CHAPTER FOUR

VITERBI DECODING OF LINEAR BLOCK CODES

4.1 CHAPTER OVERVIEW

THIS chapter sets out by describing the BCJR linear block code trellis construction method [2] for (n, k, d_{min}) linear block codes, where k is the message length, n is the code word length, and d_{min} is the minimum Hamming distance property (see *Section* 3.2.2.2) of the code. For illustrative purposes, it is used to created the trellis of a shortened binary Hamming $(5, 2, 3)$ block code (see *Section* 3.2.2.3.1). The next part of the chapter is concerned with the quantification of the complexity of a linear block code's trellis: Using a binary Hamming $(7, 4, 3)$ block code (see *Section* 3.2.2.3.1) as example, it firstly describes a simple, but tedious method whereby the state space profile and complexity [141–143] of a block code's trellis can be determined. Secondly, it discusses a trellis reduction method and illustrates the use thereof for a binary cyclic $(5, 3, 2)$ block code. In the final part of this chapter, the application of a block-wise VA [3] as an optimal ML trellis decoder for BCJR block code trellises is considered. Both hard and soft decision VA metric calculation approaches are addressed.

Although the block codes used as examples in this chapter are all binary linear block codes, the algorithms presented are readily applicable to non-binary codes, such as RS (see *Section* 3.2.2.3.3) and BCH (see *Section* 3.2.2.3.2) block codes. Unfortunately, due to the size and complexity of such codes' trellises, they do not lend themselves to be good examples whereby the algorithms presented here can be effectively demonstrated.

4.2 LINEAR BLOCK CODE TRELLIS CONSTRUCTION

In principle, every linear block code has a unique trellis description: By creating a set of parallel trellis paths, one for each code word, a simple albeit inefficient trellis is generated. In [2] *Bahl et al.* presents a more elegant approach to derive an efficient trellis structure from the linear block code's parity check matrix (see *Section* 3.2.2.1). The following two subsections are devoted to a description of this trellis construction technique.

4.2.1 CONSTRUCTING AN UNEXPURGATED LINEAR BLOCK CODE TRELLIS

As previously stated, the BCJR trellis construction method for a (n, k, d_{min}) linear block code with code word symbols from Galois field $GF\left(2^{\xi}\right)$, requires the code's parity check matrix, given by:

$$H_{BC} = [\overline{h}_0 \; \overline{h}_1 \; ... \; \overline{h}_{n-1}] \tag{4.1}$$

In *Eq.* (4.1) $\overline{h}_i$, with $i = 1, 2, ..., n$, is the i^{th} column of the parity check matrix, containing $(n-k)$ elements from the Galois field $GF\left(2^{\xi}\right)$. The trellis construction technique is based on the fact that the i^{th} $(n-k)$-tuple syndrome vector $\overline{\$}_m^i$ [47] for the m^{th} n-tuple valid code word vector $\overline{c}_m = \{c_{m,0}, c_{m,1}, ..., c_{m,n-1}\}$ can be calculated using the following recursion formula [2, 87]:

$$\overline{\$}_m^i = \overline{\$}_m^{i-1} + c_{m,i-1}.\overline{h}_{i-1} \tag{4.2}$$

where $c_{m,i}$ is the i^{th} code word symbol contained in the vector $\overline{c}_m$. The initial condition is $\overline{\$}_m^0 = \overline{0}$. Note that addition and multiplication are carried out symbol-wise in $GF\left(2^{\xi}\right)$. The block code trellis, which is a compact method to represent all $2^{k.\xi}$ code words in the code, consists of $(n+1)$ sets of nodes (states), each set containing $2^{\xi.(n-k)}$ nodes. By interconnecting the nodes with branches in a topology uniquely defined by H_{BC}, the trellis is constructed. For the purpose of this discussion, the sets of nodes are indexed by a parameter i, with $i = 0, 1, 2, ..., n$. Nodes in set i are indexed by a parameter l, with $l = 0, 1, 2, ..., 2^{\xi.(n-k)} - 1$. Therefore, the l^{th} node in the i^{th} set has an index (l, i). The branches emanating from the nodes in the trellis are indexed by the parameter j. Each branch in the trellis has an associate branch weight or decoder input branch vector, as well as a decoder output branch vector. For example, the branch weight vector and decoder output branch vector of the j^{th} branch leaving node (l, i) are $\overline{u}_{i,l}^{(j)}$ and $\overline{o}_{i,l}^{(j)}$, respectively. The trellis construction procedure is described below:

1. Set $i = 0$, where i is the trellis depth counter.
2. At a depth of $i = 0$, only node $(0, 0)$ has 2^{ξ} emanating branches. From any node (l, i), with $i > 0$, that has incoming branches, 2^{ξ} branches flow forth. The destination nodes in set $i+1$ of the branches leaving any node (l, i) are determined as follows:
   (a) Determine a length-$(n-k)$ vector $\overline{\epsilon}_i$, which contains the l^{th} possible combination of $(n-k)$ elements from $GF\left(2^{\xi}\right)$.
   (b) Let the $GF\left(2^{\xi}\right)$ symbol associated with the j^{th} branch be denoted by b_j. With $j = 0, 1, 2, ..., 2^{\xi} - 1$, proceed as follows:
      i. Compute the $(n-k)$-tuple $GF\left(2^{\xi}\right)$ vector $\overline{q}_{i,j} = (\overline{h}_i^T.b_j) + \overline{\epsilon}_i$, where addition and multiplication are carried out symbol-wise in $GF\left(2^{\xi}\right)$. The vector $\overline{h}_i^T$ represents the transpose of $\overline{h}_i$.
      ii. The destination node in set $i+1$ is node $(z_{i,j}, i+1)$, where the vector $\overline{q}_{i,j}$ contains the $z_{i,j}^{\text{th}}$ possible combination of $(n-k)$ elements from $GF\left(2^{\xi}\right)$.
      iii. If bit-wise comparison has to be used in the metric calculations of the trellis decoder, as is always the case with binary block codes, the branch weight vector assigned to the j^{th} branch is given by value $\overline{u}_{i,l}^{(j)} = \overline{w}_j$, where $\overline{w}_j$ is the sequence of binary bits representing the $GF\left(2^{\xi}\right)$ symbol b_j. When, for non-binary block codes such as RS and BCH block codes, the code word symbols are used directly in the decoding process, i.e. symbol-wise comparison during the branch metric calculations, the branch weight vector consists of a single element, namely $\overline{w}_j = b_j$.

iv. The decoder output branch vector of the j^{th} branch is equal to the branch weight vector, i.e. $\overline{o}_{i,l}^{(j)} = \overline{u}_{i,l}^{(j)}$.

(c) Repeat steps (a) and (b) for every l, where node (l, i) has one or more incoming branches.

3. Repeat step 2 for $i = 1, 2, .., n-1$.

Following the procedure outlined above, a trellis with more paths than code words in the code is created. Such a trellis is called an *unexpurgated* [3] or *unconstrained* [87] block code trellis. Shown in *Fig.* 4.1 is the unexpurgated trellis, obtained using the BCJR trellis construction method, for a binary Hamming $(5, 2, 3)$ block code (see *Section* 3.2.2.3.1) (constructed by shortening the Hamming $(7, 4, 3)$ block code, defined by the generator matrix of *Eq.* (4.4)) with the following parity check matrix:

$$H_{BC} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

In this figure, the VA decoder (see *Section* 4.4) output and input sequences associated with each branch of the trellis are indicated by the *Decoder Output Sequence* $\overline{o}_{i,l}^{(j)}$ / *Decoder Input Sequence* $\overline{u}_{i,l}^{(j)}$ labels.

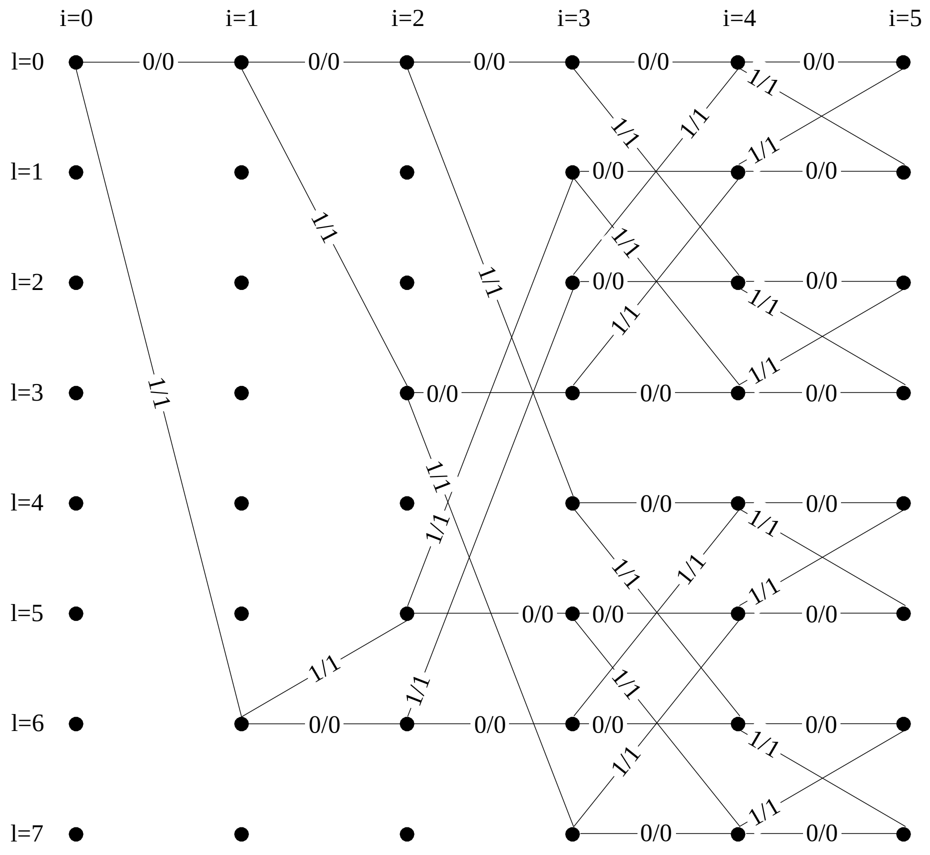

Figure 4.1: Unexpurgated Trellis of the Shortened Binary Hamming $(5, 2, 3)$ Code

4.2.2 EXPURGATING A BLOCK CODE TRELLIS

Removal of all non-code word representing paths in the unexpurgated trellis of a block code, with code word symbols from $GF(2^{\xi})$, a process referred to as *trellis expurgation* [3], involves discarding all paths that do not end in node $(0, n)$. The resultant trellis is called an *expurgated* or *constrained* block code trellis. In this trellis, only $2^{\xi \cdot k}$ unique paths, representing the valid code words in the block code, are retained.

Although the expurgation process seems simple, it usually involves tedious back-tracing through the trellis. A simple technique, applicable to the BCJR trellis structures of unextend systematic linear block codes, presented by *Staphorst*, *Büttner* and *Linde* in [52] for binary block codes and in [55] for non-binary block codes, involves removing all branches from nodes in set $i - 1$ entering the node (l, i) for $i = k + 1, k + 2, k + 3, .., n$ and $l = 2^{\xi \cdot (n-i)}, 2^{\xi \cdot (n-i)} + 1, .., 2^{\xi \cdot (n-k)} - 1$. *Fig.* 4.2 shows the expurgated trellis obtained for the binary Hamming $(5, 2, 3)$ code, defined by *Eq.* (4.3), after this path removal algorithm has been applied to the unexpurgated trellis of *Fig.* 4.1.

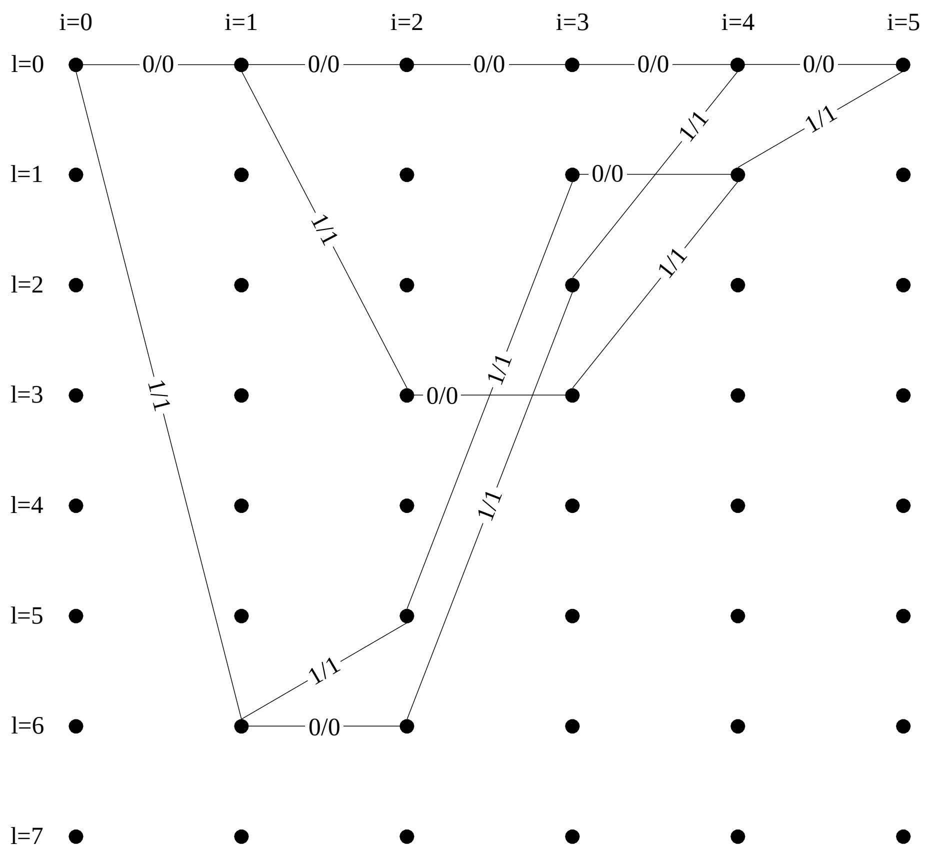

Figure 4.2: Expurgated Trellis of the Shortened Binary Hamming $(5, 2, 3)$ Code

It should be noted that the VA can be just as successfully applied to an unexpurgated trellis, as long as it is configured to only select paths starting in state $(0, 0)$ and ending in state $(0, n)$ (see *Section* 4.4). Moreover, identical BER performances are obtained by applying the VA to unexpurgated or expurgated BCJR trellis structures, irrespective of the channel conditions. However, this will result in unnecessary computations, larger decoder memory requirements and overall increased system complexity. For example, a VA applied to the unexpurgated trellis of *Fig.* 4.1 must perform 46 branch metric calculations for the branches connecting its $AN(C) = 31$ active nodes, whereas a VA running on the expurgated trellis of *Fig.* 4.2 only needs to calculate 16 branch metrics for the branches connecting its $AN(C) = 14$ active nodes.

4.3 COMPLEXITY OF LINEAR BLOCK CODE TRELLISES

For most block codes of practical interest, such as the extended binary BCH $(32, 21, 6)$ block code, used extensively in several paging protocols, the code rate $R_c = k/n$ is greater than or equal to 0.5, while the number of parity symbols $(n-k)$ is at least 10. Given the extensive use of such block codes in wireless communication systems, these codes lend themselves to be prime candidates for trellis decoding. Unfortunately, the complexity of an (n, k, d_{min}) block code's trellis grows exponentially with $\min\{k, (n-k)\}$ [3]. For example, the expurgated trellis of the previously mentioned $(32, 21, 6)$ extended BCH block code has a staggering 14972 branches [144]. Thus, the cost effectiveness of block code trellis decoders for such codes is questionable. The following subsections firstly illustrates a procedure whereby the complexity of a block code's trellis can be calculated, followed by a simple block code trellis reduction method.

4.3.1 TRELLIS COMPLEXITY CALCULATION

A simple state space complexity calculation method [141–143, 145, 146] for an (n, k, d_{min}) block code with code and message word symbols from $GF\left(2^{\xi}\right)$, defined by generator matrix G_{BC} (see *Section* 3.2.2.1), is detailed in this subsection. As is shown in this subsection, the state space complexity of a block code gives a good indication of the complexity of the block code's trellis. The state space complexity calculation procedure is illustrated for a binary Hamming $(7, 4, 3)$ block code (see *Section* 3.2.2.3.1) with the following generator matrix:

$$G_{BC} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{4.4}$$

Due to the considerable number of lengthy intermediate results obtained during the calculation of a block code's state space complexity, some of these results are omitted in the following discussion for the Hamming $(7, 4, 3)$ block code.

The first step in the estimation of the block code's state space complexity is to determine all possible code words obtainable from G_{BC}. In general, the number of unique code words that can be generated by an (n, k, d_{min}) block code's generator matrix G_{BC} is $2^{\xi.k}$. For the Hamming $(7, 4, 3)$ block code, $2^4 = 16$ unique code words exist. Let C denote this set of code words, with $\overline{c}_m = \{c_{m,0}, c_{m,1}, ..., c_{m,(n-1)}\}$ the m^{th} code word in the set.

Next, the block code's dimension and inverse dimension distributions [145,146], denoted by $\Lambda(C) = \{\Lambda_0, \Lambda_1, \Lambda_2, ..., \Lambda_n\}$ and $\Lambda^{-1}(C) = \{\Lambda_0^{-1}, \Lambda_1^{-1}, \Lambda_2^{-1}, ..., \Lambda_n^{-1}\}$, respectively, are determined. The procedure is as follows [145, 146]:

1. Define a master indexing set $\Omega(C) = \{0, 1, 2, ..., n-1\}$. The master indexing set for the Hamming $(7, 4, 3)$ block code is $\Omega(C) = \{0, 1, 2, 3, 4, 5, 6\}$.
2. Let i denote the dimension distribution element index. With $i = 0, 1, 2, ..., n$, proceed as follows:
   (a) For $j = 0, 1, 2, ..., \binom{n}{i}$, complete the following steps:
      i. Select an indexing set $\Phi^j(C) = \{\Phi_1^j, \Phi_2^j, ..., \Phi_i^j\}$, such that it is a subset from $\Omega(C)$, containing i elements that have not been selected for any previous value of j. The size of $\Phi^j(C)$ is therefore $|\Phi^j(C)| = i$. For example, if $i = 3$, one possible j^{th} indexing set

for the Hamming $(7,4,3)$ block code is:

$$\Phi^j(C) = \{1,3,6\} \tag{4.5}$$

ii. Next, a subset of code words is created from C by manipulating the symbols of each code word, pointed to by the indexing elements in $\Phi^j(C)$: For each code word in C, the code word symbols at positions $\Phi^j_1, \Phi^j_2, ..., \Phi^j_i$ are replaced with zeros, resulting in $2^{\xi.k}$ new, but necessarily unique, code words. By only keeping the unique code words, a new set of code words, denoted by $\vartheta\left(\Phi^j(C)\right)$, is created. For example, as specified by the chosen indexing subset given in *Eq.* (4.5), set $c_{m,1} = c_{m,3} = c_{m,6} = 0$ for $m = 1,2,...,16$. This results in 16 new, but not necessarily unique code words. From these 16 it can be shown that there are only 8 distinct code words in this new set, given by:

$$\vartheta\left(\Phi^j(C)\right) = \begin{bmatrix} \{0,0,0,0,0,0,0\} \\ \{1,0,0,0,1,0,0\} \\ \{1,0,1,0,0,0,0\} \\ \{1,0,1,0,1,1,0\} \\ \{0,0,1,0,1,0,0\} \\ \{1,0,0,0,0,1,0\} \\ \{0,0,1,0,0,1,0\} \\ \{0,0,0,0,1,1,0\} \end{bmatrix} \tag{4.6}$$

iii. Determine $k[\vartheta\left(\Phi^j(C)\right)]$, a parameter which can be interpreted as the effective number of message symbols that is required to generate the number of code words contained in $\vartheta\left(\Phi^j(C)\right)$ [145, 146]:

$$k[\vartheta\left(\Phi^j(C)\right)] = \log_{2^\xi}\left\{\text{Number of code words in } \vartheta\left(\Phi^j(C)\right)\right\} \tag{4.7}$$

From *Eq.* (4.6) it is clear that $k[\vartheta\left(\Phi^j(C)\right)] = 3$ for the indexing subset defined in *Eq.* (4.5) for the Hamming $(7,4,3)$ block code.

iv. The next step is to define an inverse indexing subset $\Psi^j(C) = \{\Psi^j_1, \Psi^j_2, ..., \Psi^j_{n-i}\}$ which contains all the elements from $\Omega(C)$, except those already in $\Phi^j(C)$. The inverse indexing subset associated with the indexing subset given in *Eq.* (4.5), is given by:

$$\Psi^j(C) = \{0,2,4,5\} \tag{4.8}$$

v. The inverse indexing subset is now used to create a new set of code words $\vartheta\left(\Psi^j(C)\right)$, containing all the code words from C that have zero code word symbols at the positions indicated by the indexing elements in $\Psi^j(C)$. Using the selected inverse indexing set given in *Eq.* (4.8) produces the following set of new unique code words:

$$\vartheta\left(\Psi^j(C)\right) = \begin{bmatrix} \{0,0,0,0,0,0,0\} \\ \{0,1,0,1,0,0,1\} \end{bmatrix} \tag{4.9}$$

vi. From the new set of code words $\vartheta\left(\Psi^j(C)\right)$, determine $k[\vartheta\left(\Psi^j(C)\right)]$, defined as:

$$k[\vartheta\left(\Psi^j(C)\right)] = \log_{2^\xi}\left\{\text{Number of code words in } \vartheta\left(\Psi^j(C)\right)\right\} \tag{4.10}$$

From *Eq.* (4.9) it follows that $k[\vartheta\left(\Psi^j(C)\right)] = 1$ for the Hamming $(7,4,3)$ block code.

(b) The i^{th} elements of the code's dimension and inverse dimension distributions are given by $\Lambda_i = \max\{k[\vartheta\left(\Phi^j(C)\right)]\}$ and $\Lambda_i^{-1} = \max\{k[\vartheta\left(\Psi^j(C)\right)]\}$ for $j = 0,1,2,...,\binom{n}{i}$, respectively.

If the procedure outlined above is completed for the Hamming $(7,4,3)$ block code, the dimension and inverse dimension distributions obtained will be:

$$\Lambda(C) = \{0,1,2,3,4,4,4,4\} \tag{4.11}$$

and:

$$\Lambda^{-1}(C) = \{0,0,0,0,1,2,3,4\} \tag{4.12}$$

respectively. With an (n,k,d_{min}) linear block code's dimension and inverse dimension distributions known, its state space profile is determined as follows [145, 146]:

$$\begin{aligned} SSP(C) &= \{S_0, S_1, ..., S_n\} \\ &= \{(\Lambda_0 - \Lambda_0^{-1}), (\Lambda_1 - \Lambda_1^{-1}), ..., (\Lambda_n - \Lambda_n^{-1})\} \end{aligned} \tag{4.13}$$

Using *Eq.* (4.11) and *Eq.* (4.12), the state space profile for the Hamming $(7,4,3)$ block code under investigation follows readily:

$$SSP(C) = \{0,1,2,3,3,2,1,0\} \tag{4.14}$$

At a depth of i into the code's expurgated trellis, the number of active nodes (states), i.e nodes having incoming and/or outgoing branches, is $2^{\xi.S_i}$. Thus, the total number of active nodes in the block code's expurgated trellis is [59, 145, 146]:

$$AN(C) = \sum_{i=0}^{n} 2^{\xi.S_i} \tag{4.15}$$

The final important parameter to calculate is the state space complexity of the linear block code [145, 146]:

$$SSC(C) = \max\{S_i\} \qquad \text{for } i = 0,1,2,...,n \tag{4.16}$$

From the Hamming $(7,4,3)$ block code's state space profile, given in *Eq.* (4.14), it follows that the number of active nodes and the state space complexity of the code are $AN(C) = 30$ and $SSC(C) = 3$, respectively.

With the number of active states in the trellis known, the next step is to determine an upper bound on the number of branches $NB(C)$ present in the trellis: For a trellis depth of $i < n$ into the unexpurgated BCJR trellis of an (n,k,d_{min}) linear block code with code and message word symbols from $GF\left(2^{\xi}\right)$, the number of branches exiting an active node is 2^{ξ}. Since all paths end in node $(0,n)$, it follows that $AN(C) - 1$ nodes in an expurgated trellis have departing branches. Hence, the number of branches present in an expurgated trellis is upper bounded as follows:

$$NB(C) \leq (AN(C) - 1).2^{\xi} \tag{4.17}$$

The state space complexity is a key measure [59, 141, 142, 147] of the trellis complexity, and thus also the decoding complexity of a specific linear block code. A lower value of the state space complexity results in a less complex trellis structure, leading to faster decoding and less complex trellis decoder structures.

4.3.2 REDUCING TRELLIS COMPLEXITY

It is a known fact [93, 148] that the swapping of columns of a generator matrix of an (n,k,d_{min}) block code, with code word symbols from $GF(2^{\xi})$, results in an equivalent block code. Although the resultant block code has different code words, it has the same error correcting capabilities as the original code, since the minimum Hamming distance d_{min} (see *Section* 3.2.2.2) remains unchanged. However, it can be shown [149, 150] that the state space dimension of the new code differs from that of the original code. Therefore, swapping columns in a code's generator matrix influences the complexity of the resultant block code's trellis. As will be illustrated by the following trellis reduction example, it is possible to obtain an equivalent code with a lower complexity trellis [149, 150].

The binary cyclic $(5, 3, 2)$ block code (see *Section* 3.2.2.2) to be used as an example in this section, is defined by the following generator matrix:

$$G_{BC} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{4.18}$$

From this generator matrix, the following parity check matrix can be determined:

$$H_{BC} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{4.19}$$

Fig. 4.3 depicts the expurgated trellis obtained for this parity check matrix using the algorithms

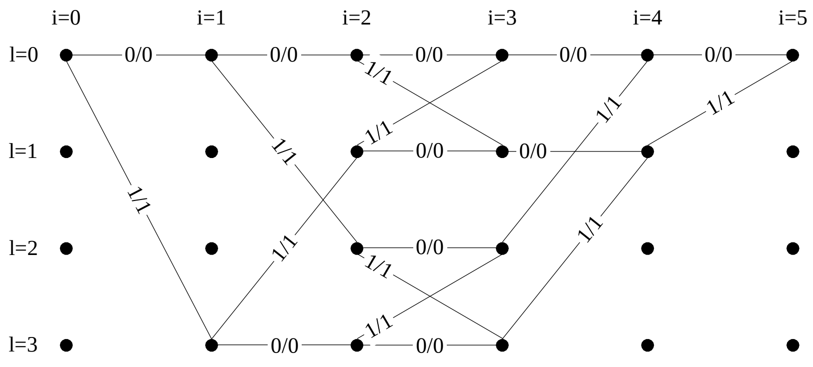

Figure 4.3: Expurgated Trellis of the Original Binary Cyclic $(5, 3, 2)$ Code

presented in *Section* 4.2.1 and *Section* 4.2.2. Following the procedures described in *Section* 4.3.1, it can be shown that the state space profile of the binary cyclic $(5, 3, 2)$ block code is given by:

$$SSP(C) = \{\ 0 \quad 1 \quad 2 \quad 2 \quad 1 \quad 0\ \} \tag{4.20}$$

The number of active nodes in the binary cyclic $(5, 3, 2)$ block code's expurgated trellis is:

$$AN(C) = \sum_{i=0}^{5} 2^{\xi . S_i} = 1 + 2 + 4 + 4 + 2 + 1 = 14 \tag{4.21}$$

Comparing these results with *Fig.* 4.3, it is clear that the calculated and actual number of active nodes in the expurgated trellis correspond. Furthermore, according to *Eq.* (4.17) the number of branches in the expurgated trellis is upper bounded by $NB(C) \leq (14 - 1).2 = 26$. To be precise, $NB(C) = 20$.

In order to find an equivalent code with the least complex trellis, the state space complexity of every possible swapped column permutation of the generator matrix has to be considered [143, 149, 150]. Thus, the state space profile has to be calculated for $n! = 5! = 120$ different permutations of the code's generator matrix. The following data was obtained from this analysis:

- There exists 8 generator matrices with $AN(C) = 10$ and $SSC(C) = 1$.
- There exists 32 generator matrices with $AN(C) = 12$ and $SSC(C) = 2$.
- There exists 80 generator matrices with $AN(C) = 14$ and $SSC(C) = 2$.

From these results it is apparent that by selecting one of the 8 generator matrices which has 10 active nodes, a minimally complex trellis is obtained. For example, one of these 8 minimal trellis generator matrices is given by:

$$G_{BC} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{4.22}$$

with the following associated parity check matrix:

$$H_{BC} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{4.23}$$

The state space profile for this equivalent code is given by:

$$SSP(C) = \{ \ 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \ \} \tag{4.24}$$

Fig. 4.4 shows the expurgated trellis of this equivalent code. Comparing *Fig.* 4.3 and *Fig.* 4.4, the

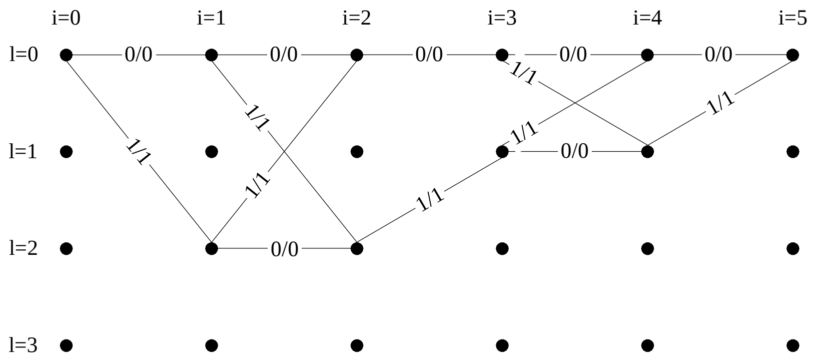

Figure 4.4: Optimally Reduced Expurgated Trellis of the Equivalent Binary Cyclic $(5, 3, 2)$ Code

reduction in trellis complexity is clearly visible. By applying the VA to the reduced trellis, decoding time as well as decoder complexity will be noticeably reduced.

4.4 TRELLIS DECODING OF LINEAR BLOCK CODES USING THE VITERBI ALGORITHM

Algebraic ML decoding of an (n, k, d_{min}) linear block code, with code word symbols from $GF\left(2^{\xi}\right)$, involves the comparison of the received code word with each of the $2^{\xi \cdot k}$ valid code words, selecting the one which proves to be the closest as the most likely transmitted one. This process can be time consuming and complex, especially for codes where both n and k are large.

Conversely, the application of the VA as ML block code decoder results in the most likely path in the code's trellis, describing the received code word, to be chosen, discarding several unlikely code words along the way before even comparing them with the received one. Depending on the parameters n and k, this decoding approach may greatly reduce decoding time and complexity.

4.4.1 THE BLOCK-WISE VITERBI ALGORITHM FOR LINEAR BLOCK CODE TRELLISES

Let $\overline{c}_m$ and $\overline{y}_m$ denote the m^{th} transmitted and the received code words, respectively, associated with the message word $\overline{d}_m$. ML decoding entails obtaining the best estimate of $\overline{c}_m$, denoted by $\hat{\overline{c}}_m$, by comparing the received code word $\overline{y}_m$ with all valid code words. Since there exists a one-to-one mapping between $\overline{d}_m$ and $\overline{c}_m$, the data vector associated with $\hat{\overline{c}}_m$, denoted by $\hat{\overline{d}}_m$, shall be identical to $\overline{d}_m$, if and only if $\hat{\overline{c}}_m = \overline{c}_m$. If $\hat{\overline{c}}_m \neq \overline{c}_m$, a decoding error is incurred. Thus, the decoding rule whereby an ML decoder chooses the optimal estimate $\hat{\overline{c}}_m$, given $\overline{y}_m$, is by minimising the probability of a decoding error. Assuming equiprobable message words, the probability of decoding error is minimised if the overall *log-likelihood function* $\ln\left(\text{Prob.}\left(\overline{y}_m|\hat{\overline{c}}_m\right)\right)$ is maximised, where $\text{Prob.}\left(\overline{y}_m|\hat{\overline{c}}_m\right)$ represents the conditional probability of receiving the code word $\overline{y}_m$, given that $\hat{\overline{c}}_m$ was the transmitted code word. The VA accomplishes this by, moving from trellis depth 0 to n, systematically discarding paths that do not maximise the overall log-likelihood function, $\ln\left[\text{Prob.}\left(\overline{y}_m|\hat{\overline{c}}_m\right)\right]$.

Since every branch in an expurgated trellis represents a symbol in a valid code word, finding the most likely path involves comparing each incoming symbol (hard decision) or sample (soft decision) of the received vector to the corresponding branch symbols. For non-binary codes, this can be done on a symbol level or on a bit level, depending on the application. For example, when an RS block code's code word symbols are transmitted as equivalent bit streams, bit level comparison will be used by the VA trellis decoder. In this dissertation only bit level comparison is considered.

Before describing the VA, as applied to block code trellises, the concept of a *metric* is introduced: In the broad sense, a metric is a measure of similarity between two entities. True to this general definition, VA decoding entails the calculation of branch and path metrics (based on the computation of log-likelihood values) in order to indicate the degree of similarity between the received bits/samples of the m^{th} transmitted code word and the valid code word paths in the code's trellis. Assuming that the $n.\xi$ received bits/samples representing the n transmitted $GF\left(2^{\xi}\right)$ code word symbols are statistically independent, the mathematical definition of the branch metric, indicating the similarity between m^{th} received code word's i^{th} set of ξ received and demodulated bits/samples (denoted by $\overline{y}_{m,i}$) and the branch weight vector of the j^{th} branch leaving node l at a trellis depth of i (denoted by $\overline{u}_{i,l}^{(j)}$) is given by:

$$\begin{aligned} BM_{m,i,l}^{(j)} &= \ln\left[\text{Prob.}\left(\overline{y}_{m,i}|\overline{u}_{i,l}^{(j)}\right)\right] \\ &= \ln\left[\prod_{a=0}^{\xi-1}\text{Prob.}\left(y_{m,i,a}|u_{i,l,a}^{(j)}\right)\right] \\ &= \sum_{a=0}^{\xi-1}\ln\left[\text{Prob.}\left(y_{m,i,a}|u_{i,l,a}^{(j)}\right)\right] \end{aligned} \tag{4.25}$$

where $y_{m,i,a}$ denotes the a^{th} element of the i^{th} ξ-tuple vector $\overline{y}_{m,i}$ of demodulated bits/samples, and $u_{i,l,a}^{(j)}$ the a^{th} branch weight vector element of the j^{th} branch leaving node (l,i). Several hard and soft decision methods whereby $BM_{m,i,l}^{(j)}$ can be calculated are discussed in *Section* 4.4.2.

As input samples (soft decision decoding) or bits (hard decision decoding) are received, path metrics are computed, indicating the probability that a certain path through the trellis, starting at node $(0,0)$ and ending in node $(0,n)$, represents the transmitted code word. The VA for linear block codes therefore operates in a block-wise fashion, whereas a more traditional sliding window approach is used for convolutional code decoding [106].

Since there are $2^{\xi\cdot(n-k)}$ states in the trellis, there will be $2^{\xi\cdot(n-k)}$ path metrics. At a decoding depth of i into the trellis for the m^{th} received code word, the path metric of the survivor path ending in node (l,i), is denoted by $PM^{sur}_{m,i,l}$. The cumulative metric [47] of the j^{th} branch leaving node l at a trellis depth of i, is given by:

$$CM^{(j)}_{m,i,l} = PM^{sur}_{m,i,l} + BM^{(j)}_{m,i,l} \tag{4.26}$$

It should be noted that, although according to the general modus operandi of ML decoders, as described at the beginning of this subsection, the probability of a decoding error is minimised if the largest path metric is chosen as the most optimal at each trellis depth, there exists certain conditions for which *Eq.* (4.25) can be simplified to such an extent that the *smallest* path metric dictates a minimum probability of decoding error. More attention is given to such scenarios in the following two subsections.

With the aforementioned definitions in mind, the forward tracing procedure performed by the VA in order to determine the survivor path metrics for the active nodes in the trellis, is as follows:

1. Set the survivor path metric $PM^{sur}_{m,0,0}$, representing the non-existent survivor path ending in node $(0,0)$, to $PM^{sur}_{m,0,0}=0$.
2. Set the survivor path metrics $PM^{sur}_{m,0,l}$, with $l=1,2,..,2^{\xi\cdot(n-k)}-1$, of the non-existent survivor paths entering the other nodes at this depth to either $-\infty$ (if an increase in path metric dictates a decrease in the probability of decoding error), or ∞ (if a decrease in path metric dictates a decrease in the probability of decoding error).
3. At decoding time instance m, $n.\xi$ bits or samples are received, which are contained within the vector $\overline{y}_m$. Using this vector, the forward tracing procedure that determines the survivor path metrics associated with the $2^{\xi\cdot(n-k)}$ paths through the trellis are as follows:
   (a) Begin decoding at a trellis depth of $i=0$.
   (b) For every node (l,i) at a trellis depth of i that has exiting branches, proceed as follows:
      i. Calculate the cumulative metric $CM^{(j)}_{m,i,l}$ for the j^{th} branch leaving node (l,i) using *Eq.* (4.26).
      ii. For every node $(b,i+1)$ at a trellis depth of $i+1$ that has incoming branches, set the survivor path metric $PM^{sur}_{m,i+1,b}$ equal to the most optimal cumulative metric calculated for these incoming branches. Discard all the branches entering node $(b,i+1)$, except the branch associated with the most optimal cumulative metric.
   (c) Repeat step (b) for $i=1,2,..,n-1$.

When the procedure as outlined above is followed, only one complete path, starting at node $(0,0)$ and ending in node $(0,n)$, will survive. Hence, the m^{th} most likely transmitted code word can be found by noting the entries in the output branch vectors of the branches in this surviving path.

After acquiring the most likely transmitted code word using the VA, a conventional algebraic decoding method can be employed to obtain the associated k-symbol message word. In the case of a systematic block code, the associated message word is simply the k systematic symbols of the VA decoded n-symbol code word.

4.4.2 HARD VERSUS SOFT DECISION DECODING

If demodulated data samples are processed by the VA immediately after demodulation, without any previous decisions having been made, the decoding process is called *soft decision decoding*. Alternatively, if the demodulated samples are classified as 1s and 0s prior to VA decoding, it is referred to as *hard decision decoding*. It has been shown [47] that soft decision decoding shows an asymptotic BER performance gain of approximately 2.1 dB over hard decision decoding for binary transmitted data in AWGN channel (see *Section* 2.2) conditions. This can be attributed to the fact that the entropy of an analogue information source, such as the output of a demodulator, is decreased by hard decision processing, i.e. 1-bit analogue to digital conversion [47]. Thus, less information is utilised during hard decision metric calculations than with soft decision metric calculations, resulting in poorer BER performances. Unfortunately, soft decision decoding's gain comes at the price of higher implementation hardware complexity, i.e. the need for high precision *Analogue-to-Digital Converter*s (ADC) and DSPs. The following subsections briefly outline branch metric calculation methods for both hard and soft decision decoding.

4.4.2.1 HARD DECISION DECODING

Two hard decision decoding branch metric calculation methods are described in this subsection. The first, which employs the *Binary Symmetric Channel* (BSC) [47] crossover probability P_{BSC}, calculates the branch metric as follows:

$$BM_{m,i,l}^{(j)} = \sum_{a=0}^{\xi-1} \begin{cases} \ln(1-P_{BSC}) & \text{if } y_{m,i,a} = u_{i,l,a}^{(j)} \\ \ln(P_{BSC}) & \text{if } y_{m,i,a} \neq u_{i,l,a}^{(j)} \end{cases} \tag{4.27}$$

The second method, which is simpler to implement in hardware, uses the Hamming distance (see *Section* 3.2.2.2) between the input bit sequence and branch weight vector to calculate the branch metric:

$$BM_{m,i,l}^{(j)} = d_H\left(\overline{y}_{m,i}, \overline{u}_{i,l}^{(j)}\right) \tag{4.28}$$

where the Hamming distance is defined as the number of bits that $\overline{y}_{m,i}$ and $\overline{u}_{i,l}^{(j)}$ differ, i.e.:

$$d_H\left(\overline{y}_{m,i}, \overline{u}_{i,l}^{(j)}\right) = \sum_{a=0}^{\xi-1} \begin{cases} 0 & \text{if } y_{m,i,a} = u_{i,l,a}^{(j)} \\ 1 & \text{if } y_{m,i,a} \neq u_{i,l,a}^{(j)} \end{cases} \tag{4.29}$$

It should be apparent that the size of a path metric and the probability of decoding error are directly proportional for the latter calculation method, whereas an indirect proportionality exists for the branch metric calculation method of *Eq.* (4.27).

4.4.2.2 SOFT DECISION DECODING

Assume that the linear block code encoder output bits of the m^{th} transmitted code were modulated and transmitted over a mobile radio channel exhibiting both slow flat (see *Section* 2.5.1.1) fading (see *Section* 2.5.1.1) and AWGN channel (see *Section* 2.2) effects. The demodulator output is described by *Eq.* (3.57), having the PDF given in *Eq.* (3.58). Using *Eq.* (3.58), the probability Prob. $\left(y_{m,i,a}|u_{i,l,a}^{(j)}\right)$ can easily be calculated [47], an exercise which is not repeated here. Employing this probability in *Eq.* (4.25), removing all factors and terms that are common to all cumulative metrics calculated at a trellis depth of i, it follows that the general expression for the VA branch metric calculation for soft decision decoding with fading amplitude CSI, is given by [60]:

$$BM_{m,i,l}^{(j)} = \sum_{a=0}^{\xi-1} \left(y_{m,i,a} - \hat{\overline{\alpha}}_{m,i,a}.u_{i,l,a}^{(j)}\right)^2 \tag{4.30}$$

where $\hat{\bar{\alpha}}_{m,i,a}$ is an estimate of the average fading amplitude associated with $y_{m,i,a}$. Once again the size of a branch metric is directly proportional to the probability of decoding error. This metric calculation method can also be readily applied to hard decision decoding. Completing the square and further removing common factors and terms, *Eq.* (4.30) simplifies to:

$$BM^{(j)}_{m,i,l} = \sum_{a=0}^{\xi-1} \left(2.y_{m,i,a}.\hat{\bar{\alpha}}_{m,i,a}.u^{(j)}_{i,l,a} - \left(\hat{\bar{\alpha}}_{m,i,a}.u^{(j)}_{i,l,a} \right)^2 \right) \tag{4.31}$$

Yet another simplification can be performed if antipodal code bit representation is used, i.e. the branch weight vectors have elements from the alphabet $\{-1, 1\}$, since $\left(u^{(j)}_{i,l,a}\right)^2 = 1$ for all branches at a trellis depth of i, resulting in:

$$BM^{(j)}_{m,i,l} = \sum_{a=0}^{\xi-1} y_{m,i,a}.\hat{\bar{\alpha}}_{m,i,a}.u^{(j)}_{i,l,a} \tag{4.32}$$

It should be noted that *Eq.* (4.32) now describes a branch metric calculation method where an increase in the path metric dictates a decrease in the probability of decoding error.

In the event that the receiver achieves perfect coherent demodulation, but no fading amplitude information is available (or only AWGN channel effects are present), the VA employs any of the above soft decision branch metric calculation methods with $\hat{\bar{\alpha}}_{m,i,a} = 1$.

4.5 ANALYTICAL BIT-ERROR-RATE PERFORMANCE EVALUATION OF VITERBI DECODED LINEAR BLOCK CODES

The following subsections briefly presents BER performance upper bounds for VA or classic ML decoded binary linear block codes in AWGN (see *Section* 2.2) and flat (see *Section* 2.5.1.1) Rayleigh (see *Section* 2.5.2.1) fading channel conditions, assuming they are employed in narrowband QPSK systems (see *Section* 5.2).

4.5.1 AWGN CHANNEL BIT-ERROR-RATE PERFORMANCE UPPER BOUND

Assuming coherent demodulation and ML decoding, the BER performance for a narrowband QPSK system employing an (n, k, d_{min}) binary linear block code, operating in AWGN channel conditions (see *Section* 2.2), is upper bounded as follows [63, 100, 151]::

$$P_b(e) \leq \sum_{h=d_{\min}}^{n} \left[\left(\sum_{w=1}^{k} \frac{w}{k} A_{w,h} \right) Q\left(\sqrt{2.h.R_c \frac{E_b}{N_0}} \right) \right] \tag{4.33}$$

where $R_c = k/n$, E_b is the energy per message word bit, N_0 is the single sided PSD of the AWGN, $Q(\cdot)$ represents the Q-function, and $\{A_{w,h}\}$ are the coefficients of the block code's IOWEF, as defined by *Eq.* (3.21) in *Section* 3.2.2.2. Defining the following constant:

$$B_h = \sum_{w=1}^{k} \frac{w}{k} A_{w,h} \tag{4.34}$$

the upper bound of *Eq.* (4.33) can be further simplified:

$$P_b(e) \leq \sum_{h=d_{\min}}^{n} B_h.Q\left(\sqrt{2.h.R_c\frac{E_b}{N_0}}\right) \tag{4.35}$$

Note that $k.B_h$ represents the total Hamming weight of all message words that yield code words of Hamming weight h. *Eq.* (4.35) suggests that there are two possible methods to decrease the BER of the block coded QPSK system:

- The BER of the system can be reduced by increasing d_{min}. This is the classic approach taken by block code designers. In addition, a well designed block code keeps the number of minimum distance code words as small as possible.
- By reducing B_h of the most significant terms in *Eq.* (4.35), which corresponds to the lowest weight code words, the BER can be reduced. This is the technique employed by PCC coding schemes [66].

4.5.2 SLOW RAYLEIGH FLAT FADING CHANNEL BIT-ERROR-RATE PERFORMANCE UPPER BOUND

Let the code word bits generated by an (n, k, d_{min}) binary linear block code be modulated by a QPSK transmitter and transmitted over flat Rayleigh fading channel (see *Section* 2.5.1.1 and *Section* 2.5.2.1). For simplicity, it is assumed that each transmitted code word bit experiences independent Rayleigh fading. This assumption is valid for a fully interleaved (see *Section* 3.2.3) flat Rayleigh fading channel where the fading amplitudes of the respective transmitted code word bits are IID stochastic variables. If the QPSK receiver achieves perfect coherent demodulation, ML decoding results in the following BER bound, conditioned on $\gamma_{c,m}^h$, the SNR per code word for the m^{th} code word of Hamming weight h [151]:

$$P_b\left(e|\gamma_{c,m}^h\right) \leq \sum_{h=d_{\min}}^{n}\left[\left(\sum_{w}^{k}\frac{w}{k}A_{w,h}\right)Q\left(\sqrt{2.R_c.\gamma_{c,m}^h}\right)\right] \tag{4.36}$$

where $R_c = k/n$, $Q(\cdot)$ represents the Q-function, and $\{A_{w,h}\}$ are the coefficients of the linear block code's IOWEF (see *Section* 3.2.2.2). The SNR per code word in this equation, is given by [151]:

$$\gamma_{c,m}^h = \sum_{i=0}^{h-1}\frac{E_b}{N_0}\overline{\alpha}_{m,i}^2 = \sum_{i=0}^{h-1}\gamma_{b,m,i} \tag{4.37}$$

where $\overline{\alpha}_{m,i}$ is the average fading amplitude experienced by the i^{th} received code word bit in the m^{th} code word. Furthermore, it is assumed that the fading amplitude is approximately constant over a code word symbol period under slow fading conditions. Since $\overline{\alpha}_{m,i}$ is a stochastic variable with a Rayleigh PDF, it follows that $\gamma_{b,m,i} = E_b/N_0.\overline{\alpha}_{m,i}^2$ has a chi-squared PDF with two degrees of freedom, given by *Eq.* (5.22) in *Section* 5.2.4.2 [47, 152]. Thus, it can be shown that the PDF of $\gamma_{c,m}^h$, denoted by $\rho\left(\gamma_{c,m}^h\right)$, resembles *Eq.* (5.45) in *Section* 5.3.4. Finally, obtaining an averaged BER performance upper bound necessitates averaging $P_b\left(e|\gamma_{c,m}^h\right)$ over the PDF of $\gamma_{c,m}^h$ [151, 152]:

$$\begin{aligned} P_b(e) &\leq \int_0^\infty P_b\left(e|\gamma_{c,m}^h\right)\rho\left(\gamma_{c,m}^h\right)d\gamma_{c,m}^h \\ &= \frac{1}{2}\sum_{h=d_{min}}^{n}\left(\sum_{w=1}^{k}\frac{w}{h}A_{w,h}\right)\left[\frac{1}{1+R_c.\overline{E}_b/N_0}\right]^h = \frac{1}{2}\sum_{h=d_{min}}^{n}B_h\left[\frac{1}{1+R_c.\overline{E}_b/N_0}\right]^h \end{aligned} \tag{4.38}$$

where the constant of *Eq.* (4.34) has been employed and $\overline{E}_b$ represents the average energy per message word bit. By assuming a fully interleaved channel, this bound will be loose when used to study block coded QPSK systems functioning in Rayleigh fading channels exhibiting a high level of correlation, i.e. slow fading (see *Section* 2.5.1.2) and no channel interleaving. A tighter upper bound, which takes the Doppler spread (see *Section* 2.4.3.3) of the slow fading channel into account, is presented in [69, 153]. However, this bound is more cumbersome to calculate than *Eq.* (4.38).

4.6 CONCLUDING REMARKS

This chapter considered the VA decoding of binary and non-binary linear block codes. Firstly, a detailed description was given of the BCJR trellis construction technique for linear block codes, followed by novel trellis complexity calculation and reduction techniques. Next followed an in-depth discussion on the block-wise VA as applied to BCJR trellises. Several hard and soft decision branch metric calculation methods, employing CSI, were considered. Lastly, BER bounds were presented for AWGN and slow Rayleigh flat fading channels. The main contributions of this chapter are the following:

1. *Section* 4.2.1 presents ”The Complete Idiots Guide to...” for the construction of unexpurgated BCJR linear block code trellises. The simple step-by-step algorithm presented is applicable to both binary and non-binary linear block codes.

2. A novel BCJR trellis expurgation scheme is presented in *Section* 4.2.2. This algorithm can be used to prune both binary and non-binary BCJR linear block code trellises to contain only paths representing valid code words.

3. *Section* 4.3.1 describes a method whereby the trellis complexity of BCJR trellises for binary and non-binary linear block codes can be calculated (or at least closely estimated). This is then followed in *Section* 4.3.2 by a rudimentary technique that can be used to reduce the complexity of the BCJR trellis structures for binary linear block codes.

4. In *Section* 4.4.1 the classic block-wise VA, applicable to BCJR linear block code trellises, is presented. *Section* 4.4.2.2 gives attention to the inclusion of fading amplitude CSI into the branch metric calculations used during soft decision decoding.