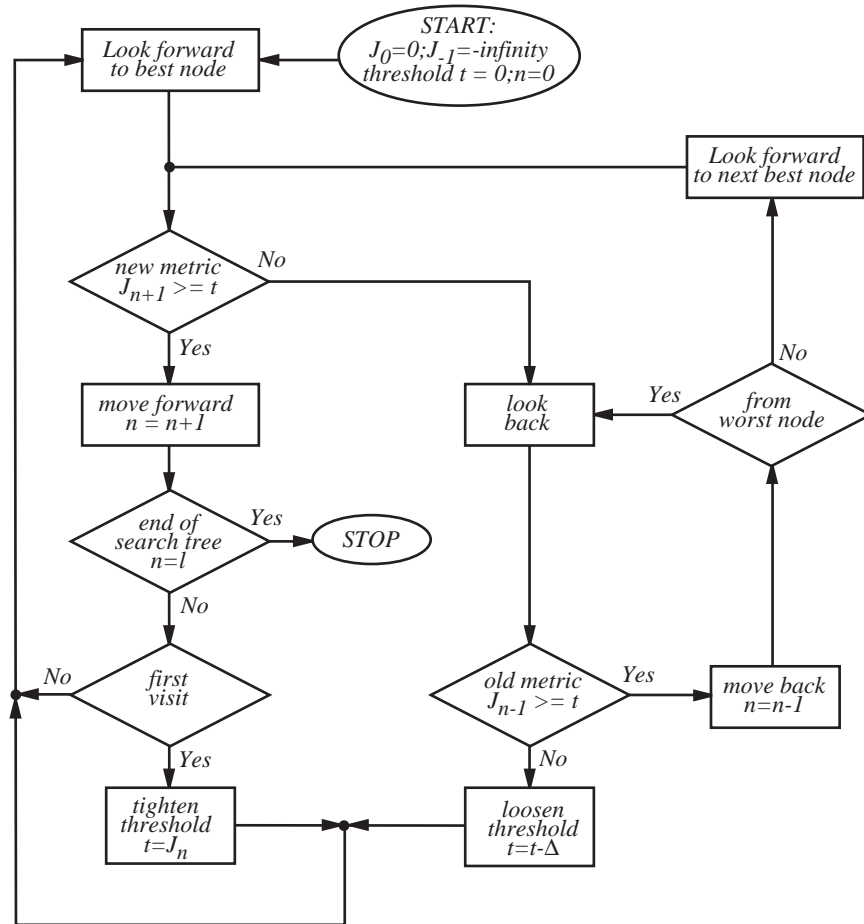


Figure 7.3: Flowchart of the Fano algorithm. The initialization of  $J_{-1} = -\infty$  has the effect that the algorithm can lower the threshold for the first step, if necessary.

extensions. Set  $n = n + 1$ .

**Step 4:** If at the end of the tree, *i.e.*,  $n = l$ , release the output symbols corresponding to the path with the best metric in the list  $L$ , otherwise go to Step 2.

The  $M$ -algorithm appeared in the literature for the first time in a paper by Jelinek Anderson [26], where it was applied to source coding. At the time, there were no real algorithms for sequential source coding other than this, so it was viewed as miraculous. In the early 1980s, applications of

the algorithm to channel coding began to appear. The research book by Anderson and Mohan on algorithmic source and channel coding [3] and reference [1] collect a lot of this material, and are the most comprehensive sources on the subject.

This algorithm is straightforward to implement and its popularity is partly due to the simple metric as compared to sequential decoding. The decoding problem with the  $M$ -Algorithm is the loss of the correct path from the list of candidates, after which the algorithm might spend a long time resynchronizing. This problem is usually addressed by framing the data. With each new frame resynchronization is achieved. The computational load of the  $M$ -algorithm is independent of the size of the code; it is proportional to  $M$ . Unlike depth-first algorithms, it is also independent of the quality of the channel, since  $M$  paths are retained irrespective of the channel quality.

A variant of the  $M$ -algorithm is the so-called  $T$ -algorithm. It differs from the  $M$ -algorithm only in Step 3, where instead of a fixed number  $M$ , all path with metrics  $L(\hat{\mathbf{x}}^{(i)}, \mathbf{y}) \geq \lambda_t - T$  are retained, where  $\lambda_t$  is the metric of the best path and  $T$  is some arbitrary threshold. The  $T$ -algorithm is therefore in a sense a hybrid between the  $M$ -algorithm and a stack-type algorithm. Its performance depends on  $T$ , but is very similar to that of the  $M$ -algorithm, and we will not discuss it further.

In Chapter 3 we have discussed that the performance of trellis codes using a maximum-likelihood detector was governed by the distance spectrum of the code, where the minimum free Euclidean distance  $d_{\text{free}}$  played a particularly important role. Since the  $M$ -algorithm is a suboptimum decoding algorithm, its performance is additionally affected by other criteria. The major criterion is the probability that the correct path is not among the  $M$  retained candidates at time  $n$ . If this happens, we lose the correct path and it usually takes a long time to resynchronize. We will see that the probability of correct path loss has no direct connection with the distance spectrum of a code.

Since the complexity of the  $M$ -algorithm is largely independent of the code size and constraint length, one usually chooses very long constraint-length codes to assure that  $d_{\text{free}}$  is appropriately large. If this is the case, the correct path loss becomes the dominant error event of the decoder.

Let us then take a closer look at the probability of losing the correct path at time  $n$ . To that end we assume that at time  $n - 1$  the correct path was among the  $M$  retained candidates as illustrated in Figure 7.4. Each of these  $M$  nodes is extended into  $2^k$  nodes at time  $n$ , of which  $M$  are to be retained. There are then a total of  $\binom{M2^k}{M}$  ways of choosing the new  $M$

retained paths at time  $n$ .

Let us denote the correct partial path by  $\tilde{\underline{x}}^{(c)}$ . The optimal strategy of the decoder will then be to retain that particular set of  $M$  candidates which maximizes the probability of containing  $\tilde{\underline{x}}^{(c)}$ . Let  $\mathcal{C}_p$  be one of the  $\binom{M2^k}{M}$  possible sets of  $M$  candidates at time  $n$ . We wish to maximize

$$\max_p \Pr \left\{ \tilde{\underline{x}}^{(c)} \in \mathcal{C}_p | \tilde{\underline{y}} \right\}. \quad (7.15)$$

Since all the partial paths  $\tilde{\underline{x}}^{(p_j)} \in \mathcal{C}_p, j = 1, \dots, M$  are distinct, the events  $\{\tilde{\underline{x}}^{(c)} = \tilde{\underline{x}}^{(p_j)}\}$  are all mutually exclusive for different  $j$ , *i.e.*, the correct path can be at most only one of the  $M$  different candidates  $\tilde{\underline{x}}^{(p_j)}$ . Equation (7.15) can therefore be evaluated as

$$\max_p \Pr \left\{ \tilde{\underline{x}}^{(c)} \in \mathcal{C}_p | \tilde{\underline{y}} \right\} = \max_p \sum_{j=1}^M \Pr \left\{ \tilde{\underline{x}}^{(c)} = \tilde{\underline{x}}^{(p_j)} | \tilde{\underline{y}} \right\}. \quad (7.16)$$

From (7.8), (7.10) and (7.13) we know that

$$\Pr \left\{ \tilde{\underline{x}}^{(c)} = \tilde{\underline{x}}^{(p_j)} | \tilde{\underline{y}} \right\} \propto \exp \left( - \sum_{r=-l}^{-l+n} \left( 2\text{Re} \left\{ x_r^{(p_j)} v_r^* \right\} - |x_r^{(p_j)}|^2 \right) \right), \quad (7.17)$$

where the proportionality constant is independent of  $\tilde{\underline{x}}^{(p_j)}$ . The maximization in (7.15) now becomes equivalent to (considering only the exponent from above)

$$\begin{aligned} \max_p \Pr \left\{ \tilde{\underline{x}}^{(c)} \in \mathcal{C}_p | \tilde{\underline{y}} \right\} &\equiv \max_p \sum_{j=1}^M \sum_{r=-l}^{-l+n} \left( 2\text{Re} \left\{ x_r^{(p_j)} v_r^* \right\} - |x_r^{(p_j)}|^2 \right) \\ &= \max_p J_n^{(p_j)}, \end{aligned} \quad (7.18)$$

*i.e.*, we simply collect the  $M$  paths with the best partial metrics  $J_n^{(p_j)}$  at time  $n$ . This argument was derived by Aulin [4]. Earlier we showed that the total metric can be broken up into the recursive form of (7.7), but now we have shown that if the detector is constrained to considering only a maximum of  $M$  path at each stage, retaining those  $M$  paths  $\tilde{\underline{x}}^{(p_j)}$  with maximum partial metrics is the optimal strategy.

The probability of correct path loss, denoted by  $\Pr(\text{CPL})$ , can now be addressed. Follow the methodology of Aulin [4], we need to evaluate the probability that the correct path  $\tilde{\underline{x}}^{(c)}$  is not among the  $M$  candidate paths.



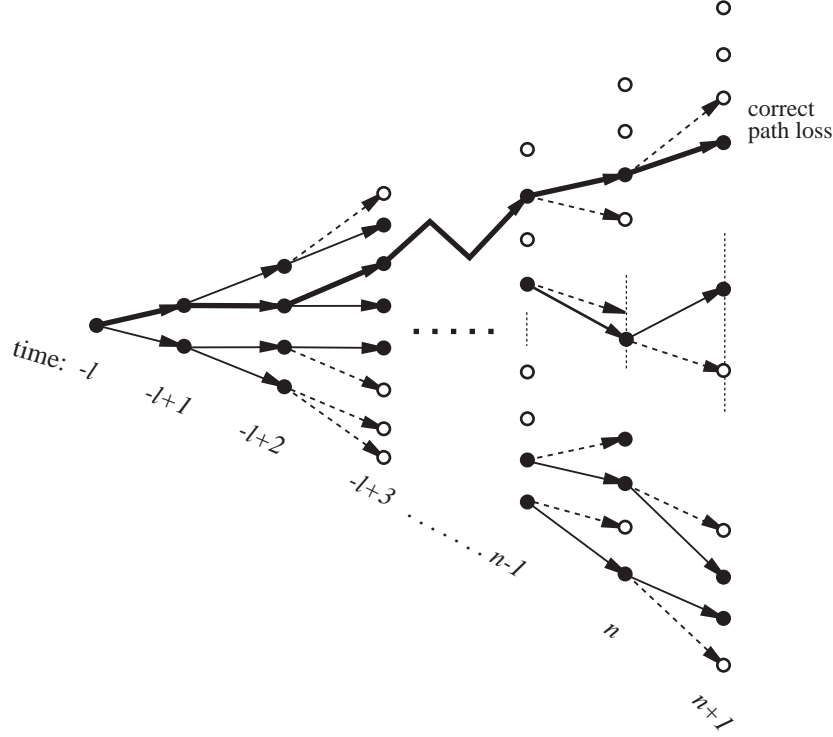


Figure 7.4: Extension of  $2^k M = 2 \cdot 4$  paths from the list of the best  $M = 4$  paths. The solid paths are those retained by the algorithm, the path indicated by the heavy line corresponds to the correct transmitted sequence.

This will happen if  $M$  paths  $\tilde{\mathbf{x}}^{(p_j)} \neq \tilde{\mathbf{x}}^{(c)}$  have a partial metric  $J_n^{(p_j)} \geq J_n^{(c)}$ , or equivalently if all the  $M$  metric differences

$$\delta_n^{(j,c)} = J_n^{(c)} - J_n^{(p_j)} = \sum_{r=-l}^{-l+n} \left( |x_r^{(p_j)}|^2 - |x_r^{(c)}|^2 - 2\text{Re} \left\{ x_r^{(p_j)} - x_r^{(c)} \right\} v_r^* \right) \quad (7.19)$$

are smaller than or equal to zero. That is,

$$\Pr(\text{CPL}|\mathcal{C}_p) = \Pr\{\underline{\delta}_n \leq \underline{0}\}; \quad \tilde{\mathbf{x}}^{(p_j)} \in \mathcal{C}_p, \quad (7.20)$$

where  $\underline{\delta}_n = \left( \delta_n^{(1,c)}, \dots, \delta_n^{(M,c)} \right)$  is the vector of metric differences at time  $n$  between the correct path and the set of paths in a given set  $\mathcal{C}_p$ , which does

not contain  $\tilde{x}^{(c)}$ .  $\Pr(\text{CPL}|\mathcal{C}_p)$  depends on the correct path  $\underline{x}^{(c)}$ , and, strictly speaking has to be averaged over all correct paths. We shall be satisfied with the correct path which produces the largest  $\Pr(\text{CPL}|\mathcal{C}_p)$ .

In Appendix 6.A we show that the probability of losing the correct path decreases exponentially with the signal-to-noise ratio, and is overbounded by

$$\Pr(\text{CPL}|\mathcal{C}_p) \leq Q\left(\sqrt{\frac{d_l^2}{2N_0}}\right). \quad (7.21)$$

The parameter  $d_l^2$  depends on  $\mathcal{C}_p$  and is known as the *Vector Euclidean distance* [4] of the path  $\tilde{x}^{(c)}$  with respect to the  $M$  error paths  $\tilde{x}^{(p_i)} \in \mathcal{C}_p$ . It is important to note here that (7.21) is an upper bound of the probability that  $M$  specific error paths have a metric larger than  $\tilde{x}^{(c)}$ . Finding  $d_l^2$  involves a combinatorial search (see Appendix 6.A).

Equation (7.21) demonstrates that the probability of correct path loss is an exponential error integral, and can thus be compared to the probability of the maximum likelihood decoder (equations (5.8) and (5.9)). The problem is finding the minimal  $d_l^2$  among all sets  $\mathcal{C}_p$ , denoted by  $\min(d_l^2)$ . This is a rather complicated combinatorial problem, since essentially all combinations of  $M$  candidates for each correct path at each time  $n$  have to be analyzed from the growing set of possibilities. Aulin [4] has studied this problem and gives several rejection rules which alleviate the complexity of finding  $d_l^2$ , but the problem remains complex and is in need of further study.

Note that  $d_l^2$  is a non-decreasing function of  $M$ , the decoder complexity, and one way of selecting  $M$  is to choose it such that

$$\min(d_l^2) \geq d_{\text{free}}^2. \quad (7.22)$$

This choice should guarantee that the performance of the  $M$ -algorithm is approximately equal to the performance of maximum-likelihood decoding. To see this, let  $\overline{P_e(M)}$  be the probability of an error event (compare equation (5.4)). Then

$$\begin{aligned} \overline{P_e(M)} &\leq \overline{P_e}(1 - \Pr(\text{CPL})) + \Pr(\text{CPL}) \\ &\leq \overline{P_e} + \Pr(\text{CPL}), \end{aligned} \quad (7.23)$$

where  $\overline{P_e}$  is of course the probability that a maximum-likelihood decoder starts an error event (Chapter 5). For high values of the signal-to-noise ratio, equation (7.23) can be approximated by

$$\overline{P_e(M)} \approx N_{d_{\text{free}}} Q\left(\frac{d_{\text{free}}}{\sqrt{2N_0}}\right) + \kappa Q\left(\frac{\min(d_l)}{\sqrt{2N_0}}\right), \quad (7.24)$$

where  $\kappa$  is some constant, which, however, is difficult to determine in the general case. Now, if (7.22) holds, the suboptimality does not exponentially dominate the error performance for high signal-to-noise ratios.

Aulin [4] has analyzed this situation for 8-PSK trellis codes and found that, in general,  $M \approx \sqrt{S}$  will fulfill condition (7.22), where  $S$  is the number of states in the code trellis.

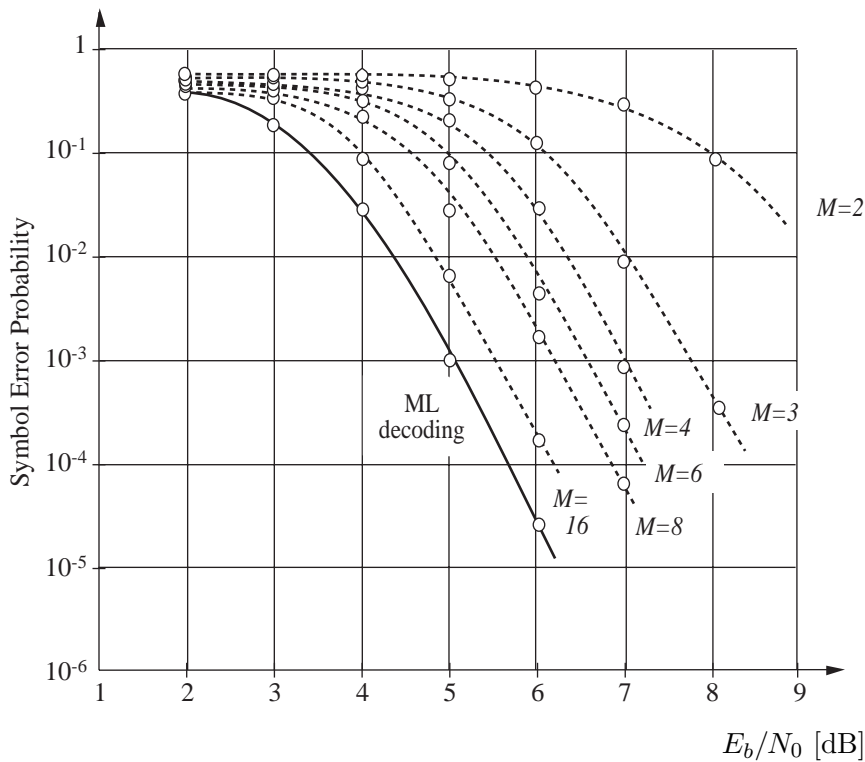


Figure 7.5: Simulation results for the 64-state optimal distance 8-PSK trellis codes decoded with the  $M$ -algorithm, using  $M = 2, 3, 4, 5, 6, 8$  and 16. The performance of maximum likelihood decoding is also included in the figure (Source [4]).

Figure 7.5 shows the simulated performance of the  $M$ -algorithm versus  $M$  for the 64-state trellis code from Table 3.1 with  $d_{\text{free}}^2 = 6.34$ .  $M = 8$  meets (7.22) according to [4], but from Figure 7.5 it is apparent that

the performance is still about 1.5dB poorer than ML-decoding. This is attributable to the resynchronization problems and the fact that we are operating at rather low values of the signal-to-noise ratio, where neither  $d_{\text{free}}^2$  nor  $\min(d_i^2)$  are necessarily dominating the error performance.

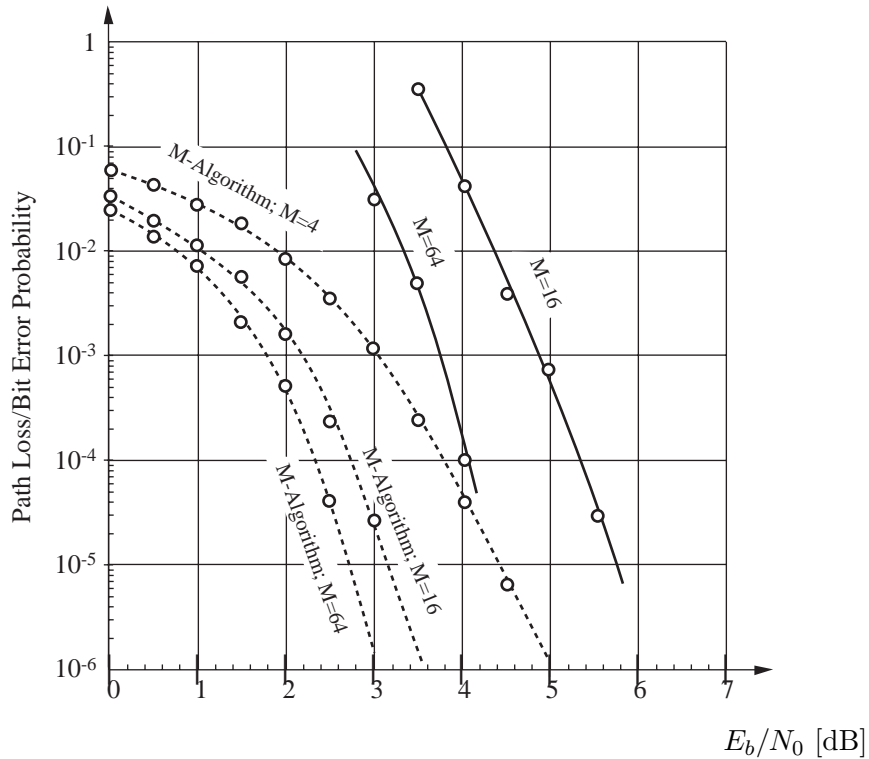


Figure 7.6: Simulation results for the 2048-state, rate  $R = 1/2$  convolutional code using the  $M$ -algorithm, for  $M = 4, 16$ , and  $M = 64$ . The dashed curves are the path loss probability and the solid curves are BER's.

Figures 7.6 and 7.7 show the empirical probability of correct path loss  $P(\text{CPL})$  and the BER for two convolutional codes and various values of  $M$ . Figure 7.6 shows simulation results for the 2048-state convolutional code,  $\nu = 11$ , from Table 4.1. The bit error rate and the probability of losing the correct path converge to the same asymptotic behavior, indicating that the probability of correct path loss and not recovery errors is the dominant error mechanism for very large values of the signal-to-noise ratio.

Figure 7.7 shows simulation results for the  $\nu = 15$  large constraint-length code for the same values of  $M$ . For this length code, path loss will be the dominant error scenario. We note that both codes have a very similar error performance, demonstrating that the code complexity has little influence.

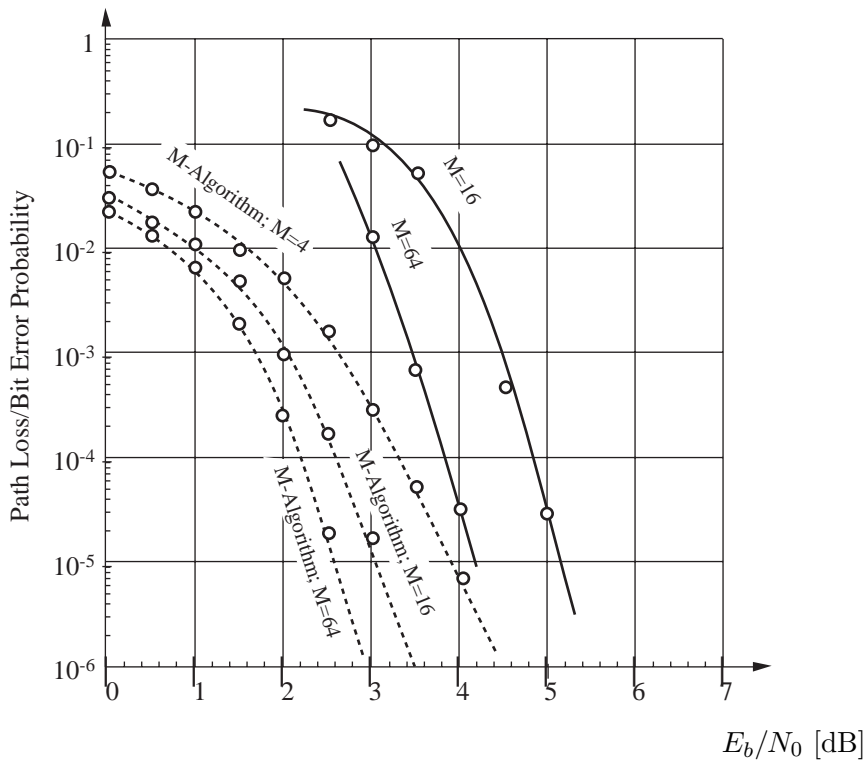


Figure 7.7: Same simulation results for the  $\nu = 15$ ,  $R = 1/2$  convolutional code.

Once the correct path is lost, the algorithm may spend a relatively long time before it finds it again, *i.e.*, before the correct path is again one of the  $M$  retained paths. Correct path recovery is a very complex problem and no complete analytical results have been found to date. There are only a few theoretical approaches to the recovery problem, such as [5]. This difficulty suggests that insight into the operation of the decoder during a recovery has to be gained through simulation studies.

Figure 7.8 shows the simulated average number of steps taken for the algorithm to recover the correct path. The simulations were done for the 2048-state,  $\nu = 11$  code, whose error performance is shown in Figure 7.6. Each instance of the simulation was performed such that the algorithm was initiated and run until the correct path was lost, then the number of steps until recovery were counted [27].

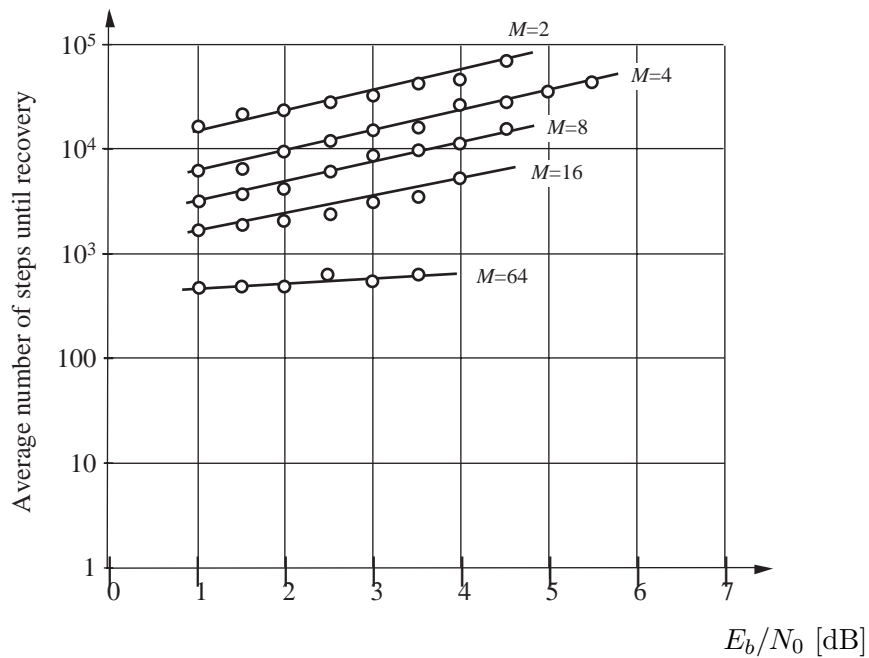


Figure 7.8: Average number of steps until recovery of the correct path for the code from Figure 7.6 (Source [27]).

Figure 7.9 shows the average number of steps until recovery for the rate  $1/2$ ,  $\nu = 11$  systematic convolutional code with generator polynomials  $g^{(0)} = 4000$ ,  $g^{(1)} = 7153$ . This code has a free Hamming distance of only  $d_{\text{free}} = 9$ , but its recovery performance is much superior to that of the non-systematic code. In fact, the average number of steps until recovery is independent of the signal-to-noise ratio, while it increases approximately linearly with  $E_b/N_0$  for the non-systematic code. This rapid recovery results in superior error performance of the systematic code compared to the non-systematic

code, shown in Figure 7.10, even though its free distance is significantly smaller. What is true, however, and can be seen clearly in Figure 7.10, is that for very large values of  $E_b/N_0$  the “stronger” code will win out due to its larger free distance.

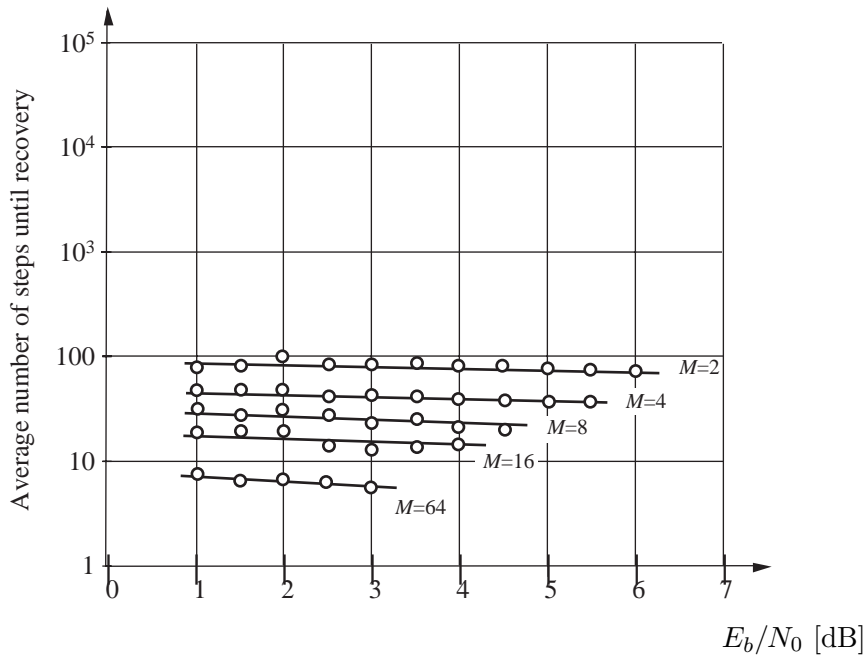


Figure 7.9: Average number of steps until recovery of the correct path for the systematic convolutional code with  $\nu = 11$  (Source [27]).

The observation that systematic convolutional codes outperform non-systematic codes for error rates  $P_b \gtrsim 10^{-6}$  has also been made by Osthoff et. al. [30]. The reason for this difference lies in the *return barrier* phenomenon, which can be explained with the aid of Figure 7.11. In order for the algorithm to recapture the correct path after a correct path loss, one of the  $M$  retained paths must correspond to a trellis state with a connection to the correct state at the next time interval. In Figure 7.11 we assume that the all-zero sequence is the correct sequence, and hence the all-zero state is the correct state for all time intervals. This assumption is made without loss of generality for convolutional codes due to their linearity. For a feed-forward realization of

a rate 1/2 code, the only state which connects to the all-zero state is the state  $(0, \dots, 0, 1)$ , denoted by  $s_m$  in the figure. In the case of a systematic code with  $g_0^{(1)} = g_\nu^{(1)} = 1$  (see Chapter 4) the two possible branch signals are (01) and (10) as indicated in Figure 7.11.

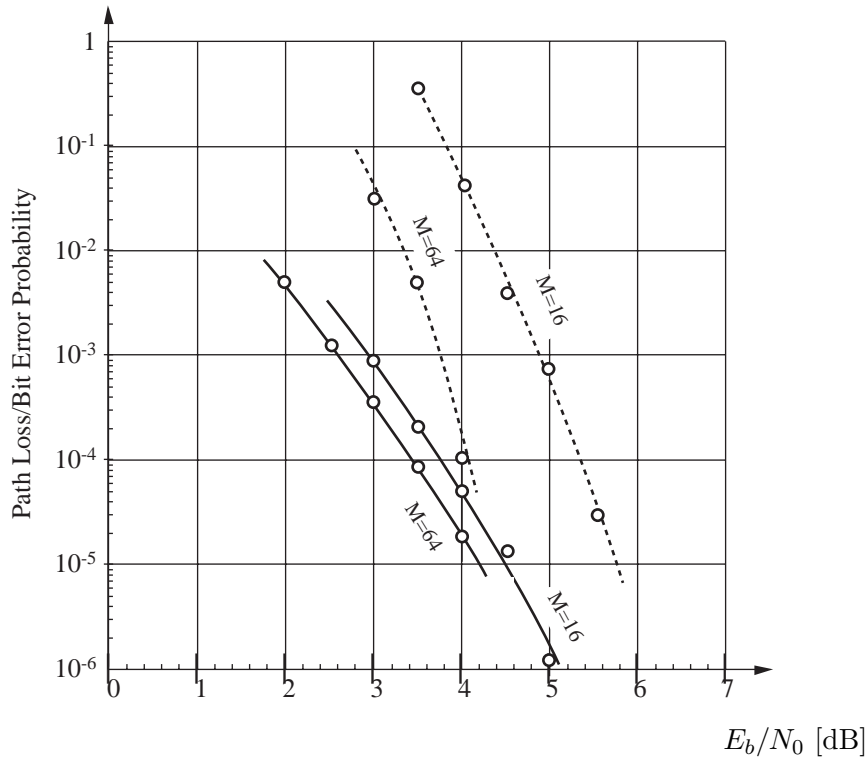


Figure 7.10: Simulation results for the superior 2048-state systematic code using the  $M$ -algorithm. The dashed curves are the error performance of the same constraint length non-systematic code from Figure 7.6 (Source [27]).

For a non-systematic, maximum free distance code on the other hand, the two branch signals are (11) and (00), respectively. Since the correct branch signal is (00), the probability that the metric of  $s_f$  (for failed) exceeds the metric of  $s_c$  equals 1/2 for the systematic code, since both branch signals are equidistant from the correct branch signal. For the non-systematic code on the other hand, this probability equals  $Q(\sqrt{E_s/N_0})$ . This explains the dependence of the path recovery on  $E_b/N_0$  for non-systematic codes, as well



as why systematic codes recapture the correct path faster with a recovery behavior which is independent of  $E_b/N_0$ .

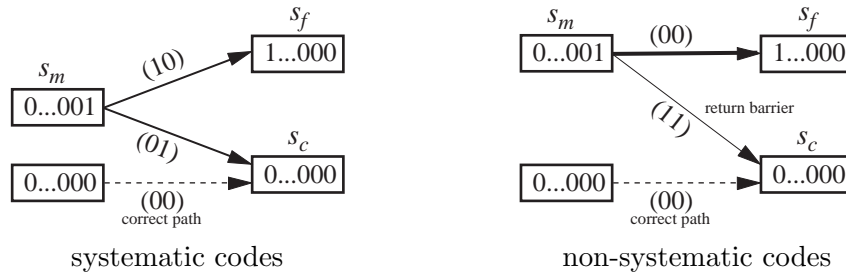


Figure 7.11: Heuristic explanation of the return barrier phenomenon in the  $M$ -algorithm.

The  $M$ -algorithm impresses with its simplicity. Unfortunately, a theoretical understanding of the algorithm is not related to this simplicity at all, and it seems that much more work in this area is needed before a coherent theory is available. This lack of a theoretical basis for the algorithm is, however, no barrier to its implementation. Early work on the application of the  $M$ -algorithm to convolutional codes, apart from Anderson [1, 2, 3, 30], was presented by Zigangirov and Kolesnik [46], while Simmons and Wittke [36], Aulin [6], and Balachandran [8], among others, have applied the  $M$ -algorithm to continuous-phase modulation. General trellis codes have not yet seen much action from the  $M$ -algorithm. An notable exception is [32].

It is generally felt that the  $M$ -algorithm is not a viable candidate algorithm for decoding binary convolutional codes, in particular with the emergence of Turbo codes and iterative decoding, however, it seems to work very well with non-binary modulations such as CPM, coded modulation, and code-division multiple access, where it may have a place in practical implementations.

## 7.6 Maximum Likelihood Decoding

The difficulty in decoding trellis codes arises from the exponential size of the growing decoding tree. In this section we will show that this tree can be reduced by merging nodes, such that the tree only grows to a maximum size of  $2^S$  nodes, where  $S$  is the number of encoder states. This merging

leads diverging paths together again and we obtain a structure resembling a trellis, as discussed for encoders in Section 3.

In order to see how this happens, let  $J_{n-1}^{(i)}$  and  $J_{n-1}^{(j)}$  be the metrics of two nodes corresponding to the partial sequences  $\tilde{\underline{x}}^{(i)}$  and  $\tilde{\underline{x}}^{(j)}$  of length  $n-1$ , respectively. Let the encoder states which correspond to  $\tilde{\underline{x}}^{(i)}$  and  $\tilde{\underline{x}}^{(j)}$  at time  $n-1$  be  $s_{n-1}^{(i)}$  and  $s_{n-1}^{(j)}$ ;  $s_{n-1}^{(i)} \neq s_{n-1}^{(j)}$ , and assume that the next extension of  $\tilde{\underline{x}}^{(i)} \rightarrow (\tilde{\underline{x}}^{(i)}, x_n^{(i)})$  and  $\tilde{\underline{x}}^{(j)} \rightarrow (\tilde{\underline{x}}^{(j)}, x_n^{(j)})$  is such that  $s_n^{(i)} = s_n^{(j)}$ , i.e., the encoder states at time  $n$  are identical. See also Figure 7.12 below.

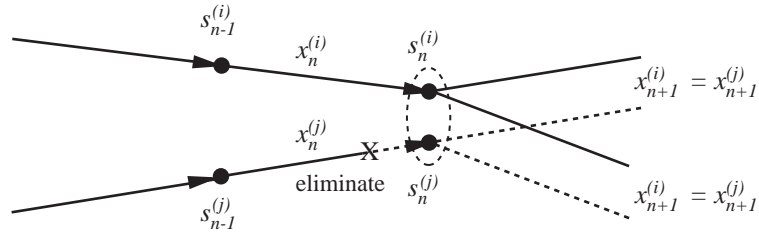


Figure 7.12: Merging nodes.

Now we propose to merge the two nodes  $(\tilde{\underline{x}}^{(i)}, x_n^{(i)})$  and  $(\tilde{\underline{x}}^{(j)}, x_n^{(j)})$  into one node, which we now call a (decoder) *state*. We retain the partial sequence which has the larger metric  $J_n$  at time  $n$  and discard the partial sequence with the smaller metric. Ties are broken arbitrarily. We are now ready to prove the following

**Theorem 7.1** (*Theorem of Non-Optimality*) *The procedure of merging nodes which correspond to identical encoder states, and discarding the path with the smaller metric never eliminates the maximum-likelihood path.*

Theorem 7.1 is sometimes referred to as the theorem of non-optimality and allows us to construct a maximum-likelihood decoder whose complexity is significantly smaller than that of an all-out exhaustive tree search.

*Proof:* The metric at time  $n+k$  for path  $i$  can be written as

$$J_{n+k}^{(i)} = J_n^{(i)} + \sum_{h=1}^k \beta_{n+h}^{(i)} \quad (7.25)$$

for every future time index  $n+k$ ;  $0 < k \leq l-n$ , where  $\beta_n^{(i)} = 2\text{Re} \{x_n^{(i)} v_n^*\} - |x_n^{(i)}|^2$  is the metric increment, now also called the *branch metric*, at time

$n$ . Now, if the nodes of path  $i$  and  $j$  correspond to the same encoder state at time  $n$ , there exists for every possible extension of the  $i$ -th path  $(x_{n+1}^{(i)}, \dots, x_{n+k}^{(i)})$  a corresponding identical extension  $(x_{n+1}^{(j)}, \dots, x_{n+k}^{(j)})$  of the  $j$ -th path. Let us then assume without loss of generality that the  $i$ -th path accumulates the largest metric at time  $l$ , *i.e.*,  $J_l^{(i)} \geq J_l^{(j)}$ . Therefore

$$J_n^{(i)} + \sum_{h=1}^{n-l} \beta_{n+h}^{(i)} \geq J_n^{(j)} + \sum_{h=1}^{n-l} \beta_{n+h}^{(j)}, \quad (7.26)$$

and  $\sum_{h=1}^{n-l} \beta_{n+h}^{(i)}$  is the maximum metric sum for the extensions from node  $(\tilde{x}^{(i)}, x_n^{(i)})$ . (Otherwise another path would have a higher final metric). But since the extensions for both paths are identical,  $\sum_{h=1}^{n-l} \beta_{n+h}^{(i)} = \sum_{h=1}^{n-l} \beta_{n+h}^{(j)}$  and  $J_n^{(i)} \geq J_n^{(j)}$ . Path  $j$  can therefore never accumulate a larger metric than path  $i$  and we may discard it with impunity at time  $n$ . Q.E.D.

The tree now folds back on itself and forms a trellis with exactly  $S$  states (See also Figure 3.2), and there are  $2^k$  paths merging in a single state at each step. Note then that there are now at most  $S$  retained partial sequences  $\tilde{x}^{(i)}$ , called the *survivors*. The most convenient labeling convention is that each state is labeled by the corresponding encoder state, plus the survivor which leads to it. This trellis is an exact replica of the encoder trellis discussed in Chapter 3 and the task of the decoder is to retrace the path the encoder traced through this trellis. Theorem 7.1 guarantees that this procedure is optimal. This method was introduced by Viterbi in 1967 [38, 31] in the context of analyzing convolutional codes, and has since become widely known as the *Viterbi Algorithm* [17]:

**Step 1:** Initialize the  $S$  states of the decoder with a metric  $J_{-l}^{(i)} = -\infty$  and survivors  $\tilde{x}^{(i)} = \{\}$ . Initialize the starting state of the encoder, usually state  $i = 0$ , with the metric  $J_{-l}^{(0)} = 0$ . Let  $n = -l$ .

**Step 2:** Calculate the branch metric

$$\beta_n = 2\text{Re} \{x_n v_n^*\} - |x_n|^2 \quad (7.27)$$

for each state  $s_n^{(i)}$  and each extension  $x_n^{(i)}$ .

**Step 3:** Follow all trellis transitions  $s_n^{(i)} \rightarrow s_{n+1}^{(i)}$  determined by the encoder FSM and, from the  $2^k$  merging paths, retain the survivor  $\tilde{x}^{(i)}$  for which  $J_{n+1}^{(i)}$  is maximized.

**Step 4:** If  $n < l$ , let  $n = n + 1$  and go to Step 2.

**Step 5:** Output the survivor  $\underline{x}^{(i)}$  which maximizes  $J_l^{(i)}$  as the maximum-likelihood estimate of the transmitted sequence.

Steps 2 and 3 are the central operations of the Viterbi algorithm and are referred to as the Add-Compare-Select (ACS) step. That is, branch metrics are added to state metrics, comparisons are made among all incoming branches, and the largest-metric path is selected.

The Viterbi algorithm and the  $M$ -algorithm are both breadth-first searches and share some similarities. In fact, one often introduces the concept of mergers also in the  $M$ -algorithm in order to avoid carrying along suboptimal paths. In fact, the  $M$ -algorithm can be operated in the trellis rather than in the tree. The Viterbi algorithm has enjoyed tremendous popularity, not only in decoding trellis codes, but also in symbol sequence estimation over channels affected by intersymbol interference [33, 18], multi-user optimal detectors [37], and speech recognition. Whenever the underlying generating process can be modeled as a finite-state machine, the Viterbi algorithm finds application.

A rather large body of literature deals with the Viterbi decoder, and there are a number of good books dealing with the subject (e.g., [20, 33, 9, 39]). One of the more important results is that it can be shown that one does not have to wait until the entire sequence is decoded before starting to output the estimated symbols  $x_n^{(i)}$ , or the corresponding data. The probability that the symbols in all survivors  $\tilde{\underline{x}}^{(i)}$  are identical for  $m < n - n_t$ , where  $n$  is the current active decoding time and  $n_t$ , called the *truncation length* or *decision depth*, (Section 3.2 and equation (4.16)) is very close to unity for  $n_t \approx 5\nu$ . This has been shown to be true for rate 1/2 convolutional codes (page 182 [11]), but the argument can easily be extended to general trellis codes. We may therefore modify the algorithm to obtain a fixed-delay decoder by modifying Step 4 and 5 of the above Viterbi algorithm as follows:

**Step 4:** If  $n \geq n_t$  output  $x_{n-n_t}^{(i)}$  from the survivor  $\tilde{\underline{x}}^{(i)}$  with the largest metric  $J_n^{(i)}$  as the estimated symbol at time  $n - n_t$ . If  $n < l - 1$ , let  $n = n + 1$  and go to Step 2.

**Step 5:** Output the remaining estimated symbols  $x_n^{(i)}$ ;  $l - n_t < n \leq l$  from the survivor  $\underline{x}^{(i)}$  which maximizes  $J_l^{(i)}$ .

We recognize that we may now let  $l \rightarrow \infty$ , *i.e.*, the complexity of our decoder is no longer determined by the length of the sequence, and it may

be operated in a continuous fashion. The simulation results in Chapter 5 were obtained with a Viterbi decoder according to the modified algorithm.

Let us spend some thoughts on the complexity of the Viterbi algorithm. Denote by  $E$  the total number of branches in the trellis, *i.e.*, for a linear-trellis there are  $S2^k$  branches per time epoch. The complexity requirements of the Viterbi algorithm can then be captured by the following [28]

**Theorem 7.2** *The Viterbi algorithm requires a complexity which is linear in the number of edges  $E$ , *i.e.*, it performs  $O(E)$  arithmetic operations (multiplications, additions and comparisons).*

*Proof:* Step 2 in the Viterbi algorithm requires the calculation of  $\beta_n$ , which needs two multiplies and an addition, as well as the addition  $J_n^{(i)} + \beta_n$  for each branch. Some of the values  $\beta_n$  may be identical, the number of arithmetic operations is therefore larger than  $E$  additions and less than  $2E$  multiplications and additions.

If we denote the number of branches entering state  $s$  by  $\rho(s)$ , step 3 requires  $\sum_{\text{states } s} (\rho(s) - 1) \leq E/2l$  comparisons per time epoch.  $\rho(s) = 2^k$  in our case, and the total number of comparisons is therefore less than  $E$ , and larger than  $E - 2lS$ .

There are then together  $O(E)$  arithmetic operations required. Q.E.D.

## 7.7 A Posteriori Probability Symbol Decoding

The purpose of the a posteriori probability (APP) algorithm is to compute a posteriori probabilities on either the information bits or the encoded symbols. These probabilities are mainly important in the iterative decoding algorithms for turbo codes discussed later in this book. Maximizing the a posteriori probabilities by themselves leads to only minor improvements in terms of bit error rates compared to the Viterbi algorithm. The algorithm was originally invented by Bahl, Cocke, Jelinek, and Raviv [7] in 1972 and was used to maximize the probability of each symbol being correct, referred to as the maximum a posteriori probability (MAP) algorithm. As mentioned, this algorithm was not widely used since it provided no significant improvement over maximum-likelihood decoding, and was significantly more complex.

With the invention of Turbo codes in 1993, however, the situation turned, and the APP became the major representative of the so-called soft-in soft-out (SISO) algorithms for providing probability information on the symbols

of a trellis code. These probabilities are required for iterative decoding schemes and concatenated coding schemes with soft decision decoding of the inner code, such as iterative decoding of turbo codes, which is discussed in Chapter 8.

Due to its importance we will first give a functional description of the algorithm before deriving the formulas in detail. Figure 7.13 shows the example trellis of a short terminated trellis code with seven sections. The transmitted signal is  $\underline{x} = [x_0, \dots, x_6]$ , and the information symbols are  $\underline{u} = [u_0, \dots, u_4, u_5 = 0, u_6 = 0]$ , i.e., there are two tail bits that drive the encoder back into the zero-state.

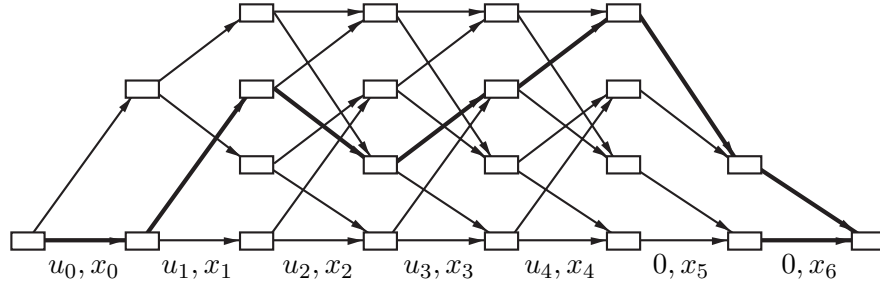


Figure 7.13: Example trellis of a short terminated trellis code.

The ultimate purpose of the algorithm is the calculation of a posteriori probabilities, such as  $\Pr[u_r|\underline{y}]$ , or  $\Pr[x_r|\underline{y}]$ , where  $\underline{y}$  is the received sequence observed at the output of a channel, whose input is the transmitted sequence  $\underline{x}$ . However, conceptually, it is more immediate to calculate the probability that the encoder traversed a specific transition in the trellis, i.e.,  $\Pr[s_r = i, s_{r+1} = j|\underline{y}]$ , where  $s_r$  is the state at epoch  $r$ , and  $s_{r+1}$  is the state at epoch  $r + 1$ . The algorithm computes this probability as the product of three terms:

$$\begin{aligned} \Pr[s_r = i, s_{r+1} = j|\underline{y}] &= \frac{1}{\Pr(\underline{y})} \Pr[s_r = i, s_{r+1} = j, \underline{y}] \\ &= \frac{1}{\Pr(\underline{y})} \alpha_{r-1}(i) \gamma_r(j, i) \beta_r(j). \end{aligned} \quad (7.28)$$

The  $\alpha$ -values are internal variables of the algorithm and are computed by the *forward recursion*

$$\alpha_{r-1}(i) = \sum_{\text{states } l} \alpha_{r-2}(l) \gamma_{r-1}(i, l). \quad (7.29)$$

This forward recursion evaluates  $\alpha$ -values at time  $r - 1$  from previously calculated  $\alpha$ -values at time  $r - 2$ , and the sum is over all states  $l$  at time  $r - 2$  that connect with state  $i$  at time  $r - 1$ . The forward recursion is illustrated in Figure 7.14. The  $\alpha$  values are initiated as  $\alpha(0) = 1, \alpha(1) = \alpha(2) = \alpha(3) = 0$ . This automatically enforces the boundary condition that the encoder starts in state 0.

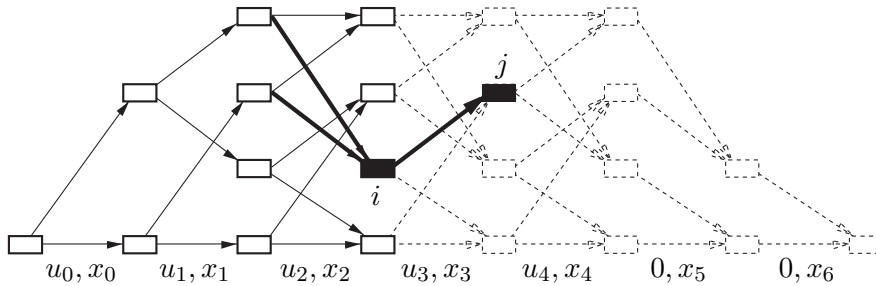


Figure 7.14: Illustration of the forward recursion of the APP algorithm.

The  $\beta$ -values are calculated by an analogous procedure, called the *backward recursion*

$$\beta_r(j) = \sum_{\text{states } k} \beta_{r+1}(k) \gamma_{r+1}(k, j), \quad (7.30)$$

and initialized as  $\beta(0) = 1, \beta(1) = \beta(2) = \beta(3) = 0$  to enforce the terminating condition of the trellis code. The sum is over all states  $k$  at time  $r + 1$  to which state  $j$  at time  $r$  connects. The backward recursion is illustrated in Figure 7.15.

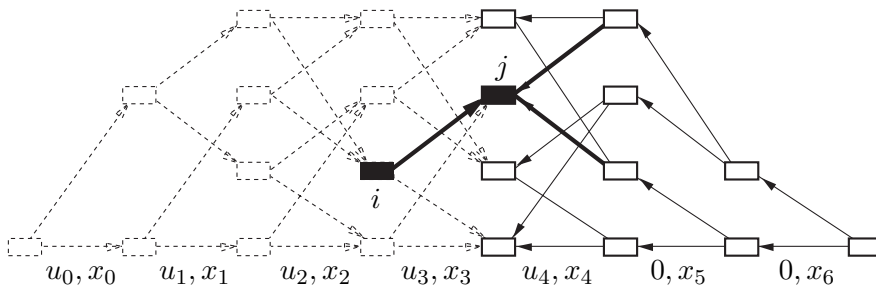


Figure 7.15: Illustration of the backward recursion of the APP algorithm.

The  $\gamma$  values are conditional transition probabilities, and are the inputs to the algorithm.  $\gamma_r(j, i)$  is the joint probability that the state at time  $r + 1$  is  $s_{r+1} = j$ , and that  $y_r$  is received, it is calculated as

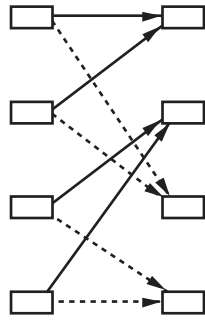
$$\gamma_r(j, i) = \Pr(s_{r+1} = j, y_r | s_r = i) = \Pr[s_{r+1} = j | s_r = i] \Pr(y_r | x_r). \quad (7.31)$$

The first term,  $\Pr[s_{r+1} = j | s_r = i]$  is the a priori transition probability, and is related to the probability of  $u_r$ . In fact, in our example, the top transition is associated with  $u_r = 1$  and the bottom transition with  $u_r = 0$ . This factor, can and will be used to account for a priori probability information on the bits  $u_r$ . In the sequel we will abbreviate this transition probability by

$$p_{ij} = \Pr(s_{r+1} = j | s_r = i) = \Pr(u_r). \quad (7.32)$$

The second term,  $\Pr(y_r | x_r)$ , is simply the conditional channel transition probability, given that symbol  $x_r$  is transmitted. Note that  $x_r$  is the symbol associated with the transition from state  $i \rightarrow j$ .

The a posteriori symbol probabilities  $\Pr[u_r | \underline{y}]$  can now be calculated from the a posteriori transition probabilities (7.28) by summing over all transitions corresponding to  $u_r = 1$ , and, separately, by summing over all transitions corresponding to  $u_r = 0$ , to obtain



$$p[u_r = 1 | \underline{y}] = \frac{1}{\Pr(\underline{y})} \sum_{\text{solid}} \Pr[s_r = i, s_{r+1} = j, \underline{y}] \quad (7.33)$$

$$p[u_r = 0 | \underline{y}] = \frac{1}{\Pr(\underline{y})} \sum_{\text{dashed}} \Pr[s_r = i, s_{r+1} = j, \underline{y}]. \quad (7.34)$$

The solid transition correspond to  $u_r = 1$ , and the dashed transitions correspond to  $u_r = 0$  as illustrated on the left.

A formal algorithm description is given at the end of this section, but first we present a rigorous derivation of the APP algorithm. This derivation was first given by Bahl et. al. [7].

In the general case we will have need for the probability

$$q_{ij}(x) = \Pr(\tau(u_r, s_r) = x | s_r = i, s_{r+1} = j), \quad (7.35)$$

that is, is the a priori probability that the output  $x_r$  at time  $r$  assumes the value  $x$  on the transition from state  $i$  to state  $j$ . This probability is typically



a deterministic function of  $i$  and  $j$ , unless there are parallel transitions, in which case  $x_r$  is determined by the uncoded information bits (see Section 3.4).

Before we proceed with the derivation, let us define the internal variables  $\alpha$  and  $\beta$  by their probabilistic meaning. These are

$$\alpha_r(j) = \Pr(s_{r+1} = j, \tilde{\mathbf{y}}), \quad (7.36)$$

the joint probability of the partial sequence  $\tilde{\mathbf{y}} = (y_{-l}, \dots, y_r)$  up to and including time epoch  $r$  and state  $s_{r+1} = j$ ; and

$$\beta_r(j) = \Pr((y_{r+1}, \dots, y_l) | s_{r+1} = j), \quad (7.37)$$

the conditional probability of the remainder of the received sequence  $\underline{\mathbf{y}}$  given that the state at time  $r + 1$  is  $j$ .

With the above we now calculate

$$\begin{aligned} \Pr(s_{r+1} = j, \underline{\mathbf{y}}) &= \Pr(s_{r+1} = j, \tilde{\mathbf{y}}, (y_{r+1}, \dots, y_l)) \\ &= \Pr(s_{r+1} = j, \tilde{\mathbf{y}}) \Pr((y_{r+1}, \dots, y_l) | s_{r+1} = j, \tilde{\mathbf{y}}) \\ &= \alpha_r(j) \beta_r(j), \end{aligned} \quad (7.38)$$

where we have used the fact that  $\Pr((y_{r+1}, \dots, y_l) | s_{r+1} = j, \tilde{\mathbf{y}}) = \Pr((y_{r+1}, \dots, y_l) | s_{r+1} = j)$ , *i.e.*, if  $s_{r+1} = j$  is known, events after time  $r$  are independent of the history  $\tilde{\mathbf{y}}$  up to  $s_{r+1}$ .

In the same way we calculate via Bayes' expansion

$$\begin{aligned} \Pr(s_r = i, s_{r+1} = j, \underline{\mathbf{y}}) &= \Pr(s_r = i, s_{r+1} = j, (y_{-l}, \dots, y_{r-1}), y_r, (y_{r+1}, \dots, y_l)) \\ &= \Pr(s_r = i, (y_{-l}, \dots, y_{r-1})) \Pr(s_{r+1} = j, y_r | s_r = i) \\ &\quad \times \Pr((y_{r+1}, \dots, y_l) | s_{r+1} = j) \\ &= \alpha_{r-1}(i) \gamma_r(j, i) \beta_r(j). \end{aligned} \quad (7.39)$$

Now, again applying Bayes' rule and  $\sum_b p(a, b) = p(a)$ , we obtain

$$\begin{aligned} \alpha_r(j) &= \sum_{\text{states } i} \Pr(s_r = i, s_{r+1} = j, \tilde{\mathbf{y}}) \\ &= \sum_{\text{states } i} \Pr(s_r = i, (y_{-l}, \dots, y_{r-1})) \Pr(s_{r+1} = j, y_r | s_r = i) \\ &= \sum_{\text{states } i} \alpha_{r-1}(i) \gamma_r(j, i). \end{aligned} \quad (7.40)$$

For a trellis code started in the zero state at time  $r = -l$  we have the starting conditions

$$\alpha_{-l-1}(0) = 1, \alpha_{-l-1}(j) = 0; \quad j \neq 0. \quad (7.41)$$

As above, we similarly develop an expression for  $\beta_r(j)$ , *i.e.*,

$$\begin{aligned}
\beta_r(j) &= \sum_{\text{states } i} \Pr(s_{r+2} = i, (y_{r+1}, \dots, y_l) | s_{r+1} = j) \\
&= \sum_{\text{states } i} \Pr(s_{r+2} = i, y_{r+1} | s_{r+1} = j) \Pr((y_{r+2}, \dots, y_l) | s_{r+2} = i) \\
&= \sum_{\text{states } i} \beta_{r+1}(i) \gamma_{r+1}(i, j). \tag{7.42}
\end{aligned}$$

The boundary condition for  $\beta_r(j)$  is

$$\beta_l(0) = 1, \beta_l(j) = 0; \quad j \neq 0, \tag{7.43}$$

for a trellis code which is terminated in the zero state.

Furthermore, the general form of the  $\gamma$  values is given by

$$\begin{aligned}
\gamma_r(j, i) &= \sum_{x_r} \Pr(s_{r+1} = j | s_r = i) \Pr(x_r | s_r = i, s_{r+1} = j) \Pr(y_r | x_r) \\
&= \sum_{x_r} p_{ij} q_{ij}(x_r) p_n(y_r - x_r), \tag{7.44}
\end{aligned}$$

where we have used the conditional density function of the AWGN channel from (2.11), *i.e.*,  $\Pr(y_r | x_r) = p_n(y_r - x_r)$ . The calculation of  $\gamma_r(j, i)$  is not very complex and can most easily be implemented by a table lookup procedure.

Equations (7.40) and (7.42) are iterative and we can now compute the a posteriori state and transition probabilities via the following algorithm:

**Step 1:** Initialize  $\alpha_{-l-1}(0) = 1, \alpha_{-l-1}(j) = 0$  for all non-zero states ( $j \neq 0$ ) of the encoder FSM, and  $\beta_l(0) = 1, \beta_l(j) = 0, j \neq 0$ . Let  $r = -l$ .

**Step 2:** For all states  $j$  calculate  $\gamma_r(j, i)$  and  $\alpha_r(j)$  via (7.44) and (7.40).

**Step 3:** If  $r < l$ , let  $r = r + 1$  and go to Step 2, else  $r = l - 1$  and go to Step 4.

**Step 4:** Calculate  $\beta_r(j)$  using (7.42). Calculate  $\Pr(s_{r+1} = j, \underline{y})$  from (7.38), and  $\Pr(s_r = i, s_{r+1} = j; \underline{y})$  from (7.28).

**Step 5:** If  $r > -l$ , let  $r = r - 1$  and go to Step 4.

**Step 6:** Terminate the algorithm and output all the values  $\Pr(s_{r+1} = j, \underline{y})$  and  $\Pr(s_r = i, s_{r+1} = j, \underline{y})$ .

Contrary to the maximum likelihood algorithm, the APP algorithms needs to go through the trellis twice, once in the forward direction, and once in the reverse direction. What is worse, all the values  $\alpha_r(j)$  must be stored from the first pass through the trellis. For a rate  $k/n$  convolutional code, for example, this requires  $2^{k\nu}2l$  storage locations since there are  $2^{k\nu}$  states for each of which we need to store a different value  $\alpha_r(j)$  at each time epoch  $r$ . The storage requirement grows exponentially in the constraint length  $\nu$  and linearly in the block length  $2l$ .

The a posteriori state and transition probabilities produced by this algorithm can now be used to calculate a posteriori information bit probabilities, *i.e.*, the probability that the information  $k$ -tuple  $u_r = u$ , where  $u$  can vary over all possible binary  $k$ -tuples. Starting from the transition probabilities  $\Pr(s_r = i, s_{r+1} = j|\underline{y})$  we simply sum over all transitions  $i \rightarrow j$  which are caused by  $u_r = u$ . Denoting these transitions by  $A(u)$ , we obtain

$$\Pr(u_r = u) = \sum_{(i,j) \in A(u)} \Pr(s_r = i, s_{r+1} = j|\underline{y}). \quad (7.45)$$

As mentioned above, another most interesting product of the APP decoder is the a posteriori probability of the transmitted output symbol  $x_r$ . Arguing analogously as above, and letting  $B(x)$  be the set of transitions on which the output signal  $x$  can occur, we obtain

$$\begin{aligned} \Pr(x_r = x) &= \sum_{(i,j) \in B(x)} \Pr(x|y_r)\Pr(s_r = i, s_{r+1} = j|\underline{y}) \\ &= \sum_{(i,j) \in B(x)} \frac{p_n(y_r - x_r)}{p(y_r)} q_{ij}(x) \Pr(s_r = i, s_{r+1} = j|\underline{y}) \end{aligned} \quad (7.46)$$

where the a priori probability of  $y_r$  can be calculated via

$$p(y_r) = \sum_{\substack{x' \\ ((i,j) \in B(x))}} p(y_r|x')q_{ij}(x'), \quad (7.47)$$

and the sum extends over all transitions  $i \rightarrow j$ .

Equation (7.46) can be much simplified if there is only one output symbol on the transition  $i \rightarrow j$  as in the introductory discussion. In that case the transition automatically determines the output symbol, and

$$\Pr(x_r = x) = \sum_{(i,j) \in B(x)} \Pr(s_r = i, s_{r+1} = j|\underline{y}). \quad (7.48)$$

One problem we have to address is that of numerical stability. The  $\alpha$  and  $\beta$  vales in (7.40) and (7.42) decay rapidly and will underflow in any fixed precision implementation. We therefore normalize both values at each epoch, i.e.,

$$\alpha_r(i) \rightarrow \frac{\alpha_r(i)}{\sum_s \alpha_r(s)}; \quad \beta_r(i) \rightarrow \frac{\beta_r(i)}{\sum_s \beta_r(s)} \quad (7.49)$$

This normalization has no effect on our final results such as (7.46), since these are similarly normalized. In fact, this normalization allows us to ignore the division by  $p(y_r)$  in (7.46), and division by  $\Pr(\underline{y})$  in (7.28), (7.33), and (7.34).

## 7.8 Log-APP, Max-Log-APP, and Approximations

### 7.8.1 The APP in the Logarithm Domain (Log-APP)

While the APP algorithm is concise and consists only of multiplications and additions, current direct digital hardware implementations of the algorithm lead to complex circuits due to many real number multiplications involved in the algorithm. In order to avoid these multiplications, we transform the algorithm into the logarithm-domain. This results in the so-called *log-APP* algorithm.

First we transform the forward recursion (7.29), (7.40) into the logarithm-domain using the definitions

$$A_r(i) = \log(\alpha_r(i)); \quad \Gamma_r(i, l) = \log(\gamma_r(i, l)) \quad (7.50)$$

to obtain the *log-domain* forward recursion

$$A_{r-1}(i) = \log \left( \sum_{\text{states } l} \exp \left( A_{r-2}(l) + \Gamma_{r-1}(i, l) \right) \right) \quad (7.51)$$

Likewise the backward recursion can be transformed into the logarithm-domain using the analogous definition  $B_r(j) = \log(\beta_r(j))$ , and we obtain

$$B_r(j) = \log \left( \sum_{\text{states } k} \exp \left( B_{r+1}(l) + \Gamma_{r+1}(k, j) \right) \right) \quad (7.52)$$

The product in (7.28) and (7.39) now turns into the simple sum

$$\alpha_{r-1}(i) \gamma_r(j, i) \beta_r(j) \rightarrow A_{r-1}(i) + \Gamma_r(j, i) + B_r(j) \quad (7.53)$$

Unfortunately, equations (7.51) and (7.52) contain  $\log()$  and  $\exp()$  functions, which seem even more complex than the original multiplications. This is true, however, in most cases of current practical interest, the APP algorithm is used to decode binary codes, i.e., there are only two branches involved at each state, and therefore only sums of two terms in (7.51) and (7.52). The logarithm of such a binary sum can be expanded as

$$\begin{aligned} \log(\exp(a) + \exp(b)) &= \log\left(\exp\left(\max(a, b)\right)\left(1 + \exp(-|a - b|)\right)\right) \\ &= \max(a, b) + \log\left(\left(1 + \exp(-|a - b|)\right)\right) \end{aligned}$$

It seems that little is gained from these manipulations, but the second term is now the only complex operation left, and there are a number of ways to approach this. The first, and most complex but precise method is to store the function

$$\log\left(\left(1 + \exp(-x)\right)\right); \quad x = |a - b|, \quad (7.54)$$

in a ROM look-up table. Given an example quantization of 4bits, this is a  $16 \times 16$  value look-up table, which is very manageable. Figure 7.8.1 shows the signal flow of this binary log-domain operation on the example of a node operation in the forward recursion.

Finally, to binary codes the algorithm computes the log-likelihood ratio (LLR)  $\lambda(u_r)$  of the information bits  $u_r$  using the a posteriori probabilities (7.45) as

$$\begin{aligned} \lambda(u_r) &= \log\left(\frac{\Pr(u_r = 1)}{\Pr(u_r = 0)}\right) = \log\left(\frac{\sum_{(i,j) \in A(u=1)} \alpha_{r-1}(i) \gamma_r(j, i) \beta_r(j)}{\sum_{(i,j) \in A(u=0)} \alpha_{r-1}(i) \gamma_r(j, i) \beta_r(j)}\right) \\ \lambda(u_r) &= \log\left(\frac{\sum_{(i,j) \in A(u=1)} \exp(A_{r-1}(i) + \Gamma_r(j, i) + B_r(j))}{\sum_{(i,j) \in A(u=0)} \exp(A_{r-1}(i) + \Gamma_r(j, i) + B_r(j))}\right). \quad (7.55) \end{aligned}$$

The LLR is the quantity which is used in the iterative decoding algorithms of binary turbo codes as discussed later in this book. The range of the LLR is  $[-\infty, \infty]$ , where a large value signifies a high probability that  $u_r = 1$ .

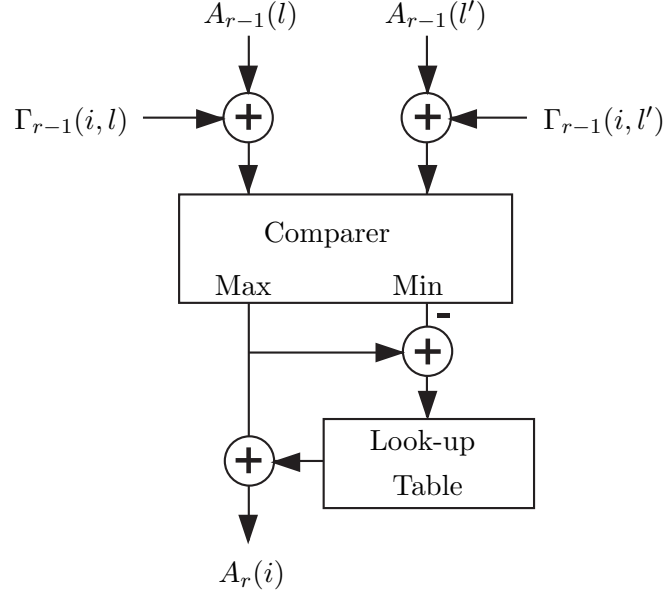


Figure 7.16: Signal flow diagram of the node calculation of the Log-APP algorithm.

### 7.8.2 Max-Log-APP

A straight-forward way of reducing the complexity of the Log-APP is to eliminate the ROM lookup table in Figure 7.8.1, [34]. This has the effect of approximating the forward and backward recursions by

$$\begin{aligned}
 A_{r-1}(i) &= \log \left( \sum_{\text{states } l} \exp \left( A_{r-2}(l) + \Gamma_{r-1}(i, l) \right) \right) \\
 &\approx \max_{\text{states } l} \left( A_{r-2}(l) + \Gamma_{r-1}(i, l) \right)
 \end{aligned} \tag{7.56}$$

and

$$\begin{aligned}
 B_r(j) &= \log \left( \sum_{\text{states } k} \exp \left( B_{r+1}(l) + \Gamma_{r+1}(k, j) \right) \right) \\
 &\approx \max_{\text{states } k} \left( B_{r+1}(l) + \Gamma_{r+1}(k, j) \right)
 \end{aligned} \tag{7.57}$$

It is very interesting to note that (7.56) is nothing else than our familiar Viterbi algorithm for maximum-likelihood sequence decoding. Furthermore,

equation (7.57) is also a Viterbi algorithm, but operated in the reverse direction.

Analogously, the final LLR calculation in (7.55) is approximated by

$$\lambda(u_r) \approx \max_{(i,j) \in A(u=1)} \left( A_{r-1}(i) + \Gamma_r(j, i) + B_r(j) \right) - \max_{(i,j) \in A(u=1)} \left( A_{r-1}(i) + \Gamma_r(j, i) + B_r(j) \right). \quad (7.58)$$

The big advantage of the Log-Max-APP algorithm is that it only uses additions and maximization operations to approximate the LLR of  $u_r$ . This computational savings is paid for by an approximate 0.5dB loss when these decoders are used to decode Turbo codes.

Further insight into the relationship between the Log-APP and its approximation can be gained by considering the expressing the LLR of  $u_r$  in its basic form, i.e.,

$$\lambda(u_r) = \log \left( \frac{\sum_{\underline{x};(u_r=1)} \exp \left( -\frac{|\underline{y} - \underline{x}|^2}{N_0} \right)}{\sum_{\underline{x};(u_r=0)} \exp \left( -\frac{|\underline{y} - \underline{x}|^2}{N_0} \right)} \right), \quad (7.59)$$

where the sum in the numerator extends over all coded sequences  $\underline{x}$  which correspond to information bit  $u_r = 1$ , and the denominator sum extends over all  $\underline{x}$  corresponding to  $u_r = 0$ .

It is quite straightforward to see that the MAX-Log-APP retains only the path in each sum which has the best metrics, and therefore the MAX-Log-APP calculates an approximation to the true LLR, given by

$$\lambda(u_r) \approx \min_{\underline{x};(u_r=0)} \frac{|\underline{y} - \underline{x}|^2}{N_0} - \min_{\underline{x};(u_r=1)} \frac{|\underline{y} - \underline{x}|^2}{N_0}, \quad (7.60)$$

i.e., the metric difference between the nearest path to  $\underline{y}$  with  $u_r = 0$  and the nearest path with  $u_r = 1$ . For constant energy signals this simplifies to

$$\lambda(u_r) \approx \frac{(\underline{x}_r^{(1)} - \underline{x}_r^{(0)}) \cdot \underline{y}}{N_0/2}, \quad (7.61)$$

where  $\underline{x}_r^{(1)} = \arg \min_{\underline{x};(u_r=1)} |\underline{y} - \underline{x}|^2$ , and  $\underline{x}_r^{(0)} = \arg \min_{\underline{x};(u_r=0)} |\underline{y} - \underline{x}|^2$ .

### 7.8.3 Approximations

For high-speed turbo decoding applications requiring up to ten iterations, evaluation of equation (7.54) may be too complex, yet one is not readily willing to accept the half a dB loss entailed by using the Max-Log-APP. A very effective way of approximating (7.54) is [23]

$$\begin{aligned} \max(a, b) + \log \left( \left( 1 + \exp(-|a - b|) \right) \right) \\ \approx \max(a, b) + \begin{cases} 0 & \text{if } |a - b| > T \\ C & \text{if } |a - b| \leq T. \end{cases} \end{aligned} \quad (7.62)$$

This simple threshold approximation is called *constant-Log-APP* algorithm. It is used in the UMTS turbo code [14], and leads to a degradation with respect to the full Log-APP of only 0.03dB on this code [12], where the optimal parameters for this code are determined to be  $C = 0.5$  and  $T = 1.5$ . This reduces the ROM look-up table of the Log-APP to a simple comparator circuit.

APP decoders are mainly used in decoders for Turbo codes of various sizes. It is therefore desirable to make the APP algorithm itself independent of the block size of the overall code. While the forward recursion can be performed in synchrony with the incoming data, the backward recursion poses a problem, since the end of the block would need to be received before it can be started. A solution lies performing a *partial backward recursion*, starting some  $D$  symbol epochs in the future and using these values to calculate the LLRs at epoch  $r$ . The basic notion of this *sliding window* implementation is illustrated in Figure 7.17.

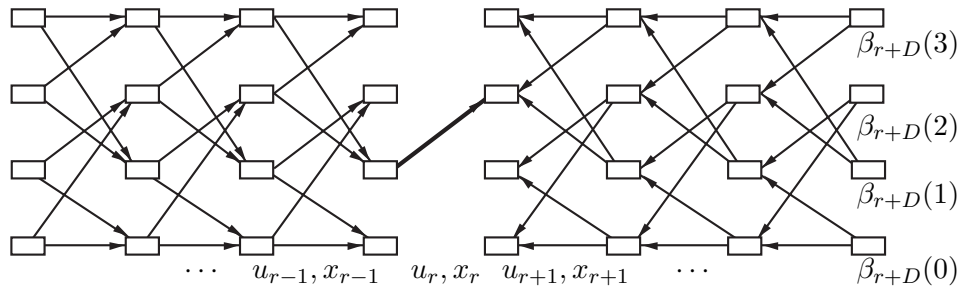


Figure 7.17: Sliding window approximation to the APP algorithm.

The question now is how to initialize the values  $\beta_{r+D}(j)$ , and the most



typical method is to give them all the same value; a uniform initialization. Note that the exact values is irrelevant since the LLR eliminates constants.

Note that at first sight it seems that we have traded in a largely increased computational load, since for each forward recursion step,  $D$  backward recursion steps are needed to find the values of  $\beta$  at epoch  $r$ . However, it is computationally much more efficient to operate this algorithm in a block fashion. That is, for every  $D$  backward recursion steps, not only a single forward step at  $r$  is executed, but a number  $R$  of forward steps. Typical values are  $D = 2R$ , which leads to efficient shared memory implementations.

## 7.9 Random Coding Analysis of Sequential Decoding

In Section 5.5 we presented random coding performance bounds for trellis codes. In that section we implicitly assumed that we were using a maximum likelihood decoder. Since sequential decoding is not a maximum likelihood decoding method, the results in Section 5.5 do not apply.

The error analysis of sequential decoding is very difficult, and, again, we find it easier to generate results for the ensemble of all trellis codes via random coding arguments. The evaluation of the error probability is not the main problem here, since, if properly dimensioned, both the stack and the Fano algorithm will almost always find the same error path as the maximum likelihood detector.

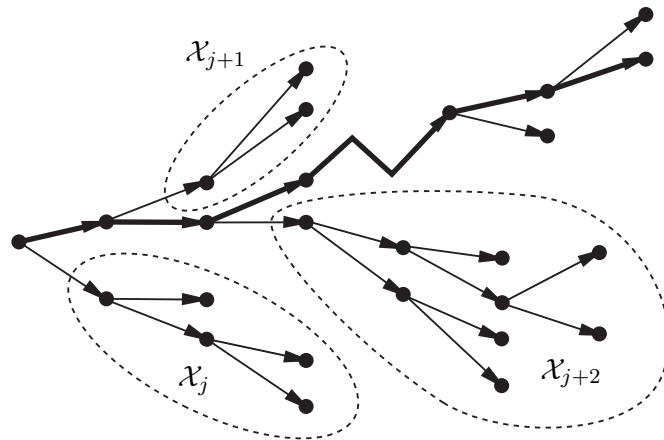


Figure 7.18: Incorrect subsets explored by a sequential decoder. The solid path is the correct one.

The difference with sequential decoding is, in contrast to ML-decoding, that its computational load is variable. And it is this computational load which can cause problems as we will see. Figure 7.18 shows an example of the search procedure of sequential decoding. The algorithm explores at each node an entire set of incorrect partial paths before finally continuing. This set at each node includes all the incorrect paths explored by the possibly multiple visits to that node as for example in the Fano algorithm. The sets  $\mathcal{X}'_j$  denote the sets of incorrect signal sequences  $\tilde{\mathbf{x}}'$  diverging at node  $j$ , which are searched by the algorithm. Further, denote the number of signal sequences in  $\mathcal{X}'_j$  by  $C_j$ . Note that  $C_j$  is also the number of computations that need to be done at node  $j$ , since each new path requires one additional metric calculation. This is the case because the algorithm explores two extensions at each step for a binary code, both resulting in distinct extension paths (Figure 7.18).

The problem becomes quite evident now, the number of computations at each node is variable, and it is this distribution of the computations which we want to examine. Again, let  $\tilde{\mathbf{x}}$  be the partial correct path through the trellis and  $\tilde{\mathbf{x}}'_j$  be a partial incorrect path which diverges from  $\tilde{\mathbf{x}}$  at node  $j$ . Furthermore, let  $L_n(\tilde{\mathbf{x}}') = L(\tilde{\mathbf{x}}', \underline{y})$  be the metric of the incorrect path at node  $n$ , and let  $L_m(\tilde{\mathbf{x}})$  be the metric of the correct path at node  $m$ . A path is searched further if and only if it is at the top of the stack, and hence, if  $L_n(\tilde{\mathbf{x}}') < L_m(\tilde{\mathbf{x}})$ , the incorrect path is not searched further until the metric of  $\tilde{\mathbf{x}}$  falls below  $L_n(\tilde{\mathbf{x}}')$ . If

$$\min_{m \geq j} L_m(\tilde{\mathbf{x}}) = \lambda_j > L_n(\tilde{\mathbf{x}}') \quad (7.63)$$

the incorrect path  $\tilde{\mathbf{x}}'$  is never searched beyond node  $n$ .

We may now overbound the probability that the number of computations at node  $j$  exceeds a given value  $N_c$  by

$$\Pr(C_j \geq N_c) \leq \sum_{\underline{x}} p(\underline{x}) \int_{\underline{y}} p(\underline{y}|\underline{x}) \mathcal{B}(|e(p(\underline{y}|\underline{x}') \geq \lambda_j)| \geq N_c) d\underline{y}, \quad (7.64)$$

where  $e(p(\underline{y}|\underline{x}') \geq \lambda_j)$  is an error path in  $\mathcal{X}'_j$  whose metric exceeds  $\lambda_j$  and  $|\star|$  is the number of such error paths.  $\mathcal{B}(\star)$  is a boolean function which equals 1 if the expression is true and 0 otherwise. The function  $\mathcal{B}(\star)$  in (7.64) then simply equals 1 if there are more than  $N_c$  error paths with metric larger than  $\lambda_j$  and 0 otherwise.

We now proceed to overbound the indicator function analogously to Chapter 5, by realizing that  $\mathcal{B}(\star) = 1$  if at least  $N_c$  error paths have a

metric such that

$$L_n(\tilde{\mathbf{x}}') \geq \lambda_j, \quad (7.65)$$

and, hence,

$$\left( \frac{1}{N_c} \sum_{\tilde{\mathbf{x}}' \in \mathcal{X}'_j} \exp(\alpha(L_n(\tilde{\mathbf{x}}') - \lambda_j)) \right)^\rho \geq 1, \quad (7.66)$$

where  $\alpha$  and  $\rho$  are arbitrary positive constants. Note that we have extended the sum in (7.66) over all error sequences as is customary in random coding analysis. We may now use (7.66) to overbound the indicator function  $\mathcal{B}(\star)$  and we obtain

$$\Pr(C_j \geq N_c) \leq N_c^{-\rho} \sum_{\underline{\mathbf{x}}} p(\underline{\mathbf{x}}) \int_{\underline{\mathbf{y}}} p(\underline{\mathbf{y}}|\underline{\mathbf{x}}) \left( \sum_{\tilde{\mathbf{x}}' \in \mathcal{X}'_j} \exp(\alpha(L_n(\tilde{\mathbf{x}}') - \lambda_j)) \right)^\rho d\underline{\mathbf{y}}, \quad (7.67)$$

and, due to (7.63)

$$\exp(-\alpha\rho\lambda_i) \leq \sum_{m=j}^{\infty} \exp(-\alpha\rho L_m(\tilde{\mathbf{x}})), \quad (7.68)$$

and we have

$$\begin{aligned} \Pr(C_j \geq N_c) &\leq N_c^{-\rho} \sum_{\underline{\mathbf{x}}} p(\underline{\mathbf{x}}) \int_{\underline{\mathbf{y}}} p(\underline{\mathbf{y}}|\underline{\mathbf{x}}) \left( \sum_{\tilde{\mathbf{x}}' \in \mathcal{X}'_j} \exp(\alpha L_n(\tilde{\mathbf{x}}')) \right)^\rho \\ &\quad \times \sum_{m=j}^{\infty} \exp(-\alpha\rho L_m(\tilde{\mathbf{x}})) d\underline{\mathbf{y}}. \end{aligned} \quad (7.69)$$

Analogously to Chapter 5, let  $c$  be the correct path and  $e$  be the incorrect path which diverges from  $c$  at node  $j$ . Let  $\mathcal{E}$  be the set of all incorrect paths, and  $\mathcal{E}'_j$  be the set of incorrect paths (not signal sequences) corresponding to  $\underline{\mathbf{x}}'_j$ . Again,  $\underline{\mathbf{x}}$  and  $\underline{\mathbf{x}}'$  are, strictly taken, the signal sequences assigned to the correct and incorrect path, respectively, and  $\tilde{\mathbf{x}}, \tilde{\mathbf{x}}'$  are the associated partial sequences. Let then  $\text{Avg}\{\Pr(C_j \geq N_c)\}$  be the ensemble average of

$\Pr(C_j \geq N_c)$  over all linear-trellis codes, *i.e.*,

$$\begin{aligned} \text{Avg}\{\Pr(C_j \geq N_c)\} &\leq \overline{N_c^{-\rho} \sum_c p(c) \int_{\underline{y}} p(\underline{y}|\underline{x}) \sum_{m=j}^{\infty} \exp(-\alpha\rho L_m(\tilde{x}))} \\ &\quad \times \left( \sum_{e \in \mathcal{E}'_j} \overline{\exp(\alpha L_n(\tilde{x}'))} \right)^\rho d\underline{y}. \quad (7.70) \end{aligned}$$

Note, we have used Jensen's inequality to pull the averaging into the second sum, which restricts  $\rho$  to  $0 \leq \rho \leq 1$ . Since we are using time-varying random trellis codes, (7.70) becomes independent of the starting node  $j$ , which we arbitrarily set to  $j = 0$  now.

Observe there are at most  $2^{kn}$  paths  $e$  of length  $n$  in  $\mathcal{E}'_j = \mathcal{E}'$ . Using this and the inequality<sup>3</sup>

$$\left( \sum a_i \right)^\rho \leq \sum a_i^\rho; \quad a_i \geq 0; \quad 0 \leq \rho \leq 1, \quad (7.71)$$

we obtain<sup>4</sup>

$$\begin{aligned} \text{Avg}\{\Pr(C_0 \geq N_c)\} &\leq \overline{N_c^{-\rho} \sum_c p(c) \int_{\underline{y}} p(\underline{y}|\underline{x}) \sum_{m=0}^{\infty} \exp(-\alpha\rho L_m(\tilde{x}))} \\ &\quad \times \sum_{n=0}^{\infty} 2^{kn\rho} \left( \overline{\exp(\alpha L_n(\tilde{x}'))} \right)^\rho d\underline{y} \quad (7.72) \\ &= N_c^{-\rho} \sum_c p(c) \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} 2^{kn\rho} \\ &\quad \times \overline{\int_{\underline{y}} p(\underline{y}|\underline{x}) \exp(-\alpha\rho L_m(\tilde{x})) \left( \overline{\exp(\alpha L_n(\tilde{x}'))} \right)^\rho d\underline{y}}. \quad (7.73) \end{aligned}$$

<sup>3</sup>This inequality is easily shown, *i.e.*,

$$\frac{\sum a_i^\rho}{(\sum a_i)^\rho} = \sum \left( \frac{a_i}{\sum a_i} \right)^\rho \geq \sum \left( \frac{a_i}{\sum a_i} \right) = 1,$$

where the inequality resulted from the fact that each term in the sum is  $\leq 1$  and  $\rho \leq 1$ .

<sup>4</sup>Note that it is here that we need the time-varying assumption of the codes (compare also Section 5.5 and Figure 5.9).

7.9. RANDOM CODING ANALYSIS OF SEQUENTIAL DECODING 41

Now we substitute the metrics (see (7.11))

$$L_m(\tilde{\mathbf{x}}) = \sum_{r=0}^m \log \left( \frac{p(y_r|x_r)}{p(y_r)} \right) - k, \quad (7.74)$$

$$L_n(\tilde{\mathbf{x}}') = \sum_{r=0}^n \log \left( \frac{p(y_r|x'_r)}{p(y_r)} \right) - k, \quad (7.75)$$

into the expression (7.73) and use  $\alpha = \frac{1}{1+\rho}$ . This let's us rewrite the exponentials in (7.73) as

$$\begin{aligned} 2^{kn\rho} \int_{\underline{y}} \overline{p(\underline{y}|\underline{x}) \exp(-\alpha\rho L_m(\tilde{\mathbf{x}})) \left( \overline{\exp(\alpha L_n(\tilde{\mathbf{x}}'))} \right)^\rho} = \\ \begin{cases} 2^{-(m-n)E_c(\rho)-m(E_{ce}(\rho)-k\rho)} & \text{if } m \geq n \\ 2^{-(n-m)(E_e(\rho)-k\rho)-n(E_{ce}(\rho)-k\rho)} & \text{if } n \geq m. \end{cases} \end{aligned} \quad (7.76)$$

The exponents used above are given by

$$\begin{aligned} 2^{-E_c(\rho)} &= \int_v \sum_x q(x)p(v|x) \left( \frac{p(v|x)}{p(v)} 2^{-k} \right)^{-\frac{\rho}{1+\rho}} \\ &= 2^k \frac{\rho}{1+\rho} \int_v \sum_x q(x)p(v|x)^{\frac{1}{1+\rho}} p(v)^{\frac{\rho}{1+\rho}} \\ &\leq 2^k \frac{\rho}{1+\rho} \left( \int_v \left( \sum_x q(x)p(v|x)^{\frac{1}{1+\rho}} \right)^{1+\rho} \right)^{\frac{1}{1+\rho}} \\ &= 2^k \frac{\rho}{1+\rho} - \frac{1}{1+\rho} E_0(\rho, \mathbf{q}) = f_c, \end{aligned} \quad (7.77)$$

where we have used Hölder's inequality above (see Chapter 5, page 157) with  $\beta_i = \sum_x q(x)p(v|x)^{\frac{1}{1+\rho}}$ ,  $\gamma_i = p(v)^{\frac{\rho}{1+\rho}}$ , and  $\lambda = \frac{1}{1+\rho}$ , and, "magically" there appears the error exponent from Chapter 5, equation (5.51)! Analogously,

we also find

$$\begin{aligned}
2^{-(E_e(\rho)-k\rho)} &= 2^{k\rho} \int_v p(v) \left( \sum_{x'} q(x') \left( \frac{p(v|x')}{p(v)} 2^{-k} \right)^{\frac{1}{1+\rho}} \right)^\rho \\
&= 2^{k\frac{\rho^2}{1+\rho}} \int_v p(v)^{\frac{1}{1+\rho}} \left( \sum_{x'} q(x') p(v|x')^{\frac{1}{1+\rho}} \right)^\rho \\
&\leq 2^{k\frac{\rho^2}{1+\rho}} \left( \int_v \left( \sum_{x'} q(x') p(v|x')^{\frac{1}{1+\rho}} \right)^{1+\rho} \right)^{\frac{\rho}{1+\rho}} \\
&= 2^{k\frac{\rho^2}{1+\rho} - \frac{\rho}{1+\rho} E_0(\rho, \mathbf{q})} = f_e,
\end{aligned} \tag{7.78}$$

where  $\lambda = \frac{\rho}{1+\rho}$ . Finally we obtain

$$\begin{aligned}
2^{-(E_{ce}(\rho)-k\rho)} &= 2^{k\rho} \int_v \sum_x q(x) p(v|x) \left( \sum_{x'} q(x') \left( \frac{p(v|x')}{p(v|x)} \right)^{\frac{1}{1+\rho}} \right)^\rho \\
&= 2^{k\rho} \int_v \sum_x q(x) p(v|x)^{\frac{1}{1+\rho}} \left( \sum_{x'} q(x') p(v|x')^{\frac{1}{1+\rho}} \right)^\rho \\
&= 2^{k\rho - E_0(\rho, \mathbf{q})}.
\end{aligned} \tag{7.79}$$

Note now that, since  $1 = \frac{\rho}{1+\rho} + \frac{1}{1+\rho}$ , we have

$$2^{k\rho - E_0(\rho, \mathbf{q})} = 2^{k\frac{\rho^2}{1+\rho} - \frac{\rho}{1+\rho} E_0(\rho, \mathbf{q})} 2^{k\frac{\rho}{1+\rho} - \frac{1}{1+\rho} E_0(\rho, \mathbf{q})} = f_e f_c \tag{7.80}$$

where the two factors  $f_e$  and  $f_c$  are defined in (7.77) and (7.78).

With this we can rewrite (7.73) as

$$\text{Avg}\{\Pr(C_0 \geq N_c)\} \leq N_c^{-\rho} \sum_c p(c) \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} f_e^n f_c^m. \tag{7.81}$$

The double infinite sum in (7.81) converges if  $f_e, f_c < 1$  and, hence from (7.80), if  $\rho k < E_0(\rho, \mathbf{q})$  and we obtain

$$\text{Avg}\{\Pr(C_0 \geq N_c)\} \leq N_c^{-\rho} \frac{1}{(1-f_e)(1-f_c)}. \tag{7.82}$$

Similarly, it can be shown [39] that there exists a lower bound on the number of computations, given by

$$\text{Avg}\{\Pr(C_j \geq N_c)\} \geq N_c^{-\rho} (1 - o(N_c)) \tag{7.83}$$

Together, (7.82) and (7.83) characterize the computational behavior of sequential decoding. It is interesting to note that if  $\rho \leq 1$ , the expectation of (7.82) and (7.83), *i.e.*, the expected number of computations becomes unbounded, since

$$\sum_{N_c=1}^{\infty} N_c^{-\rho} \quad (7.84)$$

diverges for  $\rho \leq 1$  or  $k \geq R_0$ . Information theory therefore tells us that we cannot beat the capacity limit by using very powerful codes and resorting to sequential decoding, since, what happens is that as soon as the code rate reaches  $R_0$ , the expected number of computations per node tends to infinity. In effect our decoder fails through buffer overflow. This is why  $R_0$  is often also referred to as the *computational cutoff-rate*.

Further credence to  $R_0$  is given by the observation that rates  $R = R_0$  at bit error probabilities of  $P_b = 10^{-5} - 10^{-6}$  can be achieved with trellis codes. This observation was made by Wang and Costello [41], who constructed random trellis codes for 8-PSK and 16-QAM constellations which achieve  $R_0$  with constraint lengths of 15 and 16, *i.e.*, very realizable codes.

## 7.10 Some Final Remarks

As we have seen there are two broad classes of decoders, the depth-first and the breadth-first algorithms. Many attempts have been made at comparing the respective properties of these two basic approaches, for example [32], or, for convolutional codes, [39] is an excellent and inexhaustible source of information. Many of the random coding arguments in [39] for convolutional codes can be extended to trellis codes with little effort.

Where are we standing then? Sequential decoding has been popular in particular for relatively slow transmission speeds, since the buffer sizes can then be dimensioned such that buffer overflow is controllable. Sequential decoding, however, suffers from two major drawbacks. Firstly, it is a “sequential” algorithm, *i.e.*, modern pipelining and parallelizing is very difficult if not impossible to accomplish. Secondly, the metric used in sequential decoding contains the “bias” term accounting for the different paths lengths. This makes sequential decoding very channel dependent. Furthermore, this bias term may be prohibitively complex to calculate for other than straight channel coding applications (see e.g., [42]).

Breadth-first search algorithms, in particular the optimal Viterbi algorithm and the popular  $M$ -algorithm, do not suffer from the metric “bias”

term. These structures can also be parallelized much more readily which makes them good candidates for VLSI implementations. They are therefore very popular for high-speed transmission systems. The Viterbi algorithm can be implemented with a separate metric calculator for each state. More on the implementation aspects of parallel Viterbi decoder structures can be found in [11, 16, 13]. The Viterbi decoder has proven so successful in applications that it is the algorithm of choice for most applications of code decoding at present.

The  $M$ -algorithm can also be implemented exploiting inherent parallelism of the algorithm, and [35] discusses an interesting implementation which avoids the sorting of the paths associated with the basic algorithm. The  $M$ -algorithm has also been successfully applied to multi-user detection, a problem which can also be stated as a trellis (tree) search [43], and to the decoding of block codes.

In all likelihood the importance of all these decoding algorithms, with the likely exception of the Viterbi algorithm, will fade compared to that of the APP decoder in its different forms. The impact of iterative decoding of large error control codes (see chapters 9-11 on Turbo coding and related topics) has been so revolutionary as to push other strategies into obscurity.

## Appendix 6.A

In this appendix we calculate the Vector Euclidean distance for a specific set  $\mathcal{C}_p$  of retained paths. Noting that  $\underline{\delta}_n$  from (7.20) is a vector of Gaussian random variables (see also Section 2.6), we can easily write down its probability density function [44], viz.

$$p(\underline{\delta}_n) = \frac{1}{(2\pi)^{M/2} |\mathbf{R}|^{1/2}} \exp \left( -\frac{1}{2} (\underline{\mu} - \underline{\delta}_n)^T \mathbf{R}^{-1} (\underline{\mu} - \underline{\delta}_n) \right), \quad (7.85)$$

where  $\underline{\mu}$  is the vector of mean values given by  $\mu_i = d_i^2$ , and  $\mathbf{R}$  is the covariance matrix of the Gaussian random variables  $\underline{\delta}_n$  whose entries  $r_{ij} = E \left[ (\delta_n^{(i,c)} - \mu_i)(\delta_n^{(j,c)} - \mu_j) \right]$  can be evaluated as

$$r_{ij} = \begin{cases} 2N_0 (d_i^2 + d_j^2 - d_{ij}^2) & \text{if } i \neq j \\ 4N_0 d_i^2 & \text{if } i = j \end{cases} \quad (7.86)$$

and where

$$d_{ij}^2 = \left| \tilde{\underline{x}}^{(p_i)} - \tilde{\underline{x}}^{(p_j)} \right|^2 \quad \text{and} \quad d_i^2 = \left| \tilde{\underline{x}}^{(p_i)} - \tilde{\underline{x}}^{(c)} \right|. \quad (7.87)$$



The vector  $\underline{\mu}$  of mean values is given by  $\mu_i = d_i^2$ .

Now the probability of losing the correct path at time  $n$  can be calculated by

$$\Pr(\text{CPL}|\mathcal{C}_p) = \int_{\underline{\delta}_n \leq \underline{0}} p(\underline{\delta}_n) d\underline{\delta}_n. \quad (7.88)$$

Equation (7.88) is difficult to evaluate due to the correlation of the entries in  $\underline{\delta}_n$ , but one thing we know is that the area  $\underline{\delta}_n \leq \underline{0}$  of integration is convex. This allows us to place a hyperplane through the point closest to the center of the probability density function,  $\underline{\mu}$ , and overbound (7.88) by the probability that the noise carries the point  $\underline{\mu}$  across this hyperplane. This results in a simple one-dimensional integral, whose value is given by (compare also (2.14))

$$\Pr(\text{CPL}|\mathcal{C}_p) \leq Q \left( \sqrt{\frac{d_l^2}{2N_0}} \right), \quad (7.21)$$

where  $d_l^2$ , the Vector Euclidean distance, is given by

$$d_l^2 = 2N_0 \min_{\underline{y} \leq \underline{0}} (\underline{\mu} - \underline{y})^T \mathbf{R}^{-1} (\underline{\mu} - \underline{y}), \quad (7.89)$$

and  $\underline{y}$  is simply a dummy variable of minimization.

The problem of calculating (7.21) has now been transformed into the geometric problem of finding the point on the surface of the convex polytope  $\underline{y} \leq \underline{0}$  which is closest to  $\underline{\mu}$  using the distance measure of (7.89). This situation is illustrated in Figure 7.19 for a 2-dimensional scenario. The minimization in (7.89) is a constrained minimization of a quadratic form. Obviously, some of the constraints  $\underline{y} \leq \underline{0}$  will be met with equality. These constraints are called the *active constraints*, *i.e.*, if  $\underline{y} = (\underline{y}^{(a)}, \underline{y}^{(p)})^T$  is the partitioning of  $\underline{y}$  into active and passive components,  $\underline{y}^{(a)} = \underline{0}$ . This minimum is the point  $\underline{y}_0$  in Figure 7.19. The right hand side of Figure 7.19 also shows the geometric configuration when the decorrelating linear transformation  $\underline{\delta}'_n = \sqrt{\mathbf{R}^{-1}} \underline{\delta}_n$  is applied. The vector Euclidean distance (7.89) is invariant to such a transformation, but  $E \left[ (\delta_n'^{(i,c)} - \mu'_i)(\delta_n'^{(j,c)} - \mu'_j) \right] = \delta_{ij}$ , *i.e.*, the decorrelated metric differences are independent with unit variance each. Naturally we may work in either space. Since the random variables  $\underline{\delta}'_n$  are independent, equal-variance Gaussian, we know from basic communication theory (Chapter 2, [44]), that the probability that  $\underline{\mu}'$  is carried into

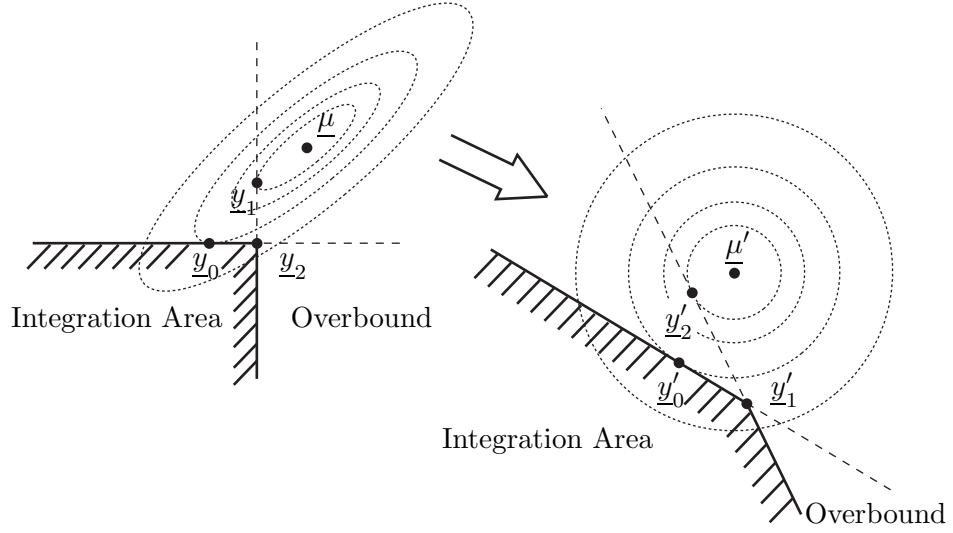


Figure 7.19: Illustration of the concept of the vector Euclidean distance with  $M = 2$ . The distance between  $\underline{y}_0$  and  $\underline{\mu}$  is  $d_l^2$ . The right hand side shows the space after decorrelation, and  $d_l^2$  equals the standard Euclidean distance between  $\underline{y}'_0$  and  $\underline{\mu}'$ .

the shaded region of integration can be overbounded by integrating over the halfplane not containing  $\underline{\mu}'$ , as illustrated in the figure. This leads to (7.21).

We now have to minimize

$$d_l^2 = 2N_0 \min_{\underline{y}^{(p)} \leq \underline{0}} \left( \begin{pmatrix} \underline{\mu}^{(p)} \\ \underline{\mu}^{(a)} \end{pmatrix} - \begin{pmatrix} \underline{y}^{(p)} \\ \underline{0} \end{pmatrix} \right)^T \begin{pmatrix} \mathbf{R}^{(pp)} & \mathbf{R}^{(pa)} \\ \mathbf{R}^{(ap)} & \mathbf{R}^{(aa)} \end{pmatrix}^{-1} \left( \begin{pmatrix} \underline{\mu}^{(p)} \\ \underline{\mu}^{(a)} \end{pmatrix} - \begin{pmatrix} \underline{y}^{(p)} \\ \underline{0} \end{pmatrix} \right), \quad (7.90)$$

where we have partitioned  $\underline{\mu}$  and  $\mathbf{R}$  analogously to  $\underline{y}$ . After some elementary operations we obtain

$$\underline{y}^{(p)} = \underline{\mu}^{(p)} + \left( \mathbf{X}^{(pp)} \right)^{-1} \mathbf{X}^{(pa)} \underline{\mu}^{(a)} \leq \underline{0}, \quad (7.91)$$

and

$$d_l^2 = 2N_0 \underline{\mu}^{(a)T} \left( \mathbf{X}^{(aa)} \right)^{-1} \underline{\mu}^{(a)}, \quad (7.92)$$

where<sup>5</sup>

$$\begin{aligned}
\mathbf{X}^{(pp)} &= \left[ \mathbf{R}^{(pp)} - \mathbf{R}^{(pa)} \left( \mathbf{R}^{(aa)} \right)^{-1} \mathbf{R}^{(ap)} \right]^{-1} \\
\mathbf{X}^{(aa)} &= \left( \mathbf{R}^{(aa)} \right)^{-1} \left[ \mathbf{I} + \mathbf{R}^{(ap)} \mathbf{X}^{(pp)} \mathbf{R}^{(pa)} \right] \\
\mathbf{X}^{(pa)} &= -\mathbf{X}^{(pp)} \mathbf{R}^{(pa)} \left( \mathbf{R}^{(aa)} \right)^{-1}.
\end{aligned} \tag{7.93}$$

We are now presented with the problem of finding the active components in order to evaluate (7.92). This is a combinatorial problem, *i.e.*, we must test all  $2^M - 1 = \sum_{i=1}^M \binom{M}{i}$  possible combinations of active components from the  $M$  entries in  $\underline{y}$  for compatibility with (7.91). This gives us the following procedure:

**Step 1:** Select all  $2^M - 1$  combinations of active components and set  $\underline{y}^{(a)} = \underline{0}$  for each.

**Step 2:** For each combination for which  $\underline{y}^{(p)} \leq \underline{0}$ , store the resulting  $d_i^2$  from (7.92) in a list.

**Step 3:** Select the smallest entry from the list in Step 2 as  $d_i^2$ .

As an example, consider again Figure 7.19. The  $2^2 - 1 = 3$  combinations correspond to the points  $\underline{y}_0, \underline{y}_1$  and  $\underline{y}_2$ . The point  $\underline{y}_1$  does not qualify, since it violates (7.91). The minimum is chosen between  $\underline{y}_0$  and  $\underline{y}_2$ . This process might be easier to visualize in the decorrelated space  $\underline{y}'$ , where all the distances are ordinary Euclidean distances, and the minimization becomes obvious.

One additional complication needs to be addressed at this point. The correlation matrix  $\mathbf{R}$  may be singular. This happens when one or more entries in  $\underline{y}$  are linearly dependent on the other entries. In the context of the

---

<sup>5</sup>These equations can readily be derived from the partitioned matrix inversion lemma:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} E & F \\ G & H \end{pmatrix},$$

where

$$E = (A - BD^{-1}C)^{-1}; F = -EBD^{-1}; G = -D^{-1}CE.$$

and

$$H = D^{-1} + D^{-1}CEBD^{-1}.$$

restriction  $\underline{y} \leq \underline{0}$ , we have redundant conditions. The problem, again, is that of finding the redundant entries which can be dropped from consideration. Fortunately, our combinatorial search helps us here. Since we are examining all combinations of possible active components, we may simply drop any dependent combinations which produce a singular  $\mathbf{R}^{(aa)}$  from further consideration without affecting  $d_l^2$ .

# Bibliography

- [1] J.B. Anderson, "Limited search trellis decoding of convolutional codes," *IEEE Trans. Inform. Theory*, vol. IT-35,, September 1989.
- [2] J.B. Anderson and S. Mohan, "Sequential coding algorithms: A survey and cost analysis," *IEEE Trans. Commun.*, vol. COM-32, No. 2, pp. 169–176, February 1984.
- [3] J.B. Anderson and S. Mohan, *Source and Channel Coding: An Algorithmic Approach*, Kluwer Academic Publishers, Boston, Mass., 1991.
- [4] T. Aulin, "Breadth First Maximum Likelihood Sequence Detection," *IEEE Trans. Commun.*, vol. COM-47, No. 2, pp. 208–216, February 1999.
- [5] T. Aulin, "Recovery Properties of the SA(B) Algorithm", Technical Report No. 105, Chalmers University of Technology, Sweden, February 1991.
- [6] T. Aulin, "Study of a new trellis decoding algorithm and its applications", Final Report, ESTEC Contract 6039/84/NL/DG, European Space Agency, Noordwijk, The Netherlands, December 1985.
- [7] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20,, pp. 284–287, March 1974.
- [8] K. Balachandran, "Design and performance of constant envelope and non-constant envelope digital phase modulation schemes", Ph.D. thesis, ECSE Dept. Rensselaer Polytechnic Institute, Troy, NY, February 1992.
- [9] R.E. Blahut, *Principles and Practice of Information Theory*, Addison-Wesley, Reading, Massachusetts, 1987.
- [10] P.R. Chevillat and D.J. Costello, Jr., "A multiple stack algorithm for erasurefree decoding of convolutional codes," *IEEE Trans. Commun.*, vol. COM-25, pp. 1460–1470, December 1977.

- [11] G.C. Clark and J.B. Cain, *Error-correction coding for digital communications*, Plenum Press, New York, 1983.
- [12] B. Classen, K. Blankenship, and V. Desai, "turbo decoding with the constant-log-MAP algorithm," *Proc. Second Int. Symp. Turbo Codes and Related Appl.*, (Brest, France), pp. 467–470, September 2000.
- [13] O.M. Collins, "The subtleties and intricacies of building a constraint length 15 convolutional decoder," *IEEE Trans. Commun.*, vol. COM-40, pp. 1810–1819, December 1992.
- [14] European Telecommunications Standards Institute, "Universal mobile telecommunications system (UMTS): Multiplexing and channel coding (FDD)," *3GPP TS 125.212 version 3.4.0*, pp. 14–20, September 23, 2000.
- [15] R.M. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. Inform. Theory*, vol. IT-9, pp. 64–74, April 1963.
- [16] G. Feygin and P.G. Gulak, "Architectural tradeoffs for survivor sequence memory management in Viterbi decoders," *IEEE Trans. Commun.*, vol. COM-41, pp. 425–429, March 1993.
- [17] G.D. Forney, Jr. "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, 1973.
- [18] G.D. Forney, "Maximum-likelihood sequence estimation of digital sequences in the presence of intersymbol interference," *IEEE Trans. Inform. Theory*, vol. IT-18, pp. 363–378, May 1972.
- [19] G.J. Foschini, "A reduced state variant of maximum likelihood sequence detection attaining optimum performance for high signal-to-noise ratios," *IEEE Trans. Inform. Theory*, vol. IT-23,, pp. 605–609, September 1977.
- [20] S. Lin and D.J. Costello, Jr., *Error Control Coding*, Prentice-Hall, Englewood Cliffs, 1983.
- [21] J.M. Geist, "An empirical comparison of two sequential decoding algorithms," *IEEE Trans. Commun.*, vol. COM-19, pp. 415–419, August 1971.
- [22] J.M. Geist, "Some properties of sequential decoding algorithms," *IEEE Trans. Inform. Theory*, vol. IT-19, pp. 519–526, July 1973.
- [23] W.J. Gross and P.G. Gulak, "Simplified map algorithm suitable for implementation of turbo decoders," *Electron. Lett.*, vol. 34, pp. 1577–1578, Aug. 6, 1998.

- [24] D. Haccoun and M.J. Ferguson, "Generalized stack algorithms for decoding convolutional codes," *IEEE Trans. Inform. Theory*, vol. IT-21,, pp. 638–651, November 1975.
- [25] F. Jelinek, "A fast sequential decoding algorithm using a stack," *IBM J. Res. Dev.*, Vol. 13, pp. 675–685, November 1969.
- [26] F. Jelinek and A.B. Anderson, "Instrumentable tree encoding of information sources" *IEEE Trans. Inform. Theory*, vol. IT-17, January 1971.
- [27] L. Ma, "Suboptimal decoding strategies", MSEE thesis, University of Texas at San Antonio, May 1996.
- [28] R.J. McEliece, "On the BCJR trellis for linear block codes," *IEEE Trans. Inform. Theory*, vol. IT-42, No. 4, pp. 1072–1092, July 1996.
- [29] J.L. Massey, "Variable-length codes and the Fano metric," *IEEE Trans. Inform. Theory*, vol. IT-18, pp. 196–198, January 1972.
- [30] H. Osthoff, J.B. Anderson, R. Johannesson, and C-F. Lin, "Systematic feed-forward convolutional encoders are better than other encoders with an  $M$ -algorithm decoder", *IEEE Trans. Inform. Theory*, vol. IT-44, No. 2, pp. 831–838, March 1998.
- [31] J.K. Omura, "On the Viterbi decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 177–179, January 1969.
- [32] G.J. Pottie and D.P. Taylor, "A comparison of reduced complexity decoding algorithms for trellis codes," *IEEE J. Select. Areas Commun.*, vol. SAC-7, No. 9, pp. 1369–1380, December 1989.
- [33] J.G. Proakis, *Digital Communications*, McGraw-Hill, Inc., 1989.
- [34] P. Robertson, P. Höher, and E. Villebrun, "Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding," *European Trans. on Telecommun.*, vol. 8,, pp. 119–125, March/April 1997.
- [35] S.J. Simmons, "A nonsorting VLSI structure for implementing the (M,L) Algorithm," *IEEE J. Select. Areas Commun.*, vol. SAC-6, pp. 538–546, April, 1988.
- [36] S.J. Simmons and P. Wittke, "Low complexity decoders for constant envelope digital modulation", *Conf. Rec.*, GlobeCom, Miami, Florida, pp. E7.7.1 – E7.7.5, November 1982.
- [37] S. Verdú, "Minimum probability of error for asynchronous Gaussian multiple-access channels," *IEEE Trans. Inform. Theory*, vol. IT-32, pp. 85-96, Jan. 1986.

- [38] A.J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260–269, April 1969.
- [39] A.J. Viterbi and J.K. Omura, *Principles of Digital Communication and Coding*, McGraw-Hill Inc. 1979.
- [40] J.M. Wozencraft and B. Reiffen, *Sequential Decoding*, M.I.T. Press, Cambridge, Mass., 1961.
- [41] F.-Q. Wang and D.J. Costello, Jr., "Probabilistic construction of large constraint length trellis codes for sequential decoding," *IEEE Trans. Commun.*, vol. COM-43, No. 9, pp. 2439–2448, Sept. 1995.
- [42] L. Wei, L.K. Rasmussen, and R. Wyrwas, "Near optimum tree-search detection schemes for bit-synchronous CDMA systems over Gaussian and two-path rayleigh fading channels," *IEEE Trans. Commun.*, vol. COM-45, No. 6, pp. 691–700, June 1997.
- [43] L. Wei and C. Schlegel, "Synchronous DS-SSMA with improved decorrelating decision-feedback multiuser detection," *IEEE Trans. Veh. Technol.*, vol. VT-43, No. 3, August 1994.
- [44] J.M. Wozencraft and I.M. Jacobs, *Principles of Communication Engineering*, Wiley, New York, 1965.
- [45] K.Sh. Zigangirov, "Some sequential decoding procedures," *Prob. Ped-erachi Inform.*, Vol. 2, pp. 13–25, 1966.
- [46] K.S. Zigangirov and V.D. Kolesnik, "List decoding of trellis codes", *Problems of Control and Information Theory*, No. 6, 1980.