This note describes the derivations for BCJR algorithm and the turbo decoding algorithm. These algorithms are described in context of the LTE parallel concatenated code, aka turbo code.

BCJR is a SISO algorithm for convolutional codes that provides the necessary soft outputs for turbo decoding.

What motivates BCJR is MAP. In context of decoding transmission bits, MAP is an estimate on the bits of 1 or 0. As such, MAP can be expressed as

$$\frac{P(\hat{d}_k = 1|\mathbf{x})}{P(\hat{d}_k = 0|\mathbf{x})} \tag{1}$$

where $\hat{d}_k$ is the bit estimate at time index k, and $\mathbf{x}$ is the receive vector.

If the ratio shown above is > 1, the decision should be to select 1 as the most likely outcome for the bit. On the other hand, if the ratio is ¡ 1, the decision should be to select 0. This calculation is straight forward but it doesn't lend itself to implementation well. The reason is the division and multiplications involved, which are highly complex circuits as compared to addition or subtraction functions. So, typically the logarithmic function is applied to the probability ratios to obtain a version of MAP called logMAP. logMAP would be

$$\log P(\hat{d}_k = 1|\mathbf{x}) - log(P(\hat{d}_k = 0|\mathbf{x}) \tag{2}$$

Due to the monotonic curve of the log function, the decision likelihood after the log transformation is equivalent to the original MAP. Hence, (2) is equivalent in its decision likelihood as (1). By using (2), however, the hardware (or computational complexity) heavy divisions and multiplications are converted to less complex additions or subtractions. Of course, there is the problem of carrying the log functions throughout computations but that is only an issue of tracking all computations in the log domain. Only the most primative variables must be converted to log at the input to the logMAP algorithm.

To start logMAP, the probability $P(\hat{d}_k|\mathbf{x})$ has to expanded. The first step of expansion is to include the trellis states of the LTE code. From the axioms of probability theory,

$$P(A) = \sum_{\{B\}} P(A, B)$$

Using that axiom on logMAP,

$$P(\hat{d}_k|\mathbf{x}) = \sum_{\{s_{k-1}\}} P(\hat{d}_k, s_{k-1}|\mathbf{x}) \tag{3}$$

where s_{k-1} is the trellis state from where $\hat{d}_k$ exit and into state s_k. This equation needs further expansion before the important first step in deriving BCJR. Another axiom in probability theory is needed, and that is

$$P(A, B) = P(B|A)P(A) \tag{4}$$

Further expansion gives

$$P(\hat{d}_k, s_{k-1}|\mathbf{x}) = \frac{P(\hat{d}_k, s_{k-1}, \mathbf{x})}{P(\mathbf{x})} \tag{5}$$

Vector $\mathbf{x}$ can be broken into two pieces $\mathbf{x}_k$ and $\mathbf{x'}_{k+1}$ where the concatenation of these two shorter vectors equals the original $\mathbf{x}$. $\mathbf{x}_k$ refers to the vector from index 0 to k and $\mathbf{x'}_{k+1}$ is the vector from index $k+1$ to the last sample of $\mathbf{x}$. The $'$ symbol marks the vector as causally indexed from the index forward. Similarly, the vector without the $'$ symbol marks the vector as anti-causally index from the first sample of the vector to the index subscript.

Again referring to the axioms of probability theory,

$$P(\hat{d}_k, s_{k-1}, \mathbf{x}_k, \mathbf{x'}_{k+1}) = P(\hat{d}_k, s_{k-1}, \mathbf{x}_k) P(\mathbf{x'}_{k+1} | \hat{d}_k, s_{k-1}) \tag{6}$$

In (6), notice that $\mathbf{x'}_{k+1}$ is not dependent on $\mathbf{x}_k$. This assumption simplifies much of the math that follows and it implies that the received signal is corrupted by a white noise source. That is, the received signal has no correlation between samples. In context of decoding, each sample refers to a bit duration.

So now

$$\begin{aligned} L(\hat{d}_k) &= \log\left(\frac{\sum P(\hat{d}_k = 1, s_{k-1}, \mathbf{x}_k) P(\mathbf{x'}_{k+1} | \hat{d}_k = 1, s_{k-1}) \quad / \quad P(\mathbf{x})}{\sum P(\hat{d}_k = 0, s_{k-1}, \mathbf{x}_k) P(\mathbf{x'}_{k+1} | \hat{d}_k = 0, s_{k-1}) \quad / \quad P(\mathbf{x})}\right) & (7) \\ &= \log\left(\frac{\sum P(\hat{d}_k = 1, s_{k-1}, \mathbf{x}_k) P(\mathbf{x'}_{k+1} | \hat{d}_k = 1, s_{k-1})}{\sum P(\hat{d}_k = 0, s_{k-1}, \mathbf{x}_k) P(\mathbf{x'}_{k+1} | \hat{d}_k = 0, s_{k-1})}\right) & (8) \end{aligned}$$

From (8), the terms of interest are $P(\hat{d}_k, s_{k-1}, \mathbf{x}_k)$ and $P(\mathbf{x'}_{k+1} | \hat{d}_k, s_{k-1})$. If these two terms are known, then making an estimate is to choose 1 if $L(\hat{d}_k) > 0$ and 0 otherwise. One type of algorithm to find these two terms is BCJR, which uses iterations through the code trellis to find the probabilities. As will be shown, iterations are recursive calculations to find the probability at index k.

In BCJR parlance, the first of the two terms, $P(\hat{d}_k, s_{k-1}, \mathbf{x_1})$, is called the forward estimates. The second term, $P(\mathbf{x_2} | \hat{d}_k, s_{k-1})$, is called the backward estimates. To derive the iteration algorithm on each of these terms, there are many steps starting with expansions.

Expanding the forward term using the axioms of probability yields the following,

$$P(\hat{d}_k, s_{k-1}, \mathbf{x_k}) = \sum_{\{s_{k-2}\}} \sum_{\{\hat{d}_{k-1}\}} P(\hat{d}_k, s_{k-1}, x_k, \hat{d}_{k-1}, s_{k-2}, \mathbf{x}_{k-1}) \tag{9}$$

After breaking up $\mathbf{x}_k$ into a sample at index k and past samples from 1 to $k-1$, (9) inserts two more terms into the probability: $\hat{d}_{k-1}$ and s_{k-2}. Adding these two new terms into the equation creates a recursion pattern. The recursive pattern is more clear further expansion. Each step of expansion is an application of an axiom of probability. (9) follows as

$$\begin{aligned} &= \sum_{\{s_{k-2}\}} \sum_{\{\hat{d}_{k-1}\}} P(\hat{d}_k, s_{k-1}, x_k | \hat{d}_{k-1}, s_{k-2}, \mathbf{x_{k-1}}) P(\hat{d}_{k-1}, s_{k-2}, \mathbf{x_{k-1}}) & (10) \\ &= \sum_{\{s_{k-2}\}} \sum_{\{\hat{d}_{k-1}\}} P(\hat{d}_k, s_{k-1}, x_k | \hat{d}_{k-1}, s_{k-2}) P(\hat{d}_{k-1}, s_{k-2}, \mathbf{x_{k-1}}) & (11) \end{aligned}$$

In the simplification step above, $\mathbf{x_{k-1}}$ is dropped as a dependent condition due to the assumption that the received samples are independent of each other. In (11), the second term is a recursive expression to the probabilities sought by (9) - replacing index k with $k-1$. Hence, the procedure described from (9) to (11) has provided an iteration algorithm to find the forward probabilities. The remaining work, which is much more substantial, on the forward probabilities is to find the conditional probabilities, the first term, in (11).

$$\begin{aligned}
&P(\hat{d}_k, s_{k-1}, x_k | \hat{d}_{k-1}, s_{k-2}) \\
&\qquad = P(x_k | \hat{d}_k, s_{k-1}, \hat{d}_{k-1}, s_{k-2}) \\
&\qquad\quad P(s_{k-1} | \hat{d}_k, \hat{d}_{k-1}, s_{k-2}) \\
&\qquad\quad P(\hat{d}_k | \hat{d}_{k-1}, s_{k-1}) \\
&\qquad = P(x_k | \hat{d}_k, s_{k-1}) \\
&\qquad\quad P(s_{k-1} | \hat{d}_{k-1}, s_{k-2}) P(\hat{d}_k)
\end{aligned}$$

To get the final simplification shown above, there are three definitions applied:

1. s_{k-1} is not dependent on $\hat{d}_k$. This means that bits at index k correspond to propagation *into* the state at k.
2. $P(s_k | \hat{d}_k, s_{k-1}) = 1$ because the trellis must be a known structure.
3. $\hat{d}_k$ is not dependent on past bits or past trellis states and for maximum message capacity, $P(\hat{d}_k) = \frac{1}{2}$.

So,

$$P(\hat{d}_k, s_{k-1}, x_k | \hat{d}_{k-1}, s_{k-2}) = \frac{1}{2} P(x_k | s_{k-1}) \tag{12}$$

where given $\hat{d}_{k-1}, s_{k-2}$, the next state s_{k-1} is known or equivalent in information. Substituting (12) back to (9) require the summation to change to the entire set of s_{k-1}. By entire set, that means only the valid states from $k-1$ the given code trellis would allow. The valid states are inferred conditions known from the code and so will be just written as

$$P(s_k, \mathbf{x_k}) = \sum_{\{s_{k-1}\}} \frac{1}{2} P(x_k | s_{k-1}) P(s_{k-1}, \mathbf{x_{k-1}}) \tag{13}$$

where 2-tuple of $(\hat{d}_k, s_{k-1})$ is replaced with s_k for any corresponding index.

(12) is general in the sense that any code rate could be expressed by x_k. In the example here, the FEC from LTE specifications is used, which has a rate 1/3. Therefore, x_k is a 3-tuple written as $(x_k^{(0)}, x_k^{(1)}, x_k^{(2)})$. $x_k^{(0)}$ is the data bit in the rate 1/3 systematic code of LTE. $x_k^{(1)}$ and $x_k^{(2)}$ are the first and second parity bits, respectively.

Applying the axiom of probability on $P(x_k | s_{k-1})$,

$$\begin{aligned}
P(x_k | s_{k-1}) &= P(x_k^{(0)}, x_k^{(1)}, x_k^{(2)} | s_{k-1}) \\
&= P(x_k^{(0)} | s_{k-1}, x_k^{(1)}, x_k^{(2)}) P(x_k^{(1)} | s_{k-1}, x_k^{(2)}) \\
&\quad P(x_k^{(2)} | s_{k-1})
\end{aligned}$$

Since the data bit doesn't dependent on any of the parity bits and the parity bits don't dependent on each other, the simplified result is

$$P(x_k | s_{k-1}) = P(x_k^{(0)} | s_{k-1}) P(x_k^{(1)} | s_{k-1}) P(x_k^{(2)} | s_{k-1}) \tag{14}$$

Given the channel is AWGN then,

$$P(x_k^{(i)} | s_{k-1}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x_k^{(i)} - \hat{d}_k^{(i)})^2}{2\sigma^2} \right) \tag{15}$$

for $i = 0, 1, 2$, and d_{k-1} is determined by the transition implied by the previous state s_{k-1} .

Substituting (15) into (14) and expanding the exponentials,

$$a \exp\left(\frac{-\left(x_k^{(0)}\right)^2 + 2x_k^{(0)}\hat{d}_k^{(0)} - \left(d_k^{(0)}\right)^2}{2\sigma^2}\right) a \exp\left(\frac{-\left(x_k^{(1)}\right)^2 + 2x_k^{(1)}\hat{d}_k^{(1)} - \left(d_k^{(1)}\right)^2}{2\sigma^2}\right)$$
$$a \exp\left(\frac{-\left(x_k^{(2)}\right)^2 + 2x_k^{(2)}\hat{d}_k^{(2)} - \left(d_k^{(2)}\right)^2}{2\sigma^2}\right)$$

where $a = \frac{1}{\sqrt{2\pi\sigma^2}}$. Furthermore, $\left(\hat{d}_k^{(0)}\right)^2 = 1$ (or any arbitrary constant depending on notation. 1 is used for simplicity) so the above equation reduces to

$$\begin{aligned} P(x_k|s_{k-1}) &= \\ & a^3 \exp\left(\frac{-\left(x_k^{(0)}\right)^2 - \left(x_k^{(1)}\right)^2 - \left(x_k^{(2)}\right)^2}{2\sigma^2}\right) \\ & \exp\left(\frac{x_k^{(0)}\hat{d}_k^{(0)} + x_k^{(1)}\hat{d}_k^{(1)} + x_k^{(2)}\hat{d}_k^{(2)}}{\sigma^2}\right) \\ & \exp\left(\frac{-3}{2\sigma^2}\right) \end{aligned}$$

Further simplification is possible for the purpose of estimation. The reason is that the terms $x_k^{(i)}$, σ, and a are given terms which doesn't change with new estimates. The only terms that are of interest for estimation are $d_k^{(i)}$. So further simplification yields

$$P(x_k|s_{k-1}) = b \exp\left(x_k^{(0)}\hat{d}_k^{(0)} + x_k^{(1)}\hat{d}_k^{(1)} + x_k^{(2)}\hat{d}_k^{(2)}\right) \tag{16}$$

where

$$b = a^3 \exp\left(\frac{-\left(x_k^{(0)}\right)^2 - \left(x_k^{(1)}\right)^2 - \left(x_k^{(2)}\right)^2}{2\sigma^2}\right) \exp\left(\frac{-3}{2\sigma^2}\right) \exp\left(\frac{1}{\sigma^2}\right)$$

At this point, the forward part of the estimation is not yet complete because from (14) to (16), the trellis decoding has not considered the code as concatenated coding. With parallel concatenated codes, or turbo codes, the interleaver that separates the parallel codes isn't part of the trellis structure of each *constituent* code. Therefore, the parity bit from the *constituent* encoder before the interleaver shouldn't be mixed with the parity bit from the *constituent* encoder after the interleaver. That is, in (16) $d_k^{(1)}$ can't appear together with $d_k^{(2)}$. To have the forward estimation ready for turbo decoding, (16) must be specified for before and after the interleaver. For the forward estimate before the interleaver, (16) becomes

$$P_1(x_k|s_{k-1}) = b_1 \exp\left(x_k^{(0)}\hat{d}_k^{(0)} + x_k^{(1)}\hat{d}_k^{(1)}\right) \tag{17}$$

where

$$b_1 = a^3 \exp\left(\frac{-\left(x_k^{(0)}\right)^2 - \left(x_k^{(1)}\right)^2}{2\sigma^2}\right) \exp\left(\frac{-3}{2\sigma^2}\right) \exp\left(\frac{1}{\sigma^2}\right)$$

And the forward estimate after the interleaver becomes

$$P_2(x_k|s_{k-1}) = b_2 \exp\left(x_k^{(0)} d_k^{(0)} + x_k^{(2)} \hat{d}_k^{(2)}\right) \tag{18}$$

where

$$b_2 = a^3 \exp\left(\frac{-\left(x_k^{(0)}\right)^2 - \left(x_k^{(2)}\right)^2}{2\sigma^2}\right) \exp\left(\frac{-3}{2\sigma^2}\right) \exp\left(\frac{1}{\sigma^2}\right)$$

These forward terms should only be computed for trellis transitions that are valid for the given code. Transitions that are not part of the trellis could be skipped entirely.

For the backward trellis, start with

$$P(\mathbf{x'}_{k+1}|\hat{d}_k, s_{k-1}) = P(x_{k+1}, \mathbf{x'}_{k+2}|s_k) \tag{19}$$

$$P(x_{k+1}, \mathbf{x'}_{k+2}|s_k) = \sum_{\{\hat{d}_{k+1}\}} P(\hat{d}_{k+1}, x_{k+1}, \mathbf{x'}_{k+2}|s_k)$$

$$= \sum_{\{\hat{d}_{k+1}\}} P(\mathbf{x'}_{k+2}|\hat{d}_{k+1}, x_{k+1}, s_k) P(\hat{d}_{k+1}, x_{k+1}|s_k)$$

Since the forward estimation used the assumption that received samples are uncorrelated, the backward estimate can also use this assumption. Thus, the above equation reduces to

$$(19) = \sum_{\{\hat{d}_{k+1}\}} P(\mathbf{x'}_{k+2}|s_{k+1}) P(\hat{d}_{k+1}, x_{k+1}|s_k) \tag{20}$$

In (20) the first term in the summation is similar to the right hand side of (19) and could be substituted using the expansions shown above in a recursive way. This recursion yields the iterations for the backward estimates. Also note that the summation term is over $\hat{d}_{k+1}$ which the conditional probability is dependent on but simplified with s_{k+1}. Implementation of these equations has to account for the summation and not move that outside of summation. The second term in (20) can be simplified further.

$$P(\hat{d}_{k+1}, x_{k+1}|s_k) = P(x_{k+1}|\hat{d}_{k+1}, s_k) P(\hat{d}_{k+1}|s_k) \tag{21}$$

Similar to the derivations in the forward estimation part, $P(\hat{d}_{k+1}|s_k) = P(\hat{d}_{k+1}) = 1/2$ for random transmitted bits. So finally,

$$P(\hat{d}_{k+1}, x_{k+1}|s_k) = \frac{1}{2} P(x_{k+1}|s_{k+1}) \tag{22}$$

The above equation is very similar to (12) but the indices are slightly different. Being the backward estimation here, the probability calculations are for branches between s_k and s_{k+1}. So (22) implies that the state s_k would be known for each estimate of $\hat{d}_{k+1}$. Using these definitions, equations (17) and (18) can be used to make the same computations for the backward estimates.

While the BCJR algorithms can be computed as shown, the multiplication of the recursive calculations can be computationally intensive. Applying the same idea as the log-likelihood probability to transform multiply operations to add operations, the recursive calculations can also be transformed to the log domain. To show how that's done, take the example of the forward estimates before the interleaver in (17). Substituting (17) to (11) yields a formula that in structure looks like the following.

$$f_k = f_{k-1} b \exp(z_k) \tag{23}$$

where f_k is the symbol for the probability in (9), and z_k is the exponential arguments in (17).

From (23), along with recursive multiply operations, there are also exponential terms. Numerically, the exponential calculations can cause dynamic range overflow if the arguments differ even by reasonable magnitudes. Thus, the equation in the form of (23) is not numerically stable as well as having multiplications. Therefore, a numerical stable implementation would compute the $\log(f_k)$.

$$\log(f_k) = \log(f_{k-1}) + \log(b) + z_k \tag{24}$$

Equation (24), however, can't be directly substituted into (9) due to the log form. Transforming to the linear domain would yield $\exp(\log(f_k))$ but even this doesn't fit well into (9) due to the summation of exponential terms. Substituting (9) to (8) could be written in another way. The alternative equation would yield the same decision metric as (8) by transforming

$$\log\left(\sum P\right) \rightarrow \sum \log\left(P\right) \tag{25}$$

where P is some probability that represent a term in (8). The right arrow indicates that the transformation yield the same decision metric. The proof for (25) is that

$$\log\left(\sum P_1\right) > \log\left(\sum P_0\right) \iff \sum \log(P_1) > \sum \log(P_0) \tag{26}$$

if $P_1 > 0$ and $P_0 > 0$.

Using (26) transformation, (9) can be substituted into (8) to yield

$$\sum \log\left(P_1\right) - \sum \log\left(P_0\right) \tag{27}$$

where P_1 represents the numerator of (8) and P_0 represents the denominator. The summation must be over the set of all states.

From the new expression, the log-likelihood could be written as

$$\begin{aligned} L(\hat{d}) &= \sum \log(f_k(s_k, \hat{d}_k = 1)) + \log(g_k(s_k, \hat{d}_k = 1)) - \\ &\quad \left(\sum \log(f_k(s_k, \hat{d}_k = 0)) + \log(g_k(s_k, \hat{d}_k = 0))\right) \end{aligned}$$

where g_k is the backward probabilities in (19). The general equations for f_k and g_k are

$$\log(f_k(s_k, \hat{d}_k)) = \log \sum_{\{\hat{d}_{k-1}\}} (f_{k-1}(s_{k-1}, \hat{d}_{k-1})) + \log(b) + z_k(s_k, \hat{d}_k) + \log\frac{1}{2} \tag{28}$$

$$\log(g_k(s_k, \hat{d}_k)) = \log \sum_{\{\hat{d}_{k+1}\}} (g_{k+1}(g_{k+1}, \hat{d}_{k+1})) + \log(b) + z_k(s_k, \hat{d}_k) + \log\frac{1}{2} \tag{29}$$

The constants $\log(b)$ and $\log(1/2)$ can be dropped because they will cancel each other out in (28). With the z_k term, the transformation suggested by (25) can't be applied again. So now, the problem seems that the recursive computations of the terms should be in log form but due to a summation, the recursive terms are in the original domain. Take only the general forward equation as an example, it may be re-written as

$$\log(f_k(s_k, \hat{d}_k)) = \log \sum_{\{\hat{d}_{k-1}\}} \exp\left(\log(f_{k-1}(s_{k-1}, \hat{d}_{k-1}))\right) \tag{30}$$

For the LTE example used thus far, $\hat{d}_k \in 1, 0$ therefore

$$\log(f_k(s_k, \hat{d}_k)) = \log\left(\exp\left(\log(f_{k-1}(s_{k-1}, \hat{d}_{k-1} = 0))\right) + \exp\left(\log(f_{k-1}(s_{k-1}, \hat{d}_{k-1} = 1))\right)\right) \tag{31}$$

Equation (31) has two notable features. First, the form of the equation can be written in another way that is more numerically stable. Second, for any state s_k on the left side of the equation, and a given $\hat{d}_{k-1}$, the state s_{k-1} is a known state. So in implementation of (31), a common routine should exist to quickly yield s_{k-1} given $\hat{d}_{k-1}$ and s_k.

The numerically stable form to write (31) is sometimes referred to as the Jacobi form. It's shown below.

$$\log\left(\exp A + \exp B\right) = \log\left(\exp(A - B) + 1\right) + B \tag{32}$$

where $B \geq A$ and A and B represent each of the $\log f_{k-1}$ terms in (31). In this form, the term that is larger must be designated as B and would ensure that the exponential function has a negative argument. Thus, in implementation there is never any large position numbers as argument to exp() function. Sometimes to ease implementation, finding the maximum of A or B is sufficient to approximate (32). This is called the max-log implementation.

Following the similar derivation from (30) to (32), the similar equations can be found for the backward equations.

With the BCJR algorithm explained, the iterative or turbo decoding procedures can now be explained. To be continued...