The Dual SISO Algorithm

Jeremy Thorpe

September 6, 2001

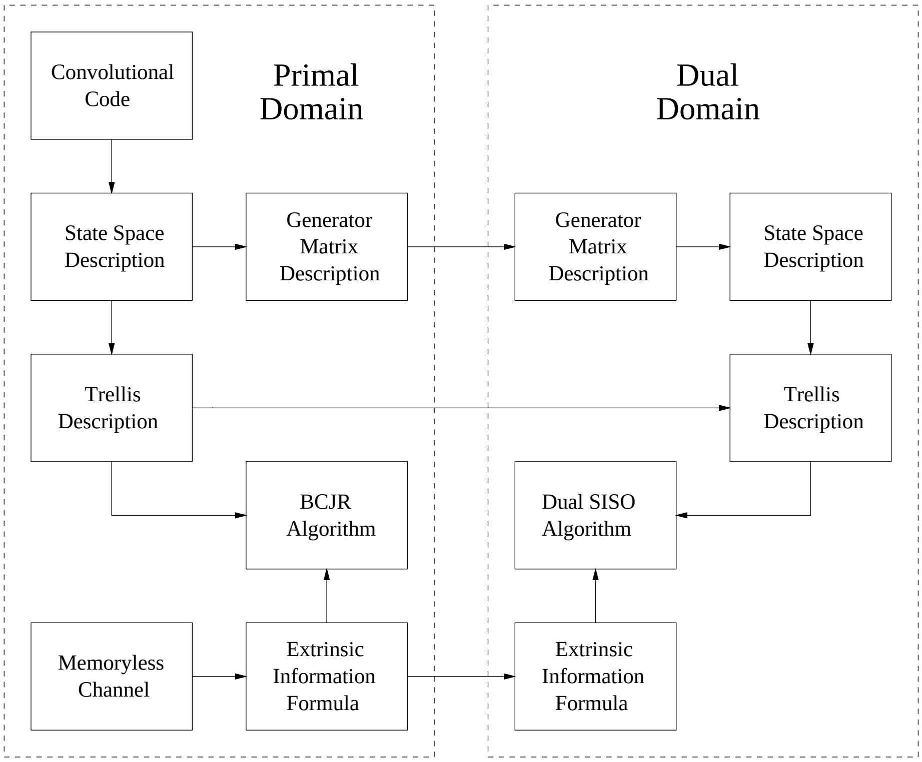
Contents

1 Abstract

This paper is intended to be a tutorial on the fundamentals and implementation of the Dual-domain Soft-in-soft-out (DSISO) convolutional code decoding algorithm. We first investigate dual codes in the setting of infinite length codes, and then extend this duality to finite-length codes, which usually involves truncation and termination. Then, we give a description of the trellis for convolutional codes and derive the useful properties which allow the fast algorithms to be applied. We derive rules for calculation of a posteriori probability and extrinsic information using codes in both domains and show how these quantities can be efficiently calculated. [Finally, we give an example of a turbo-coding system which can use and profit from the use of the DSISO, and discuss some of the potential pitfalls associated with DSISO and not the primal domain SISO algorithm.]

2 Introduction

The following figure shows the framework we will use to describe the Dual SISO algorithm. In the primal domain, we give the general view of the BCJR algorithm as obtained from the trellis description of the code combined with the equation for bitwise probability. In the dual domain, we first define the dual code algebraically via the rational generator matrix $\mathcal{G}(D)$, then define the trellis of the convolutional code, taking care to manage the truncation and termination of each code. From the probability equation involving summing over codewords in the primal domain, we obtain a formula involving summing over codewords in the dual domain. Finally, we derive the equations of the DSISO algorithm.



Schematic for understanding the DSISO algorithm

In section 3, we give two equivalent descriptions of a convolutional code, beginning with the state-space description and deriving the generator matrix description. In section 4, we define the dual of a convolutional code and prove some of their useful properties. In section 5, we derive a trellis description of a convolutional code based on the state-space description and describe the duality between truncation and termination of primal and dual codes. In section 6, we derive a formula to calculate a posteriori probability given Y in terms of the primal and dual codes. In section 7, we combine the trellis description with the a posteriori probability calculation rule to define the BCJR algorithm in the primal domain and Dual SISO (DSISO) algorithm in the dual domain. In section 8, we discuss the complexity of each algorithm and explore the merits of implementing DSISO in the logarithm domain.

3 Convolutional Codes

In this section, we give definitions of state space and generator matrix descriptions of convolutional code generators, and then define the more abstract term "convolutional code" and its intrinsic memory. We end with an example which illustrates each of these concepts.

3.1 State Space Encoders

Whereas a linear block code encoder maps k_0-dimensional "message" vectors (over the finite field $\mathbb{F}$) into n_0-dimensional "code" vectors, a linear (time-invariant) convolutional encoder maps a sequence of k_0-dimensional input vectors to a sequence of n_0-dimensional output vectors over $\mathbb{F}$.

From system theory, we know that any realizable LTI system (henceforth we restrict our attention to realizable systems) has a state-space realization with finite number of storage elements m, the *degree* of the realization. This realization has the form:

$$\underline{s}(i+1) = \underline{s}(i)\mathcal{A} + \underline{u}(i)\mathcal{B} \tag{3.1}$$

$$\underline{x}(i) = \underline{s}(i)\mathcal{C} + \underline{u}(i)\mathcal{D} \tag{3.2}$$

where the input $u(i)$ has dimension k_0, the output $x(i)$ has dimension n_0 and $s(i)$, the state vector, has dimension m, and the matricies $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{D}$ have the appropriate dimensions and elements from the field $\mathbb{F}$.

or, in D-transform domain,

$$D^{-1}\underline{S}(D) = \underline{S}(D)\mathcal{A} + \underline{U}(D)\mathcal{B} \tag{3.3}$$

$$\underline{X}(D) = \underline{S}(D)\mathcal{C} + \underline{U}(D)\mathcal{D} \tag{3.4}$$

where the D-transform is defined as:

$$\mathsf{D}\{\underline{x}\} = \underline{X}(D) = \sum_{i \geq 0} \underline{x}(i) D^i \tag{3.5}$$

The inverse D-transform is defined in the obvious way:

$$\mathsf{D}^{-1}\{X(D)\}(i) = \text{coeff}(D^i)\underline{X}(D) \tag{3.6}$$

3.2 Generator Matrix Description

The generator-matrix description is obtained from the state space description by solving for $\underline{X}(D)$ in terms of $\underline{U}(D)$.

We note that these equations can be solved for $\underline{X}(D)$ in terms of $\underline{U}(D)$, only if we can take the inverse of a polynomial in D, with coefficients in $\mathbb{F}$, which is shown in mceliece to be satisfied in the field of rational functions of D over $\mathbb{F}$. Continuing algebraically:

$$\underline{X}(D) = \underline{U}(D)\mathcal{D} + \mathcal{B}(D^{-1}\mathcal{I}_m - \mathcal{A})^{-1}\mathcal{C} \equiv \underline{U}(D)\mathcal{G}(D) \tag{3.7}$$

In general the matrix $\mathcal{G}(D)$ is a matrix of rational functions of D, meaning each entry is a rational function of D.

As a result of 3.1, for any $\mathbb{C}$ generated by $\mathcal{G}(D)$, we may pre-multiply the generator $\mathcal{G}(D)$ by diag$\{d_1, d_2, ..., d_{k_0}\}$, where d_i is the least common denominator of the i^{th} row of $\mathcal{G}(D)$ to yield a polynomial generator matrix of $\mathbb{C}$.

Here, we define the *Forney Indicies* and degree of a polynomial generator matrix. The Forney Index of row i, f_i of polynomial $\mathcal{G}(D)$ is equal to the maximum degree of D in $\mathcal{G}_{i,:}(D)$.

There is a simple realization of a polynomial generator matrix which uses exactly f_i delay elements per row. Two examples are shown in 4.2. For this reason, the *degree* of $\mathcal{G}(D)$ is the sum of its Forney indices.

3.3 Encoder-independent Description

Having derived the generator-matrix description of a convolutional code, we can, by direct analogy with block codes, define the convolutional code $\mathbb{C}$ generated by $\mathcal{G}(D)$ to be the set of codewords which can be generated, in other words, the rowspace of its generator matrix $\mathcal{G}(D)$

In particular, this means that we may pre-multiply $\mathcal{G}(D)$ by any non-singular $k_0 \times k_0$ matrix to obtain a new generator matrix for $\mathbb{C}$:

Theorem 3.1 *If a (n, k) convolutional code $\mathbb{C}$ is generated by generator matrix $\mathcal{G}(D)$, and $\mathcal{T}(D)$ is a non-singular $k_0 \times k_0$ rational matrix, then $\mathcal{T}(D)\mathcal{G}(D)$ also generates $\mathbb{C}$.*

Proof since $\mathcal{T}(D)$ is non-singular, $\mathcal{T}(D)\mathcal{G}(D)$ has the same rowspace as $\mathcal{G}(D)$. $\blacksquare$

We define the memory of a convolutional code to be equal to the minimum degree of all of its encoders. We also term each encoder of minimum degree a *minimal encoder*.

Minimal degree encoders are desireable from an engineering perpective because they require the least possible number of storage element. Furthermore, we will se in section 8 that minimal degree code representations are also very desireable for BCJR and DSISO decoders since their required memory is exponential in the degree of the representation.

It is shown in mceliece that there is a minimal and polynomial generator for any convolutional code, a fact which we will use several times throughout this paper.

3.4 Example

We introduce an example of a convolutional code which will be used throughout the rest of this tutorial. It is the same example as in [2]. We begin with the (cannonical controller form) state-space description.

$$D^{-1}\underline{S}(D) = \underline{S}(D)\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix} + \underline{U}(D)\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \tag{3.8}$$

$$\underline{X}(D) = \underline{S}(D)\begin{bmatrix}1 & 1 & 1\\ 1 & 0 & 0\end{bmatrix} + \underline{U}(D)\begin{bmatrix}1 & 0 & 1\\ 0 & 1 & 1\end{bmatrix} \tag{3.9}$$

By 3.7, we can immediately find a generator matrix, which turns out in this case to be a polynomail generator matrix:

$$\underline{X}(D) = \underline{U}(D)\mathcal{G}(D), \mathcal{G}(D) = \begin{bmatrix}1+D & D & 1+D\\ D & 1 & 1\end{bmatrix} \tag{3.10}$$

And we note that both are encoders of $\mathbb{C} = \text{rowspace}\mathcal{G}(D)$. In addition, each encoder has degree 2, which is the memory of the code. $\mathcal{G}_{poly}(D)$ is therefore a canonical generator matrix.

4 Dual Convolutional Codes

4.1 Duality Theorem

For convolutional code $\mathbb{C}$, we define the dual convolutional code $\mathbb{C}'$ to the the set of all codewords such that the inner product $x \cdot w' = 0$ for all $x \in \mathbb{C}, w \in \mathbb{C}'$ for all pairs of words for which the sum converges.

There is a theorem, due to Forney in 1970 which states that an (n_0, k, m_0) convolutional code has a dual $(n_0, n_0 - k_0, m)$ convolutional code. The proof can be found in [4].

We take the following definition of dual code (which differs from the definitions given in [2], and [4], but is the definition of *dual code* in [5]). Two codewords x and w are orthogonal if and only if the correlation of the two signals is 0:

$$\sum_{j=-\infty}^{\infty} \underline{x}(j)\underline{w}^T(j) = 0 \tag{4.1}$$

However, for convolutional codes, we know that a time-shifted version of a codeword is also a codeword, so that the correlation *function* $\phi xw(i)$ is also identically zero:

$$\phi_{xw}(i) = \sum_{j=-\infty}^{\infty} \underline{x}(i+j)\underline{w}^T(j) = 0 \tag{4.2}$$

Transforming into frequency domain:

$$\Phi_{XW}(D) = \sum_{i=-\infty}^{\infty} D^i \sum_{j=-\infty}^{\infty} \underline{x}(i+j)\underline{w}^T(j) = 0 \tag{4.3}$$

changing the order of summation:

$$= \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} D^i \underline{x}(k)\underline{w}^T(j) = 0 \tag{4.4}$$

letting $k = i + j$ we have:

$$= \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} D^{k-j} \underline{x}(k)\underline{w}^T(j) = 0 \tag{4.5}$$

changing back the order of summation and separating j and k terms:

$$= \sum_{k=-\infty}^{\infty} D^k \underline{x}(k) \sum_{j=-\infty}^{\infty} D^{-j} \underline{w}^T(j) = 0 \tag{4.6}$$

$$= \underline{X}(D)\underline{W}^T(D^{-1}) = 0 \tag{4.7}$$

Returning to Convolutional codes, for given $\mathcal{G}(D)$, if we can find $\mathcal{H}(D)$ such that

$$\mathcal{G}(D)\mathcal{H}(D^{-1}) = \mathbf{0} \tag{4.8}$$

then the encoded sequences $\underline{X}(D) = \underline{U}(D)\mathcal{G}(D)$ and $\underline{W}(D) = \underline{V}(D)\mathcal{H}(D)$ will be orthogonal:

$$\Phi_{XW}(D) = \underline{X}(D)\underline{W}^T(-D) = \underline{U}(D)\mathcal{G}(D)(\underline{V}(D^{-1})\mathcal{H}(D^{-1}))^T \tag{4.9}$$

$$= \underline{U}(D)\mathcal{G}(D)\mathcal{H}^T(D^{-1})\underline{U}^T(D^{-1}) \tag{4.10}$$

$$= \underline{V}(D)\mathbf{0}\underline{U}^T(D^{-1}) = 0 \tag{4.11}$$

We note that the above equation is true only if the original sum converges. For our purposes, we will apply this equation to *polynomial* codewords $\underline{X}(D)$ and $\underline{W}(D)$, avoiding potential problems with ill-defined sums.

Finally, we note that such an $\mathcal{H}(D)$ can be found in the following way. Construct a systematic Generator Matrix $\mathcal{G}_{sys}(D) = \mathcal{T}(D)\mathcal{G}(D) = [\mathcal{I}_{k_0}|\mathcal{A}(D)]$ by gauss-elimination. Then, the matrix $\mathcal{H}_{sys}(D) = [\mathcal{A}^T(D^{-1})|\mathcal{I}_{n_0-k_0}]$ has the desired property.

It should be noted that the definition we have taken in 4.8 is by no means universal. It seems the usual definition holds that $\mathcal{H}(D)$ is dual to $\mathcal{G}(D)$ iff $\mathcal{G}(D)\mathcal{H}^T(D) = 0$. However, in that case there is the awkwardness that one of the codebooks must be time-reversed to achieve duality in the block-code sense, at which point the two definitions become identical.

4.2 Example

We illustrate one method of finding an inverse using the same example as in the previous section:

First, we generate a systematic matrix by pre-multiplying by a $k \times k$ non-singular matrix:

$$\mathcal{G}_{sys}(D) = \frac{1}{1+D+D^2}\begin{bmatrix} 1 & D \\ D & 1+D \end{bmatrix}\mathcal{G}_{poly} \tag{4.12}$$

$$= \begin{bmatrix} 1 & 0 & \frac{1}{1+D+D^2} \\ 0 & 1 & \frac{1+D^2}{1+D+D^2} \end{bmatrix} \tag{4.13}$$

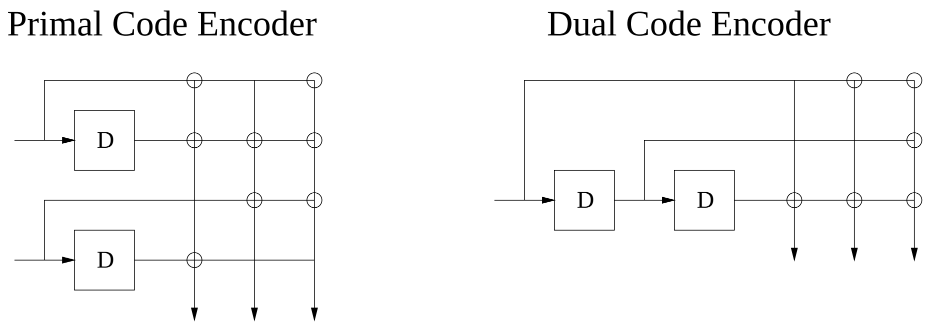
From which we can immediately obtain H_{sys}:

$$\mathcal{H}_{sys} = \begin{bmatrix} \frac{1}{1+D^{-1}+D^{-2}} & \frac{1+D^{-2}}{1+D^{-1}+D^{-2}} & 1 \end{bmatrix} \tag{4.14}$$

We first convert this matrix, which is rational in D^{-1}, into one which is rational in D by multiplying each row by D^{f_i} where f_i is the Forney Index of that row (in the example, there is only one row and its Forney Index is 2). From the Rational Parity-check matrix, we can obtain a polynomial one by multiplying by the least common denominator:

$$\mathcal{H}_{poly} = D^2(1+D^{-1}+D^{-2}) \cdot \mathcal{H}_{sys} = \begin{bmatrix} D^2 & 1+D^2 & 1+D+D^2 \end{bmatrix} \tag{4.15}$$

The following diagram shows the obvious realization of the polynomial generators of the primal and dual codes.



Realizations of generators for primal and dual codes

5 Trellis Description of Convolutional Codes

In this section, we define the trellis corresponding to a particular state space encoder. Furthermore, we show the duality between truncation and termination of codes in the primal and dual domain.

5.1 Trellis Description

We derive the Trellis description of a convolutional code directly from the state-space description. A trellis is a directed graph with the property that there are no directed loops (that is, starting at any one state, it is impossible to get back to the same state taking steps only in forward direction).

The trellis for a convolutional code is obtained by writing a node for each m-dimensional state vector s at each time t. Then, for each state $(s(i), i)$ and each input $u(i)$, draw an edge from $(s(i), i)$ to $(s(i+1), i+1)$ where $s(i+1) = s(i)\mathcal{A} + u(i)\mathcal{B}$.

5.2 Trellis Truncation/Termination

There are two important ways to begin and end a finite-length convolutional code. They are *truncation* and *termination.* Truncation on the left [right] means that the encoder can start [end] in any state, while termination on the left [right] means that the encoder must start [end] in the all-zeros state.

We will define a finite-length convolutional codebook which may be either right-terminated or right-truncated, and either left-terminated or left-truncated. For simplicity, we will define it through its canonical generator $\mathcal{G}(D)$, which is a polynomial generator matrix with degree equal to the memory of the code. Suppose a finite length codeword $\underline{X}_F$ is to be the 0^{th} through $(L-1)^{th}$ vectors of a codeword $underline X = \underline{U}(D)\mathcal{G}(D)$ from , and thus the overall length of the codeword is $n = n_0 \cdot L$.

Without loss of generality, we may suppose that $\underline{U}(D)$ is polynomial and has initial component in row i, D^{-f_i} and final component D^{L-1}, since all other components do not affect symbols included in $\underline{X}_F$. If, in addition to this restriction, $\underline{U}(D)$ has initial component D^0 for all i, we say that this codeword is *left terminated*, since the state of the encoder at time 0 would be the all-zeros state. Similarly, if the final component of $\underline{U}(D)$ is D^{L-1-f_i} for each i, then the codeword is said to be *right terminated*.

left-terminated codewords have the property that $\underline{x}(i) = 0$, for $i < 0$, and right-terminated codewords have the property that $\underline{x}(i) = 0$, for $i >= t$.

It can be verified that the dimension of the left and right terminated code is:

$$k_0 \cdot L - \sum_{i=1}^{k_0} f_i = k_0 \cdot L - m \tag{5.1}$$

By similar calculations, the dimension of a left and right truncated codeword is $k_0 \cdot L + m$ and that of a left-terminated right-truncated or a left-truncated right-terminated one is just $k_0 \cdot L$.

One might guess that a *terminated* code has a *truncated* dual, since in that case the dimension of the primal, $k_0 \cdot L - m$ plus the dimension of the dual, $(n_0 - k_0) + m$ exactly equals the length $n_0 \cdot L$ in that case. Indeed, if all pairs of codewords can be shown to be orthogonal, then these codes are dual since the dimensions sum to n. We will show that this is the case.

Theorem 5.1 *If $X(D) = U(D)G(D)$, and $W(D) = U(D)G(D)$, and if at least one of $\underline{x}_T(i)$ or $\underline{w}_T(i)$ is* right-terminated, *and at least one is* left terminated *then:*

$$\sum_{i=0}^{L-1} \underline{x}(i)\underline{w}(i) = 0 \tag{5.2}$$

Proof Since $\underline{X} \in \mathbb{C}$ and $\underline{W} \in \mathbb{C}'$ and each has a finite number of nonzero elements (since each is the product of two polynomials), we have:

$$\sum_{i=-\infty}^{\infty} \underline{x}(i)\underline{w}(i) = 0 \tag{5.3}$$

However, since by assumption, one of $\underline{x}(i)$ and $\underline{w}(i)$ is left-truncated, and by the property that left-terminated codes have $\underline{x}(i) = 0$, for $i < 0$ we have:

$$\sum_{i=0}^{\infty} \underline{x}(i)\underline{w}(i) = 0 \tag{5.4}$$

Similarly, because one of the codewords is right-truncated:

$$\sum_{i=0}^{L-1} \underline{x}(i)\underline{w}(i) = 0 \tag{5.5}$$

But since $\underline{x}(i) = \underline{x}_F(i)$, and $\underline{w}(i) = \underline{w}_F(i)$:

$$\sum_{i=0}^{L-1} \underline{x}_F(i)\underline{w}_F(i) = 0 \tag{5.6}$$

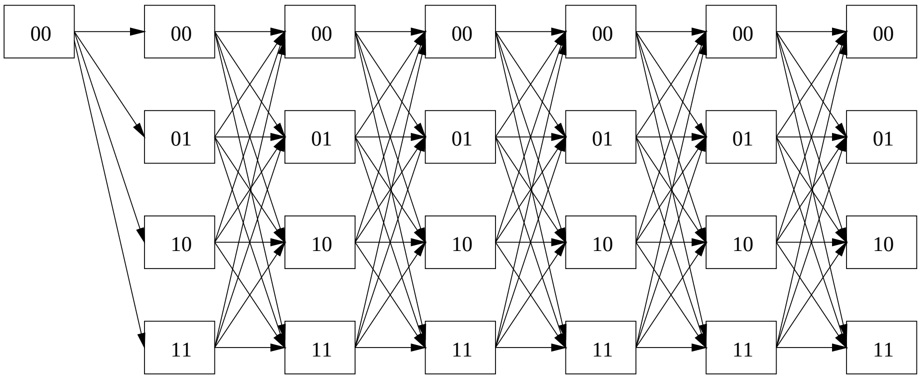
$\blacksquare$

Thus, a terminated convolutional code has an unterminated dual.

5.3 Example

To illustrate the termination/truncation duality aspect of dual convolutional codes, we consider as the primal code, that obtained by beginning the decoder defined in 3.8 in the all-zero state and running the encoder for $L = 6$ time steps. Since we do not restrict the input in any way, the encoder can end in any state.

Furthermore, it can be immediately seen from the matrices $\mathcal{A}$ and $\mathcal{B}$ that all transitions are possible. In general, we could determine the set of possible transitions from the states at time t to those at time $t+1$ by applying each of the 2^k possible inputs to each of the 2^m states. The trellis for this code is shown in the following figure:



From the polynomial parity-check matrix

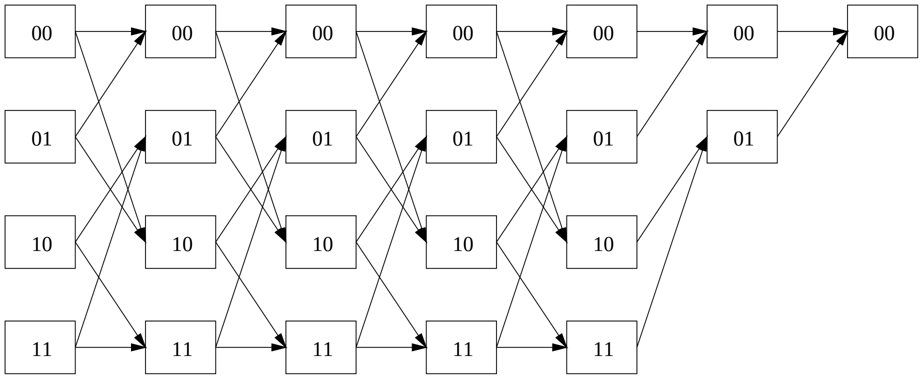
$$\mathcal{H}_{poly} = \begin{bmatrix} 1 & 1 + D^2 & 1 + D + D^2 \end{bmatrix} \tag{5.7}$$

we can obtain the state-space representation:

$$D^{-1}\underline{S}(D) = \underline{S}(D)\begin{bmatrix}0 & 1\\0 & 0\end{bmatrix} + \underline{U}(D)\begin{bmatrix}1 & 0\end{bmatrix} \tag{5.8}$$

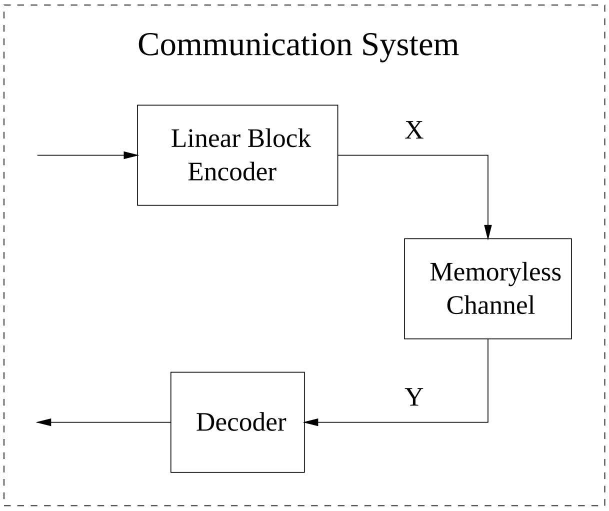
$$\underline{X}(D) = \underline{S}(D)\begin{bmatrix}0 & 1 & 0\\0 & 1 & 1\end{bmatrix} + \underline{U}(D)\begin{bmatrix}1 & 1 & 1\end{bmatrix} \tag{5.9}$$

From the discussion in the previous subsection, we know that the words in the dual code are generated by starting in an arbitrary state and ending in the all-zeros state. We also know that for the dual code $k = 1$ so that there are only two transitions possible per state. The exact transitions can be found by examining $\mathcal{A}$ and $\mathcal{B}$. The trellis for the dual code looks like:



6 Block Codes APP and Extrinsic Information

Stepping away from convolutional codes for the moment, we will prove several important theorems regarding bitwise a posteriori probabilities of code-bits of linear block codes. In particular, we will give formulas for the a posteriori probability and extrinsic information in terms of the words of the primal and dual codes. We note the similarity between all four forms, particularly the forms for extrinsic information in the primal and dual domain.



Suppose a codeword $\underline{x}$ is chosen at random from binary linear codebook $\mathbb{C}$ of dimension (n, k) with uniform probability. $\underline{x}$ is transmitted over a memoryless channel and the random variable $\underline{y}$ is observed. Each of the following four theorems provides a way of calculating (a simple function of) the posteriori probability distribution for each bit of $\underline{x}$ given $y_{(i)}$ in terms of words of the primal or dual code book $\mathbb{C}'$.

As a note on notation, we will use the notation $Pr(y_i|x_i)$ as a shorthand for $Pr(Y_i = y_i|X_i = x_i)$, except when we use constants, as in $Pr(y_i|X_i = 0)$, (which we do not replace by $Pr(y_i|0)$, since that expression would in general be ambiguous).

6.1 Primal Domain APP

In this section, we give a formula for the a posteriori probability distribution on a bit x_i of the codeword in terms of the received codeword $\underline{y}$.

Theorem 6.1 $\lambda_{apost,i} = \frac{Pr(X_i=0|\underline{y})}{Pr(X_i=1|\underline{y})} = \frac{\sum\limits_{\underline{x}\in\mathbb{C}, x_i=1} \prod\limits_{j=1}^{n} \lambda_j(y_j)^{x_j}}{\sum\limits_{\underline{x}\in\mathbb{C}, x_i=0} \prod\limits_{j=1}^{n} \lambda_j(y_j)^{x_j}}$

where $\lambda_j(y_j) = \frac{Pr(y_j|X_j=1)}{Pr(y_j|X_j=0)}$

Proof The formula for A Posteriori probability ratio $\lambda_{apost,i}$ of bit i expressed in terms of codewords in the primal domain is quite straightforward:

$$\lambda_{apost,i} = \frac{Pr(X_i = 1|\underline{y})}{Pr(X_i = 0|\underline{y})} = \frac{\sum\limits_{\underline{x}\in\mathbb{C}, x_i=1} Pr(\underline{x}|\underline{y})}{\sum\limits_{\underline{x}\in\mathbb{C}, x_i=0} Pr(\underline{x}|\underline{y})} \tag{6.1}$$

$$= \frac{\sum\limits_{\underline{x} \in \mathbb{C}, x_i = 1} Pr(y|x) \frac{Pr(\underline{x})}{Pr(\underline{y})}}{\sum\limits_{\underline{x} \in \mathbb{C}, x_i = 0} Pr(y|x) \frac{Pr(\underline{x})}{Pr(\underline{y})}} \tag{6.2}$$

Noting that $Pr(x) = 2^{-k}, \forall x$, and $Pr(y)$ is also a constant, both can be brought outside the sum and cancelled:

$$= \frac{\sum\limits_{\underline{x} \in \mathbb{C}, x_i = 1} Pr(\underline{y}|\underline{x})}{\sum\limits_{\underline{x} \in \mathbb{C}, x_i = 0} Pr(\underline{y}|\underline{x})} \tag{6.3}$$

By the fact that the channel is memoryless:

$$= \frac{\sum\limits_{\underline{x} \in \mathbb{C}, x_i = 1} \prod\limits_{j=1}^{n} Pr(y_j|x_j)}{\sum\limits_{\underline{x} \in \mathbb{C}, x_i = 0} \prod\limits_{j=1}^{n} Pr(y_j|x_j)} \tag{6.4}$$

dividing each term by $Pr(y_j|x_j = 0)$:

$$= \frac{\sum\limits_{\underline{x} \in \mathbb{C}, x_i = 1} \prod\limits_{j=1}^{n} \frac{Pr(y_j|x_j)}{Pr(y_j|x_j=0)}}{\sum\limits_{\underline{x} \in \mathbb{C}, x_i = 0} \prod\limits_{j=1}^{n} \frac{Pr(y_j|x_j)}{Pr(y_j|x_j=0)}} \tag{6.5}$$

by perfect induction on the cases $x_j = \{0, 1\}$,

$$= \frac{\sum\limits_{\underline{x} \in \mathbb{C}, x_i = 1} \prod\limits_{j=1}^{n} (\frac{Pr(y_j|x_j=1)}{Pr(y_j|x_j=0)})^{x_j}}{\sum\limits_{\underline{x} \in \mathbb{C}, x_i = 0} \prod\limits_{j=1}^{n} (\frac{Pr(y_j|x_j=1)}{Pr(y_j|x_j=0)})^{x_j}} \tag{6.6}$$

Substituting $\lambda_j(y_j) = \frac{Pr(y_j|x_j=1)}{Pr(y_j|x_j=0)}$, we have:

$$= \frac{\sum\limits_{\underline{x} \in \mathbb{C}, x_i = 1} \prod\limits_{j=1}^{n} \lambda_j(y_j)^{x_j}}{\sum\limits_{\underline{x} \in \mathbb{C}, x_i = 0} \prod\limits_{j=1}^{n} \lambda_j(y_j)^{x_j}} \tag{6.7}$$

$\blacksquare$

6.2 Primal Domain Extrinsic Information

We begin this section with a definition of extrinsic information:

$$\lambda_{ext,i} = \frac{Pr(X_i = 1|\underline{y}_{(i)})}{Pr(X_i = 0|\underline{y}_{(i)})} \tag{6.8}$$

Where $\underline{y}_{(i)}$ means all components of $\underline{y}$ except for y_i. While this definition may be slightly unfamiliar, it is arguably more basic than, and can be shown to be equivalent to, other definitions found in the literature.

Theorem 6.2 $\lambda_{ext,i} = \frac{Pr(X_i=1|\underline{y_{(i)}})}{Pr(X_i=0|\underline{y_{(i)}})} = \frac{\sum\limits_{\underline{x}\in\mathbb{C},x_i=1}\prod\limits_{j=1}^{n}\lambda_j(y_j)^{x_j}}{\lambda_i\cdot\sum\limits_{\underline{x}\in\mathbb{C},x_i=0}\prod\limits_{j=1}^{n}\lambda_j(y_j)^{x_j}}$

Proof We can imagine this as setting y_i to $\varnothing$ (the null symbol), so that the probabilty $Pr(y_i = \varnothing) = 1$ regardless of x_i, hence $\lambda_i(y_i) = \frac{1}{1} = 1$, so:

$$\lambda_{ext,i} = \frac{Pr(X_i = 1|\underline{y_{(i)}})}{Pr(X_i = 0|\underline{y_{(i)}})} = \left.\frac{\sum\limits_{\underline{x}\in\mathbb{C},x_i=1}\prod\limits_{j=1}^{n}\lambda_j(y_j)^{x_j}}{\sum\limits_{\underline{x}\in\mathbb{C},x_i=0}\prod\limits_{j=1}^{n}\lambda_j(y_j)^{x_j}}\right|_{\lambda_i(y_i)=1} \tag{6.9}$$

$$= \frac{\lambda_i^{-1}\cdot\sum\limits_{\underline{x}\in\mathbb{C},x_i=1}\prod\limits_{j=1}^{n}\lambda_j(y_j)^{x_j}}{\sum\limits_{\underline{x}\in\mathbb{C},x_i=0}\prod\limits_{j=1}^{n}\lambda_j(y_j)^{x_j}} \tag{6.10}$$

$$= \frac{\sum\limits_{\underline{x}\in\mathbb{C},x_i=1}\prod\limits_{j=1}^{n}\lambda_j(y_j)^{x_j}}{\lambda_i\cdot\sum\limits_{\underline{x}\in\mathbb{C},x_i=0}\prod\limits_{j=1}^{n}\lambda_j(y_j)^{x_j}} \tag{6.11}$$

$\blacksquare$

6.3 Dual Domain APP

Theorem 6.3

$$\phi_{apost,i} = Pr(X_i = 0|\underline{y}) - Pr(X_i = 1|\underline{y})$$
$$= \frac{\phi_i(y_i)\cdot\sum\limits_{\underline{w}\in\mathbb{C}',w_i=0}\prod\limits_{j=1}^{n}\phi_j(y_j)^{w_j} + \phi_i(y_i)^{-1}\sum\limits_{\underline{w}\in\mathbb{C}',w_i=1}\prod\limits_{j=1}^{n}\phi_j(y_j)^{w_j}}{\sum\limits_{\underline{w}\in\mathbb{C}',w_i=0}\prod\limits_{j=1}^{n}\phi_j(y_j)^{w_j}\quad\sum\limits_{\underline{w}\in\mathbb{C}',w_i=1}\prod\limits_{j=1}^{n}\phi_j(y_j)^{w_j}}$$

where $\phi_j = \frac{Pr(y_j|x_j=0)-Pr(y_j|x_j=1)}{Pr(y_j|x_j=0)+Pr(y_j|x_j=1)}$.

Proof

$$Pr(X_i = 0|\underline{y}) - Pr(X_i = 1|\underline{y}) = \sum_{\underline{x}\in\mathbb{C}} Pr(\underline{x}|\underline{y}) \cdot (-1)^{x_i} \tag{6.12}$$

We divide by $\sum_{\underline{x}\in\mathbb{C}} Pr(\underline{x}|\underline{y}) = 1$

$$= \frac{\sum_{\underline{x}\in\mathbb{C}} Pr(\underline{x}|\underline{y})(-1)^{x_i}}{\sum_{\underline{x}\in\mathbb{C}} Pr(\underline{x}|\underline{y})} \tag{6.13}$$

By Bayes' theorem:

$$= \frac{\sum_{\underline{x}\in\mathbb{C}} \frac{Pr(\underline{x})}{Pr(\underline{y})} Pr(\underline{y}|\underline{x})(-1)^{x_i}}{\sum_{\underline{x}\in\mathbb{C}} \frac{Pr(\underline{x})}{Pr(\underline{y})} Pr(\underline{y}|\underline{x})} \tag{6.14}$$

Since $Pr(\underline{x}) = 2^-k$ (constant), and $Pr(\underline{y})$ is not a function of $\underline{x}$, $\frac{Pr(\underline{x})}{Pr(\underline{y})}$ can be brought outside each sum and then canceled.

$$= \frac{\sum_{\underline{x}\in\mathbb{C}} Pr(\underline{y}|\underline{x})(-1)^{x_i}}{\sum_{\underline{x}\in\mathbb{C}} Pr(\underline{y}|\underline{x})} \tag{6.15}$$

Applying 12.2 (proof in appendix), taking $f(x)$ to be $Pr(\underline{y}|\underline{x})(-1)^{x_i}$ in the numerator and $Pr(\underline{y}|\underline{x})$ in the denominator,

$$= \frac{2^{-r} \sum_{\underline{w}\in\mathbb{C}'} \sum_{\underline{x}\in\{0,1\}^n} (-1)^{\underline{x}\cdot\underline{w}+x_i} Pr(\underline{y}|\underline{x})}{2^{-r} \sum_{\underline{w}\in\mathbb{C}'} \sum_{\underline{x}\in\{0,1\}^n} (-1)^{\underline{x}\cdot\underline{w}} Pr(\underline{y}|\underline{x})} \tag{6.16}$$

Now, we divide the numerator and denominator by $\sum_{\underline{x}\in\{0,1\}^n} Pr(\underline{y}|\underline{x})$ and bring the new denominator inside each sum:

$$= \frac{\sum_{\underline{w}\in\mathbb{C}'} \frac{\sum_{\underline{x}\in\{0,1\}^n} (-1)^{\underline{x}\cdot\underline{w}+x_i} Pr(\underline{y}|\underline{x})}{\sum_{\underline{x}\in\{0,1\}^n} Pr(\underline{y}|\underline{x})}}{\sum_{\underline{w}\in\mathbb{C}'} \frac{\sum_{\underline{x}\in\{0,1\}^n} (-1)^{\underline{x}\cdot\underline{w}} Pr(\underline{y}|\underline{x})}{\sum_{\underline{x}\in\{0,1\}^n} Pr(\underline{y}|\underline{x})}} \tag{6.17}$$

Because the channel is memoryless, we replace $Pr(\underline{y}|\underline{x})$ with $\prod_{j=1}^{n} Pr(y_j|x_j)$. We also replace $(-1)^{\underline{x}\cdot\underline{w}}$ with $\prod_{j=1}^{n}(-1)^{x_j\cdot w_j}$ and replace $(-1)^{x_i}$ with $\prod_{j=1}^{n}(-1)^{x_i\cdot\delta_{ij}}$ where δ_{ij} is 1 if and only if $i=j$ and 0 otherwise.

$$= \frac{\sum_{\underline{w}\in\mathbb{C}'} \frac{\sum_{\underline{x}\in\{0,1\}^n}\prod_{j=1}^{n}(-1)^{x_j\cdot(w_j+\delta_{ij})}Pr(y_j|x_j)}{\sum_{\underline{x}\in\{0,1\}^n}\prod_{j=1}^{n}Pr(y_j|x_j)}}{\sum_{\underline{w}\in\mathbb{C}'} \frac{\sum_{\underline{x}\in\{0,1\}^n}\prod_{j=1}^{n}(-1)^{x_j\cdot(w_j+\delta_{ij})}Pr(y_j|x_j)}{\sum_{\underline{x}\in\{0,1\}^n}\prod_{j=1}^{n}Pr(y_j|x_j)}} \tag{6.18}$$

By the distributive law we replace $\sum_{\underline{x}\in\{0,1\}^n}\prod_{j=1}^{n} f_j(x_j)$ with $\prod_{j=1}^{n}(f_j(0)+f_j(1))$.

$$\sum_{\underline{x}\in\{0,1\}^n}\prod_{j=1}^{n} f_j(x_j) =$$
$$\sum_{\underline{x_1}\in\{0,1\}}\sum_{\underline{x_2}\in\{0,1\}}\ldots\sum_{\underline{x_n}\in\{0,1\}} f_1(x_1)\cdot f_2(x_2)\cdot\ldots\cdot f_n(x_n) =$$
$$\sum_{\underline{x_1}\in\{0,1\}} f_1(x_1)\sum_{\underline{x_2}\in\{0,1\}} f_2(x_2)\ldots\sum_{\underline{x_n}\in\{0,1\}} f_n(x_n) =$$
$$\prod_{j=1}^{n}\sum_{x_j\in\{0,1\}} f_j(x_j) =$$
$$\prod_{j=1}^{n}(f_j(0)+f_j(1))$$

$$= \frac{\sum_{\underline{w}\in\mathbb{C}'} \frac{\prod_{j=1}^{n}(Pr(y_j|x_j=0)+(-1)^{w_j+\delta_{ij}}Pr(y_j|x_j=1))}{\prod_{j=1}^{n}(Pr(y_j|x_j=0)+Pr(y_j|x_j=1))}}{\sum_{\underline{w}\in\mathbb{C}'} \frac{\prod_{j=1}^{n}(Pr(y_j|x_j=0)+(-1)^{w_j}Pr(y_j|x_j=1))}{\prod_{j=1}^{n}(Pr(y_j|x_j=0)+Pr(y_j|x_j=1))}} \tag{6.19}$$

taking the product outside of the fraction and replacing $(-1)^{w_j+\delta_{ij}}$ with $(-1)^{w_j\oplus\delta_{ij}}$

$$= \frac{\sum_{\underline{w}\in\mathbb{C}'}\prod_{j=1}^{n} \frac{Pr(y_j|x_j=0)+(-1)^{w_j\oplus\delta_{ij}}Pr(y_j|x_j=1)}{Pr(y_j|x_j=0)+Pr(y_j|x_j=1)}}{\sum_{\underline{w}\in\mathbb{C}'}\prod_{j=1}^{n} \frac{Pr(y_j|x_j=0)+(-1)^{w_j}Pr(y_j|x_j=1)}{Pr(y_j|x_j=0)+Pr(y_j|x_j=1)}} \tag{6.20}$$

By perfect induction the two cases $w_j\oplus\delta_{ij}=\{0,1\}$ it can be seen that:

$$= \frac{\sum\limits_{\underline{w} \in \mathbb{C}'} \prod\limits_{j=1}^{n} \left(\frac{Pr(y_j|x_j=0)-Pr(y_j|x_j=1)}{Pr(y_j|x_j=0)+Pr(y_j|x_j=1)}\right)^{w_j \oplus \delta_{ij}}}{\sum\limits_{\underline{w} \in \mathbb{C}'} \prod\limits_{j=1}^{n} \left(\frac{Pr(y_j|x_j=0)-Pr(y_j|x_j=1)}{Pr(y_j|x_j=0)+Pr(y_j|x_j=1)}\right)^{w_j}} \tag{6.21}$$

Substituting the definition of ϕ_j, we have:

$$= \frac{\sum\limits_{\underline{w} \in \mathbb{C}'} \prod\limits_{j=1}^{n} \phi_j^{w_j \oplus \delta_{ij}}}{\sum\limits_{\underline{w} \in \mathbb{C}'} \prod\limits_{j=1}^{n} \phi_j^{w_j}} \tag{6.22}$$

Separating the sum into two parts:

$$= \frac{\sum\limits_{\underline{w} \in \mathbb{C}', w_i=0} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j+\delta_{ij}} + \sum\limits_{\underline{w} \in \mathbb{C}', w_i=1} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j+\delta_{ij}}}{\sum\limits_{\underline{w} \in \mathbb{C}', w_i=0} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j} + \sum\limits_{\underline{w} \in \mathbb{C}', w_i=1} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j}} \tag{6.23}$$

Examining each of the two sums in the numerator, we see that we can replace the term δ_{ij} in the exponent with the factor $\phi_i(y_i)$ or $\phi_i(y_i)^{-1}$ as follows:

$$= \frac{\phi_i(y_i) \cdot \sum\limits_{\underline{w} \in \mathbb{C}', w_i=0} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j} + \phi_i(y_i)^{-1} \sum\limits_{\underline{w} \in \mathbb{C}', w_i=1} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j}}{\sum\limits_{\underline{w} \in \mathbb{C}', w_i=0} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j} \sum\limits_{\underline{w} \in \mathbb{C}', w_i=1} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j}} \tag{6.24}$$

∎

6.4 Dual Domain Extrinsic Information

The SISO decoder module for a turbo-code returns the likelihood function of x_i predicated on $\underline{y}_{(i)}$, where $\underline{y}_{(i)}$ stands for the vector $\underline{y}$ excluding the i^{th} component.

Theorem 6.4 $\phi_{ext,i} = Pr(X_i = 0|\underline{y_{(i)}}) - Pr(x = 1|\underline{y_{(i)}}) = \frac{\sum\limits_{\underline{w} \in \mathbb{C}', w_i=0} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j}}{\phi_i(y_i) \sum\limits_{\underline{w} \in \mathbb{C}', w_i=1} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j}}$

Proof Again setting $y_i = \varnothing$, we have $\frac{Pr(y_i|X_i=0)-Pr(y_i|X_i=1)}{Pr(y_i|X_i=0)+Pr(y_i|X_i=1)} = \frac{1-1}{1+1} = 0$, so the quantity on the left can be calculated from 6.23 by setting ϕ_i to 0:

$$\phi_{ext,i} = \left. \frac{\sum\limits_{\underline{w}\in\mathbb{C}',w_i=0} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j+\delta_{ij}} + \sum\limits_{\underline{w}\in\mathbb{C}',w_i=1} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j+\delta_{ij}}}{\sum\limits_{\underline{w}\in\mathbb{C}',w_i=0} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j} + \sum\limits_{\underline{w}\in\mathbb{C}',w_i=1} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j}} \right|_{\phi_i(y_i)=0} \tag{6.25}$$

We can see that two of the terms drop out because they include the factor $\phi_i(y_i) = 0$, so that we can rewrite:

$$= \frac{\sum\limits_{\underline{w}\in\mathbb{C}',w_i=0} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j+\delta_{ij}}}{\sum\limits_{\underline{w}\in\mathbb{C}',w_i=1} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j}} \tag{6.26}$$

$$= \frac{\phi_i(y_i)^{-1} \sum\limits_{\underline{w}\in\mathbb{C}',w_i=0} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j}}{\sum\limits_{\underline{w}\in\mathbb{C}',w_i=1} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j}} \tag{6.27}$$

$$= \frac{\sum\limits_{\underline{w}\in\mathbb{C}',w_i=0} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j}}{\phi_i(y_i) \sum\limits_{\underline{w}\in\mathbb{C}',w_i=1} \prod\limits_{j=1}^{n} \phi_j(y_j)^{w_j}} \tag{6.28}$$

$\blacksquare$

7 BCJR and DSISO algorithms

In this chapter, we show how to make one of four types of SISO decoders using one simple building block, in combination with just a few extra arithmetic operations.

Considering a finite-length (truncated or terminated) binary convolutional code $\mathbb{C}$ of length $n_0 \cdot L$, we will derive an algorithm to calculate the quantity:

$$S_{a,\mathbb{C},\underline{f}}(i) = \sum_{\underline{x}\in\mathbb{C},x_i=a} \prod_{j=1}^{L\cdot n_0} f_j^{x_j} \tag{7.1}$$

for $a \in \{0, 1\}$, and arbirtary $f \in \mathbb{R}^{n_0 \cdot L}$.

We then show how each of the four quantities $\lambda_{apost,i}$, $\lambda_{ext,i}$, $\phi_{apost,i}$, $\phi_{ext,i}$ can simply be computed in terms of quantities S.

7.1 Calculation of S

From 5, we remember that each codeword is represented by a path through the trellis. If i represents a time in the trellis (such that $0 <= i <= L$), we let s_i be a possible state of the encoder at time i.

For $0 <= i <= L$, we let $\mathbb{P}_{s_i}^{s_{i+1}}$ represent all of the (partial) codewords associated with paths from s_i to s_{i+1}. In general there can be multiple paths even between adjacent states in the generator. However, if the Forney Indices are all non-zero, then there will be either 1 or 0 paths between each adjacent pair of states.

We will also let $\mathbb{P}_0^{s_i}$ represent all partial codewords associated with paths from some state at time 0 to the state s_i, and $\mathbb{P}_{s_i}^L$ represent partial codewords from state s_i to some state at time L.

In order to calculate For a particular i, $0 <= i <= L-1$, we note that every path through the trellis is a path from s_0 to s_i, combined with a path from s_i to s_{i+1}, combined with a path from s_{i+1} to s_L, for some states (s_i, s_{i+1}):

$$\mathbb{C} = \bigcup_{s_i, s_{i+1}} \mathbb{P}_0^{s_i} \times \mathbb{P}_{s_i}^{s_{i+1}} \times \mathbb{P}_{s_{i+1}}^L \tag{7.2}$$

Using that fact, we can rewrite the following:

$$S_{a,\mathbb{C},\underline{f}}(l) = \sum_{\underline{x} \in \mathbb{C}, x_l = a} \prod_{j=1}^{n} f_j^{x_j} \tag{7.3}$$

In terms of the sum over the paths going through each pair of states at time i and $i+1$:

$$= \sum_{s_i, s_{i+1}} \left[\sum_{x_1^{n_0 i} \in \mathbb{P}_0^{s_i}} \sum_{x_{n_0 i+1}^{n_0 (i+1)} \in \mathbb{P}_{s_i}^{s_{i+1}}} \sum_{x_{n_0 (i+1)+1}^{n_0 L} \in \mathbb{P}_{s_{i+1}}^L} \prod_{j=1}^{n_0 i} f_j^{x_j} \prod_{j=i \cdot n_0+1}^{(i+1)\cdot n_0} f_j^{x_j} \prod_{j=(i+1)\cdot n_0+1}^{L \cdot n_0} f_j^{x_j} \cdot \delta_{x_l,a} \right] \tag{7.4}$$

By the distributive law, we can rearrange as follows:

$$= \sum_{s_i, s_{i+1}} \left[\sum_{x_1^{n_0 i} \in \mathbb{P}_0^{s_i}} \prod_{j=1}^{n_0 i} f_j^{x_j} \sum_{x_{n_0 i+1}^{n_0 (i+1)} \in \mathbb{P}_{s_i}^{s_{i+1}}} \prod_{j=i \cdot n_0+1}^{(i+1)\cdot n_0} f_j^{x_j} \sum_{x_{n_0 (i+1)+1}^{n_0 L} \in \mathbb{P}_{s_{i+1}}^L} \prod_{j=(i+1)\cdot n_0+1}^{L \cdot n_0} f_j^{x_j} \cdot \delta_{x_l,a} \right] \tag{7.5}$$

And finally:

$$= \sum_{s_i,s_{i+1}} \left[\sum_{x_1^{n_0 i} \in \mathbb{P}_0^{s_i}} \prod_{j=1}^{n_0 i} f_j^{x_j} \right] \left[\sum_{x_{n_0 i+1}^{n_0(i+1)} \in \mathbb{P}_{s_i}^{s_{i+1}}} \prod_{j=i\cdot n_0+1}^{(i+1)\cdot n_0} f_j^{x_j} \right] \left[\sum_{x_{n_0(i+1)+1}^{n_0 L} \in \mathbb{P}_{s_{i+1}}^{L}} \prod_{j=(i+1)\cdot n_0+1}^{L\cdot n_0} f_j^{x_j} \right] \cdot \delta_{x_l,a} \tag{7.6}$$

Where $i \cdot n_0 < l < L \cdot n_0$

We identify each of the three sums over partial paths:

$$\alpha_{s_i} = \sum_{x_1^{n_0 i} \in \mathbb{P}_0^{s_i}} \prod_{j=1}^{n_0 i} f_j^{x_j} \tag{7.7}$$

$$\gamma_{s_i,s_{i+1}} = \sum_{x_{n_0 i+1}^{n_0(i+1)} \in \mathbb{P}_{s_i}^{s_{i+1}}} \prod_{j=i\cdot n_0+1}^{(i+1)\cdot n_0} f_j^{x_j} \tag{7.8}$$

$$\beta s_{i+1} = \sum_{x_{n_0(i+1)+1}^{n_0 L} \in \mathbb{P}_{s_{i+1}}^{L}} \prod_{j=(i+1)\cdot n_0+1}^{L\cdot n_0} f_j^{x_j} \tag{7.9}$$

So that

$$S_{a,\mathbb{C},\underline{f}}(l) = \sum_{s_i,s_{i+1}} \alpha_{s_i} \gamma_{s_i,s_{i+1}} \beta s_{i+1} \cdot \delta_{x_i,a} \tag{7.10}$$

Now, if we can find a fast algorithm to calculate the quantities α, β, and γ, then we can also quickly calculate anything of the form $S_{a,\mathbb{C},\underline{f}}(l) = \sum_{\underline{x}\in\mathbb{C},x_l=a} \prod_{j=1}^{n} f_j^{x_j}$. We note that we can calculate each γ directly and we will now show that we can recursively calculate α and β.

The calculation of α will proceed from each of $\alpha s_0, \forall s_0$ sequentially through each of $\alpha s_{L-1} \forall s_{L-1}$. We begin with boundary conditions for each of α:

$$\alpha_{s_0} = \sum_{x_1^{n_0 i} \in \mathbb{P}_0^{s_0}} \prod_{j=1}^{0} f_j^{x_j} \tag{7.11}$$

Since the empty product is equal to 1, α_{s_0} is equal to 1 if there is any path going through s_0 and 0 otherwise, thus for left-terminated convolutional codes:

$$\alpha_{s_0} = \begin{cases} 1 & s_0 = \underline{0} \\ 0 & otherwise \end{cases} \tag{7.12}$$

and for left-truncated codes:

$$\alpha_{s_0} = 1 \tag{7.13}$$

Similarly, for right-terminated codes:

$$beta_{s_{L-1}} = \begin{cases} 1 & s_{L-1} = \underline{0} \\ 0 & otherwise \end{cases} \tag{7.14}$$

and for right-truncated codes:

$$beta_{s_{L-1}} = 1 \tag{7.15}$$

Now, we will consider calculation of $\alpha_{s_{i+1}} \forall s_{i+1}$ in terms of α_{s_i}.

$$\alpha_{s_{i+1}} = \sum_{x_1^{n_0 i+1} \in \mathbb{P}_0^{s_{i+1}}} \prod_{j=1}^{n_0 i+1} f_j^{x_j} \tag{7.16}$$

Since every path from a state at time 0 to a state at time $i+1$ goes through exactly one state at time i:

$$= \sum_{s_i} \left[\sum_{x_1^{n_0 i} \in \mathbb{P}_0^{s_i}} \prod_{j=1}^{n_0 i} f_j^{x_j} \cdot \sum_{x_{n_0 i+1}^{n_0 (i+1)} \in \mathbb{P}_0^{i}} \prod_{j=1}^{n_0 i} f_j^{x_j} \right] \tag{7.17}$$

$$= \sum_{s_i} [\alpha_{s_i} \gamma_{s_i, s_{i+1}}] \tag{7.18}$$

Similarly, for the quantities β:

$$\beta s_i = \sum_{x_{n_0 i+1}^{n_0 L} \in \mathbb{P}_{s_{i+1}}^{L}} \prod_{j=(i+1)\cdot n_0+1}^{L \cdot n_0} f_j^{x_j} \tag{7.19}$$

$$= \sum_{s_{i+1}} \left[\sum_{x_{n_0 i+1}^{n_0 (i+1)} \in \mathbb{P}_{s_i}^{s_{i+1}}} \prod_{j=i\cdot n_0+1}^{(i+1)\cdot n_0} f_j^{x_j} \sum_{x_{n_0 (i+1)+1}^{n_0 L} \in \mathbb{P}_{s_{i+1}}^{L}} \prod_{j=(i+1)\cdot n_0+1}^{L \cdot n_0} f_j^{x_j} \right] \tag{7.20}$$

$$= \sum_{s_{i+1}} \left[\gamma_{s_i, s_{i+1}} \beta s_{i+1} \right] \tag{7.21}$$

Thus, each α and β can be calculated with only a relatively few arithmetic operations. We will examine the exact number required in the following section.

7.2 SISO Realization

Now, we turn our attention to realizations of each of the four kinds of SISO decoders described in the previous section.

from 6.1, we have:

$$\lambda_{apost,i} = \sum_{\underline{x}\in\mathbb{C},x_i=1} \prod_{j=1}^{n} \lambda_j(y_j)^{x_j} \sum_{\underline{x}\in\mathbb{C},x_i=0} \prod_{j=1}^{n} \lambda_j(y_j)^{x_j} \tag{7.22}$$

Which we rewrite in terms of quantities S:

$$= \frac{S_{1,\mathbb{C},\underline{\lambda}}(i)}{S_{0,\mathbb{C},\underline{\lambda}}(i)} \tag{7.23}$$

From 6.2

$$\lambda_{ext,i} = \frac{\sum_{\underline{x}\in\mathbb{C},x_i=1} \prod_{j=1}^{n} \lambda_j(y_j)^{x_j}}{\lambda_i \cdot \sum_{\underline{x}\in\mathbb{C},x_i=0} \prod_{j=1}^{n} \lambda_j(y_j)^{x_j}} \tag{7.24}$$

$$= \frac{S_{0,\mathbb{C},\underline{\lambda}}(i)}{\lambda_i(y_i) S_{1,\mathbb{C},\underline{\lambda}}(i)} \tag{7.25}$$

From 6.3

$$\phi_{apost,i} = \frac{\phi_i(y_i) \cdot \sum_{\underline{w}\in\mathbb{C}',w_i=0} \prod_{j=1}^{n} \phi_j(y_j)^{w_j} + \phi_i(y_i)^{-1} \sum_{\underline{w}\in\mathbb{C}',w_i=1} \prod_{j=1}^{n} \phi_j(y_j)^{w_j}}{\sum_{\underline{w}\in\mathbb{C}',w_i=0} \prod_{j=1}^{n} \phi_j(y_j)^{w_j} + \sum_{\underline{w}\in\mathbb{C}',w_i=1} \prod_{j=1}^{n} \phi_j(y_j)^{w_j}} \tag{7.26}$$

$$= \frac{\phi_i(y_i) \cdot S_{0,\mathbb{C}',\phi(y)}(i) + \phi_i(y_i)^{-1} \cdot S_{1,\mathbb{C}',\phi(y)}(i)}{\cdot S_{0,\mathbb{C}',\phi(y)}(i) + \cdot S_{1,\mathbb{C}',\phi(y)}(i)} \tag{7.27}$$

And finally, from 6.4

$$\phi_{ext,i} = \frac{\sum_{\underline{w}\in\mathbb{C}',w_i=0} \prod_{j=1}^{n} \phi_j(y_j)^{w_j}}{\phi_i(y_i) \sum_{\underline{w}\in\mathbb{C}',w_i=1} \prod_{j=1}^{n} \phi_j(y_j)^{w_j}} \tag{7.28}$$

$$= \frac{\cdot S_{1,\mathbb{C}',\phi(y)}(i)}{\phi_i(y_i) \cdot S_{0,\mathbb{C}',\phi(y)}(i)} \tag{7.29}$$

8 Complexity

The complexity of the BCJR and DSISO algorithms can be divided into the complexity of calculating the four quantities γ, α, $beta$, and λ_{out} (or ϕ_{out} instead of $\lambda|out$ in the dual domain). Since there is necessarily some signal processing done before running the decoding algorithm, we will assume that either λ or ϕ is freely available at the start of the algorithm, and that either quantity $lambda_{out}$ or ϕ_{out} is acceptable as output, so that neither algorithm suffers due to the format of its input or output. In real systems, this might be unacceptable for various reasons, in which case one or the other of the algorithms (but more likely DSISO algorithm) might have to pay a small penalty for transforming its input and/or output.

8.1 Probabiliy domain complexity

We term the domain in which calculations are done directly on algebraic functions of probability elements *probability domain algorithms*, as opposed to *log-probability domain algorithms*. Because the formulas we have derived are algebraic functions of probability elements, computations in this domain involve only algebraic operations.

The computation of each of the $L \cdot 2^{m+k_0}$ components of γ requires n_0 multiplications.

The computation of each of the $L \cdot 2^m$ components of α requires 2_0^k multiplications and $2_0^k - 1$ additions.

The computation of each of the $L \cdot 2^m$ components of β also require the same, though limited-memory applications will likely use the sliding-window technique, raising this cost by some ammount.

Finally, the computation of each of L components of λ_{out} requires $2 \cdot 2^m + k_0$ multiplications, $2^m + k_0 - 2$ additions, and a single division.

In total, neglecting terms exponentially smaller than 2^{m+k_0}, the complexity is:

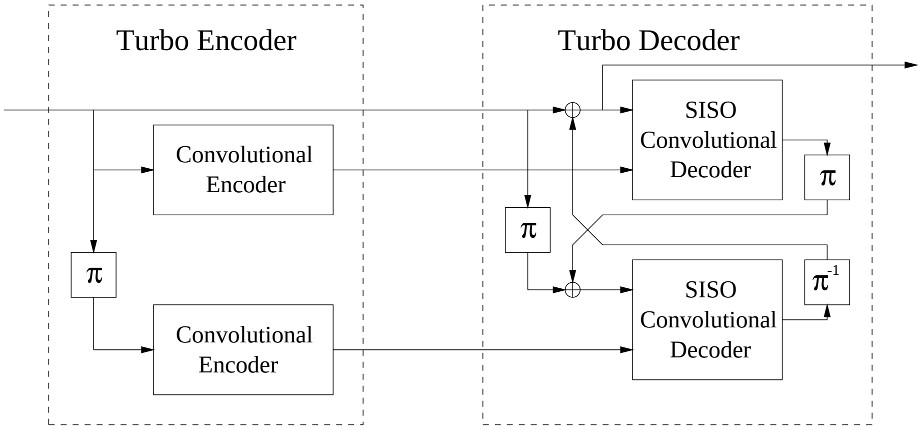
- $(n_0 + 4) \cdot 2^{m+k_0}$ multiplications
- $2 \cdot 2^{m+k_0}$ additions

8.2 Log-probability domain complexity

Since the probability-domain algorithm involves multiplication, there is the idea that performing calculations in the logarithmic domain may simplify the algorithm since it takes multiplications (relatively complex operations) into additions (relatively simple operations). Unfortunately, the probability domain algorithm also makes use of additions which, when brought into logarithmic domain, become non-algebraic functions, which theoretically take an infinite time to compute. The idea, however, is to generate a good-enough approximation to this function and thus reduce complexity on the whole, while maintaining high numerical accuracy.

[Discussion of possible extension of max^* algorithm to the case of positive and negative quantities]

9 Turbo Coding System using DSISO



[Discussion of a possible parallel turbo-coding system using truncated systematic convolutional encoders]

10 Discussion

10.1 Numerical Accuracy

[Discussion of why it's perhaps easier to gurantee numerical accuracy in the primal domain than the dual domain]

10.2 Limitation to Systematic Information

[Discusion of the causes and consequences of the DSISO's limitation to code-symbols as opposed to information-symbols]

11 Notation

$\mathbb{F}$ - A finite field
k_0 - Dimension of the input to a convolutional encoder.
n_0 - Dimension of the output of a convolutional encoder.
m - Dimension of the memory of a state-space encoder; memory of a code.
$\underline{s}(i)$ - State (vector) of a convolutional encoder at time i
$\underline{u}(i)$ - Input (vector) to a convolutional encoder at time i
$\underline{x}(i)$ - Output (vector) of a convolutional encoder at time i
$\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$ - Matricies of scalars in $\mathbb{F}$

D - Formal variable of the D-transform
$\mathcal{G}(D)$ - a $(k_0 \times n_0)$ generator matrix with elements over $\mathbb{F}(D)$
$\mathcal{H}(D)$ - a $(n_0 - k_0 \times n_0)$ parity matrix with elements over $\mathbb{F}(D)$
$\mathcal{T}(D)$ - a $(k_0 \times k_0)$ matrix with elements over $\mathbb{F}(D)$
$\lambda_{apost,i}$ - the quantity $\frac{Pr(X_i=1|y)}{Pr(X_i=0|y)}$
$\lambda_{ext,i}$ - the quantity $\frac{Pr(X_i=1|y_{(i)})}{Pr(X_i=0|y_{(i)})}$
$\lambda_i(y_i)$ - the quantity $\frac{Pr(y_i|x_i=1)}{Pr(y_i|x_i=0)}$
$\phi_{apost,i}$ - the quantity $\frac{Pr(X_i=0|y)-Pr(X_i=1|y)}{Pr(X_i=0|y)+Pr(X_i=1|y)}$
$\phi_{ext,i}$ - the quantity $\frac{Pr(X_i=0|y_{(i)})-Pr(X_i=1|y_{(i)})}{Pr(X_i=0|y_{(i)})+Pr(X_i=1|y_{(i)})}$
$\phi_i(y_i)$ - the quantity $\frac{Pr(y_i|x_i=0)-Pr(y_i|x_i=1)}{Pr(y_i|x_i=0)+Pr(y_i|x_i=1)}$
$\mathbb{C}$ - Code (set of codewords)
$\mathbb{C}'$ - Dual Code (set of words orthogonal to all codewords)
$\underline{X}$ - The transmitted codeword
$\underline{Y}$ - The received vector
$\underline{x}$, $\underline{w}$ - variable in $\{0,1\}^n$
x_i - the i^{th} component of $\underline{x}$
$Pr(\underline{x}) = Pr(\underline{X} = \underline{x})$

12 Appendix

Here, we prove the following lemma and its corrolary:

Lemma 12.1 $\sum_{\underline{w}\in\mathbb{C}'} (-1)^{\underline{x}\cdot\underline{w}} = \begin{cases} 2^r & \underline{x} \in \mathbb{C}, \\ 0 & \underline{x} \notin \mathbb{C}. \end{cases}$

Proof if $\underline{x} \in \mathbb{C}$, then the inner product with all $\underline{w} \in \mathbb{C}'$ is 0 and each term in the sum is 1. There are 2^r terms, so the sum is equal to 2^r

if $\underline{x} \notin \mathbb{C}$, then there exists some $\underline{\alpha} \in \mathbb{C}$ s.t. $\underline{\alpha} \cdot \underline{x} \neq 0$. It is a property of linear codes that for $\underline{\alpha} \in \mathbb{C}'$, if $\underline{w} \in \mathbb{C}'$ then $\underline{w} + \underline{\alpha} \in \mathbb{C}'$ so:

$$\sum_{\underline{w}\in\mathbb{C}'} (-1)^{\underline{w}\cdot\underline{x}} = \sum_{\underline{w}\in\mathbb{C}'} (-1)^{(\underline{w}+\underline{\alpha})\cdot\underline{x}} \tag{12.1}$$

$$= \sum_{\underline{w}\in\mathbb{C}'} (-1)^{\underline{w}\cdot\underline{x}} \cdot (-1)^{\underline{x}\cdot\underline{\alpha}} \tag{12.2}$$

$$= -\sum_{\underline{w}\in\mathbb{C}'} (-1)^{\underline{x}\cdot\underline{w}} \tag{12.3}$$

if $a = -a$, then $a = 0$

$$= 0 \tag{12.4}$$

∎

Corollary 12.2

$$2^{-r} \sum_{\underline{w} \in \mathbb{C}'} \sum_{\underline{x} \in \{0,1\}^n} (-1)^{\underline{x} \cdot \underline{w}} f(\underline{x}) = \sum_{\underline{x} \in \mathbb{C}} f(\underline{x}) \tag{12.5}$$

Proof Changing the order of summation:

$$2^{-r} \sum_{\underline{w} \in \mathbb{C}'} \sum_{\underline{x} \in \{0,1\}^n} (-1)^{\underline{x} \cdot \underline{w}} f(\underline{x}) = 2^{-r} \sum_{\underline{x} \in \{0,1\}^n} \sum_{\underline{w} \in \mathbb{C}'} (-1)^{\underline{x} \cdot \underline{w}} f(\underline{x}) \tag{12.6}$$

By the distributive law

$$= 2^{-r} \sum_{\underline{x} \in \{0,1\}^n} f(\underline{x}) \sum_{\underline{w} \in \mathbb{C}'} (-1)^{\underline{x} \cdot \underline{w}} \tag{12.7}$$

By 12.1

$$= 2^{-r} \sum_{\underline{x} \in \mathbb{C}} f(\underline{x}) 2^r \tag{12.8}$$

Bringing 2^r out of the summation:

$$= \sum_{\underline{x} \in \mathbb{C}} f(\underline{x}) \tag{12.9}$$

∎

References

[1] C. R. Hartmann, L. D. Rudolph, "An optimum symbol-by-symbol decoding rule for linear codes", *IEEE Transactions on information Theory*, vol. IT-22, pp. 514-517, Sept. 1976.

[2] S. Riedel, "Map decoding of convolutional codes using reciprocal dual codes", *IEEE Transactions on information Theory*, vol. 44 3, pp. 1176-1197, May 1998.

[3] G. Montorsi, S. Benedetto "An Additive Version of the DISO Algorithm for the Dual Code" IEEE International Symposium on Information Theory, 2001

[4] R. McEliece "The Algebraic Theory of Convolutional Codes" *The Handbook of Coding Theory*, 1998.

[5] R. Johanneson, K. Zigangirov "Fundamentals of Convolutional Coding" IEEE Press, 1999.