Turbo coding

Kalle Ruttik

Communications Laboratory
Helsinki University of Technology

April 24, 2007

Introduction

Channel coding theorem suggest that a randomly generated code with appropriate distribution is likely a good code if its block length is high.
Problem: Decoding

- In case of long block lengths the codes without a structure are difficult to decode.

A fully random codes can be avoided. The codes whose spectrum resembles spectrum of a random code are good codes.
Such random like codes can be generated by using interleaver.

Interleaver performs permutation of the bit sequence.

Permutation can be either of information bits sequence or of the parity bits.

Parallel concated convolutional codes (PCCC)

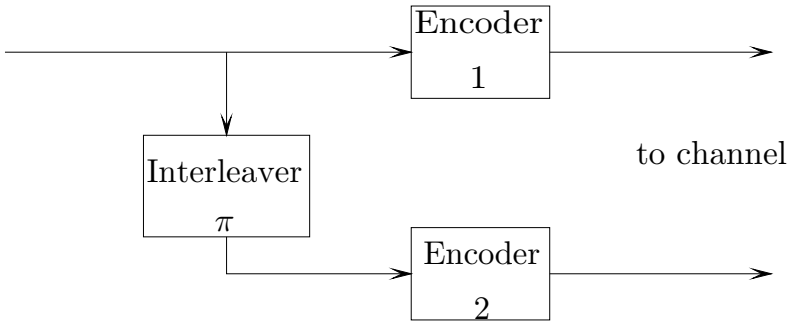


Figure: Encoder

Decoding turbo codes

- A parallel-concatenated convolutional code cannot be decoded by a standard serial dynamic programming algorithm.
  - The number of states considered in trellis evaluation of two interleaved code is squared compared to the forward-backward algorithm on one code.
  - Changing a symbol in one part of the turbo-coded codeword will affect possible paths in this part in one code and also the distant part in other code where this bit is "interleaved".
  - Optimal path in one constituent codeword does not have to be optimal path in the other codeword.

Berrou approach

- Calculate the likelihood of each bit of the original dataword of being 0 or 1 accordingly to the trellis of the first code.
- The second decoder uses the likelihood from the first decoder to calculate the new probability of the received bits but now accordingly to the received sequence of the second coder $y^{(2)}$.
- Bit estimate from the second decoder is feed again into first decoder.
- Instead of serially decoding each of the two trellises we decode both of them in parallel fashion.

Decoding turbo codes

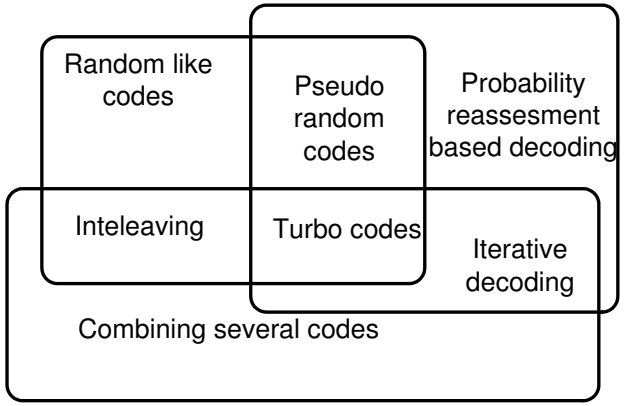


Figure: The ideas influencing evolution of turbo coding. Accordingly to fig. 1 from Battail97.

Code as a constraint

- The bit estimates calculated based on the coder structure are *aposteriori* probabilities of the bit constrained by the code.
- Contraint means that among of all possible bit sequences only some are allowed - they are possible codewords. The codewords limit the possible bit sequecnes.
- The aposteriori probability is calculated over the probabilities of possible codewords.
- The rule of the conditional probability

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

where A correspond to a event that a certain bit in the codeword is 1 (or zero). B requires that the bit sequence is allowable codeword.

Example: repetition code

- We have three samples c_1, c_2, c_3.
- If to take the samples one by one they can be either zero or one.
- We have additional information: the samples are generated as a repetition code.
- Let denote the valid configurations as $S = \{(c_1, c_2, c_3)\}$.
- The set of possible codewords (the constraint set) is $S = \{(0,0,0), (1,1,1)\}$.

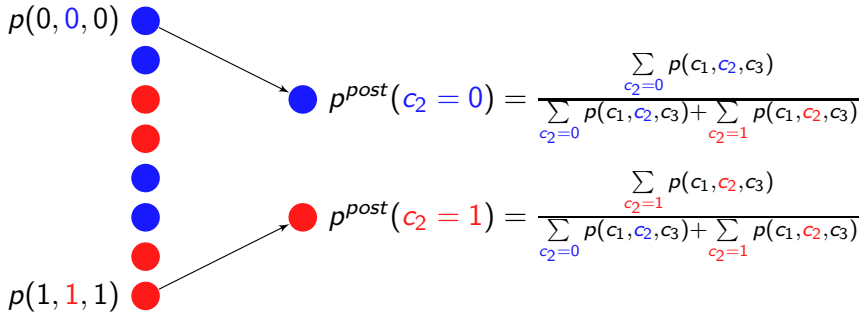
- In our case the *a posterior* probability for the sample $c_2 = 1$ is

$$p_2^{post} = \frac{\sum_{\substack{(c_1,c_2,c_3)\in S \\ c_2=1}} P(c_1, c_2, c_3)}{\sum_{(c_1,c_2,c_3)\in S} P(c_1, c_2, c_3)}$$

In the numerator is summation over all configurations in S such that $c_2 = 1$, in the denominator is the normalization, the sum of all the probabilities of all configurations in S.

Example: Repetition code

$p(c_1, c_2, c_3)$



In the repetition code is only one possible codeword with $c_2 = 1$ and onw codeword with $c_2 = 0$

- If the prior probabilities are independent the joint probability can be factorised

$$P(c_1, c_2, c_3) = P(c_1) P(c_2) P(c_3)$$

- The values for the prior probabilities could be acquired by measurements.
  For example if we are handling channel outputs them $p(c_k) = p(y_k|x_k)$
- A numerical excample: We have observed the samples and concluded that the samples have values 1 with the following probabilities
  $P(c_1 = 1) = \frac{1}{4}, P(c_2 = 1) = \frac{1}{2}, P(c_3 = 1) = \frac{1}{3}$,
  where c_i stands for the i-th sample and $i = 1, 2, 3$.
  What is the probability that the second bit is one?

- The probability of the second sample being one is

$$p_2^{post} = \frac{p_1 p_2 p_3}{p_1 p_2 p_3 + (1 - p_1)(1 - p_2)(1 - p_3)}$$

- In numerical values

$$p_2^{post} = \frac{\frac{1}{4} \cdot \frac{1}{2} \cdot \frac{1}{3}}{\frac{1}{4} \cdot \frac{1}{3} \cdot \frac{1}{3} + \left(1 - \frac{1}{4}\right)\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right)} = 0.1429$$

- The probability of $p_2(c_2 = 0)$

$$\begin{aligned} p_2(c_2 = 0) &= \frac{\left(1-\frac{1}{4}\right)\left(1-\frac{1}{2}\right)\left(1-\frac{1}{3}\right)}{\frac{1}{4}\cdot\frac{1}{2}\cdot\frac{1}{3}+\left(1-\frac{1}{4}\right)\left(1-\frac{1}{2}\right)\left(1-\frac{1}{3}\right)} \\ &= 0.8571 = 1 - p_2^{post} \end{aligned}$$

- By using likelihood ratio we can simplify further

$$\begin{aligned}\frac{p_2^{post}}{\left(1-p_2^{post}\right)} &= \frac{\sum_{\substack{(c_1,c_2,c_3)\in S \\ c_2=1}} P(c_1,c_2,c_3)}{\sum_{\substack{(c_1,c_2,c_3)\in S \\ c_2=1}} P(c_1,c_2,c_3)} \\ &\Rightarrow \frac{p_1\cdot p_2\cdot p_3}{(1-p_1)\cdot(1-p_2)\cdot(1-p_3)} \\ &= \frac{p_1}{(1-p_1)}\cdot\frac{p_2}{(1-p_2)}\cdot\frac{p_3}{(1-p_3)}\end{aligned}$$

- In the logarithmic domain

$$\ln\frac{p\left(c_1=1\right)p\left(c_2=1\right)p\left(c_3=1\right)}{p\left(c_1=0\right)p\left(c_2=0\right)p\left(c_3=0\right)}$$

$$=\ln\frac{p\left(c_1=1\right)}{p\left(c_1=0\right)}+\ln\frac{p\left(c_2=1\right)}{p\left(c_2=0\right)}+\ln\frac{p\left(c_3=1\right)}{p\left(c_3=0\right)}$$

$$L^{post}(c_3)=L(c_1)+L(c_2)+L(c_3)$$

- Probability 0.5 indicates that nothing is known about the bit.

$$L(c_2)=0$$

- Even if we do not know aything about the bit but we know probabilties for the other bits we can calculate the aposteriori probability for the unknown bit.

$$L^{post}(c_2)=L(c_1)+L(c_3)$$

- The posteriori probability calculation for a bit can be separated into two parts:
  - part describing prior probability
  - part impacted by the constraint imposed by the code. This later part is calculated only based on the probabilities of other bits.

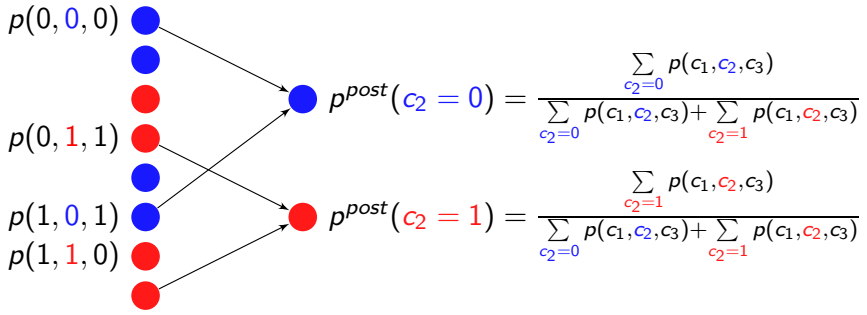
- Assume now that there can be even number of ones among the three samples $S=\{(0,0,0),(1,1,0),(1,0,1),(0,1,1)\}$
- The first two bits are either 0 or 1 the third bit is calculated as *XOR* of first two bits.
- Assume the measured probability for the second bit is 0.5
- The posterior probability of the second sample is

$$p_2^{post}=\frac{p_1\left(1-p_3\right)+\left(1-p_1\right)p_3}{p_1\cdot p_3+p_1\cdot\left(1-p_3\right)+\left(1-p_1\right)\cdot p_3+\left(1-p_1\right)\cdot\left(1-p_3\right)}$$

- The probability that the second sample is 1 is given by the probability that exactly one of the other two samples is 1.
- The probability that the second sample is 0 is given by the probability that both other samples are 0 or both of them are 1.

Example: Repetition code

$p(c_1,c_2,c_3)$



- Likelihood ratio for the second sample *aposterior* probability is

$$\frac{p^{post}(c_2=1)}{p^{post}(c_2=0)} = \frac{p_2\left(p_1(1-p_3)+(1-p_1)p_3\right)}{(1-p_2)\left(p_1p_3+(1-p_1)(1-p_3)\right)}$$

$$= \frac{\frac{1}{2}\left(\frac{1}{4}\left(1-\frac{1}{3}\right)+\left(1-\frac{1}{4}\right)\frac{1}{3}\right)}{\left(1-\frac{1}{2}\right)\left(\frac{1}{4}\frac{1}{3}+\left(1-\frac{1}{4}\right)\left(1-\frac{1}{3}\right)\right)} = 0.595$$

Parity check in log domain

- In logarithmic domain can be separated the prior for the bit and information from the other bits.

$$\log\left(\frac{p_2(c_2=1)\cdot p\left((c_1\oplus c_3)=1\right)}{p_2(c_2=0)\cdot p\left((c_1\oplus c_3)=0\right)}\right) =$$

$$\ln\left(\frac{p_2(c_2=1)}{p_2(c_2=0)}\cdot\frac{p_1(c_1=1)p_3(c_3=0)+p_1(c_1=0)p_3(c_3=1)}{p_1(c_1=0)p_3(c_3=0)+p_1(c_1=1)p_3(c_3=1)}\right) =$$

$$L_2(c_2)+\ln\left(\frac{p_1(c_1=1)p_3(c_3=0)+p_1(c_1=0)p_3(c_3=1)}{p_1(c_1=0)p_3(c_3=0)+p_1(c_1=1)p_3(c_3=1)}\right)$$

Probabilities in log domain

- Here we give the probability calculation folmulas for the binary code, GF(2).
- The log-likelihood ratio (LLR) of c is

$$\begin{aligned} L(c) &= \ln\frac{p(c=1)}{p(c=0)} = \ln\frac{p(c=1)}{1-p(c=1)} \\ p(c=1) &= \frac{e^{L(c)}}{1+e^{L(c)}} = \frac{1}{1+e^{-L(c)}} \Rightarrow \\ p(c=0) &= 1-p(c=1) = \frac{1}{1+e^{L(c)}} = \frac{e^{-L(c)}}{1+e^{-L(c)}} \end{aligned}$$

Incorporating proabilities from different encoders

- Often we have two or more independent ways to calculate the aposteriori probability of the bit.
- The bit estimates from different sources are similar to repetition code. All the estiamtes have to have the same bit value.
- Because all the estiamtes have to be either 0 or 1 in log domain we can simple sum together the loglikelihood ratios from different estimations.

Single Parity Check Product Code (Example)

- SPC product code - a simple example of a concatenated code
- Two separate coding steps - horizontal, vertical

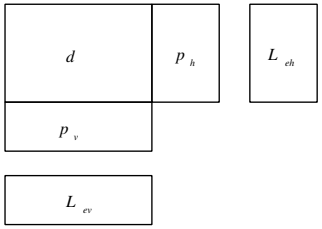


Figure: Two dimensional product code

$k_1 \times k_2$ data array d; $n_2 - k_2$ parity bits p_h; $n_1 - k_1$ parity bits p_v, L_{eh}, L_{ev} stand for the extrinsic LLR values learned from the horizontal and vertical decoding steps.

Numerical example

d_1	d_2	d_3	d_4	p_{1h}	p_{2h}	p_{1v}	p_{2v}
$+1$	$+1$	$+1$	-1	$+1$	-1	$+1$	-1
0.25	2.0	5.0	1.0	1.0	-1.5	2.0	-2.5

d_1	d_2	p_{1h}
d_3	d_4	p_{2h}
p_{1v}	p_{2v}	

$\Rightarrow$

$+$	$+$	$+$
$+$	$-$	$-$
$+$	$-$	

$\Rightarrow$

$L_c(x_1) = 0.25$	$L_c(x_2) = 2.0$	$L_c(x_{12}) = 1.0$
$L_c(x_3) = 5.0$	$L_c(x_4) = 1.0$	$L_c(x_{34}) = -1.5$
$L_c(x_{13}) = 2.0$	$L_c(x_{24}) = -2.5$	

Decoding Algorithm

1. Set prior information $p(\hat{d}) = 0.5$ for all d $L(\hat{d}) = 0$.
2. Sum the observed probability and prior probability $L_c(d_\#) + L(d_\#)$.
3. Decode horizontally. Obtain the bit probabilities based on the constraint posed by the horisontal code. The result is called horizontal extrinsic information.
   - The parity bit is generated by the xor of information bits in the horisontal line.
   - The extrinsic information can be generated as the *aposteriori* probability calculation accordingly to parity check. For example

$$L_h^{extr}(d_1) = \ln\left(\frac{p(d_2=1)p(p_{1h}=-1) + p(d_2=-1)p(p_{1h}=1)}{p(d_2=-1)p(p_{1h}=-1) + p(d_2=1)p(p_{1h}=1)}\right)$$

4. Set $L(\hat{d}_1) = L_h^{extr}(d_1)$.
5. Combine the *aposteriori* horisontal and priori information for each bit. For example for the bit 1

$$L^{combined}(d_1) = L_c(d_1) + L(\hat{d}_1)$$

$$p^c(d_1 = 1) = \frac{e^{L^{combined}(d_1)}}{\left(1 + e^{L^{combined}(d_1)}\right)}$$

6. Decode vertically. In computations instead of p use p^c. Obtain the vertical extrinsic information for each bit. For example

$$L_v^{extr}(d_1) = \ln\left(\frac{p^c(d_3=1)p(p_{1v}=-1) + p^c(d_3=-1)p(p_{1v}=1)}{p^c(d_3=-1)p(p_{1v}=-1) + p^c(d_3=1)p(p_{1v}=1)}\right)$$

Decoding example ...

7. If the interations have not finished
   - combine the information of the *aposteriori* vertical $L_h^{extr}(d_\#)$ and priori information $L(d_\#)$ for each bit.
   - go back to stage 2.

   else
   - Combine all the information for the bit the priori *aposteriori* vertical and horisontal.
     $L^d(d_\#) = L_c(d_\#) + L_v^{extr}(d_\#) + L_h^{extr}(d_\#)$
   - Compare the likelihood ratio of the bit with the decision level (0).

Decoding example ...

The soft output for the received signal corresponding to data d_i

$$L(\hat{d}_i) = L_c(x_i) + L(\hat{d}_i) + L_h^{extr}(d)$$

Decode horizontally

$$\begin{aligned} L_h^{extr}(d_1) &= \ln\left(\frac{p(d_2=1)p(p_{1h}=-1)+p(d_2=-1)p(p_{1h}=1)}{p(d_2=-1)p(p_{1h}=-1)+p(d_2=1)p(p_{1h}=1)}\right) \\ &= 0.74 = newL(\hat{d}_1) \end{aligned}$$

$$L_h^{extr}(d_2) = +0.12 = newL(\hat{d}_2)$$

$$L_h^{extr}(d_3) = -0.60 = newL(\hat{d}_3)$$

$$L_h^{extr}(d_4) = -1.47 = newL(\hat{d}_4)$$

+0.25	+2.0
+5.0	+1.0

+

+0.74	+0.12
−0.60	−1.47

after 1st horizontal decoding

Decode vertically

$$L_v^{extr}(d_1) = +0.33 = newL(\hat{d}_1)$$

$$L_v^{extr}(d_2) = +0.09 = newL(\hat{d}_2)$$

$$L_v^{extr}(d_3) = -0.36 = newL(\hat{d}_3)$$

$$L_v^{extr}(d_4) = -0.26 = newL(\hat{d}_4)$$

+0.33	+0.09
+0.36	−0.26

Extrinsic information after 1st vertical decoding

Soft output after 1st iteration $L(\hat{d}) = L_c(x) + L_{eh}(d) + L_{ev}(d)$

+0.25	+2	+	+0.74	+0.12	+	+0.33	+0.09	=	+1.31	+2.20
+5.0	+1		−0.60	−1.47		+0.36	−0.26		+4.75	−0.74

We can see an iterative process:

1. Decode first code and calculate extrinsic information for each bit.
   - In first iteration the information from other code is zero.
2. Decode the second code by using extrinsic information from the first decoder.
3. Return to the first step by using the extrinsic information from the second decoder.

Parallel concated convolutional codes (PCCC)

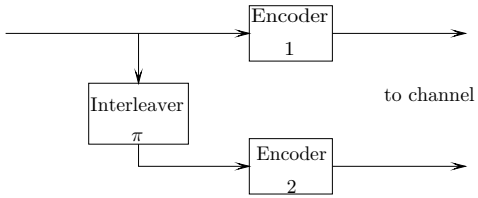


Figure: Encoder

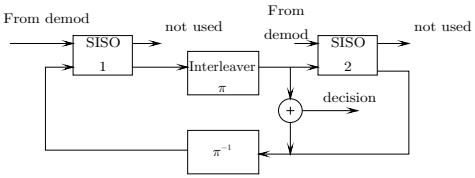


Figure: Decoder

Parallel concated convolutional codes (PCCC)

- Encoder contains two or more systematic convolutional encoders
- The constituent encoders code the same interleaved data stream
- The systematic bits are transmitted only once
- In reciever the extrinsic information is calculated for the information bit based on one constituent code and feed to the decoder of other code

Serial concated convolutional codes (SCCC)

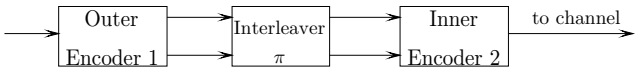


Figure: Encoder

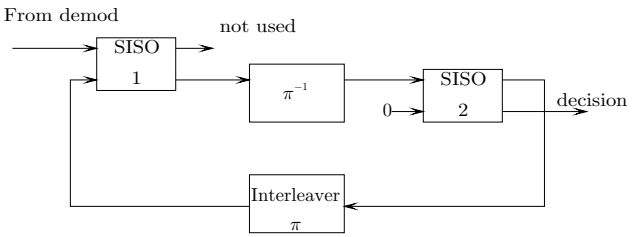
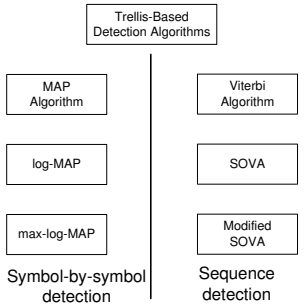


Figure: Decoder

Serial concated convolutional codes (SCCC)

- Code is formed by concatenating two encoders
  - The output coded bit stream from the outer encoder is interleaved and feed to the inner encoder
- The decoder
  - The inner decoder calculates the loglikelihoods of information symbols at the output of the inner decoder and deinterleaver them
  - The outer decode is decoded and loglikelihoods for the coded bits are calculated
  - The coded bits loglikelihoods are interleaved and feed back to the inner decoder
  - The decision are made after the decoding iterations on the loglikelihoods of the information bits at the output of the outer decoder

Algorithms for Iterative (Turbo) Data Processing



Requirements

Accept soft-inputs in the form of a priori probabilities or log-likelihood ratios

Produce APP for output data

Soft-Input Soft-Output

- MAP: Maximum A Posteriori (symbol-by-symbol)
- SOVA: Soft Output Viterbi Algorithm

MAP algorithm

- MAP algorithm operates in probability domain.
- The probability of all codewords passing some particular edge from initial state i to final state j at stage k is

$$p_k^A(b_{k,i,j}) = \frac{1}{p(Y_1^N)} \sum_{(b_{k,i,j}} A_{k-1,i} \cdot M_{k,i,j} \cdot B_{k,j}$$

- When probablity is expressed by loglikelihood value we have to deal with numbers in very large range. (overflows in computers).

Simplification of MAP: Log-MAP algorithm

- Log-MAP algorithm is a transformation of MAP into logarithmic domain.

Modification to MAP algorithm

- The MAP algorithm logarithmic domain is expressed with replaced computations
  - Multiplication is converted to addition.
  - Addition is converted to $\max*(\cdot)$ operation.

$$max*(x,y) = \log\left(e^x + e^y\right) = \max(x,y) + \log\left(1 + e^{-|x-y|}\right)$$

- The terms for calculating the probabilities in the trellis are converted

$$\begin{aligned} \alpha_{k,i} &= \log(A_{k,i}) \\ \beta_{k,j} &= \log(B_{k,j}) \\ \gamma_{k,i,j} &= \log(M_{k,i,j}) \end{aligned}$$

Modification to MAP algorithm

- The complete logMAP algorithm is

$$\begin{aligned} L(b_k) &= \log \sum_{b_k=1} A_{k-1,i} \cdot M_{k,i,j} \cdot B_{k,j} \\ &\quad - \log \sum_{b_k=0} A_{k-1,i} \cdot M_{k,i,j} \cdot B_{k,j} \\ &= \max_{b_k=1}* \left(\alpha_{k-1,i} + \gamma_{k,i,j} + \beta_{k,j}\right) \\ &\quad - \max_{b_k=0}* \left(\alpha_{k-1,i} + \gamma_{k,i,j} + \beta_{k,j}\right) \end{aligned}$$

$$\alpha_{k,i} = \log\left(\sum_{i_1} A_{k-1,i_1} \cdot M_{k,i_1,i}\right)$$

$$\beta_{k,j} = \log\left(\sum_{j_1} M_{k+1,j,j_1} \cdot B_{k+1,j_1}\right)$$

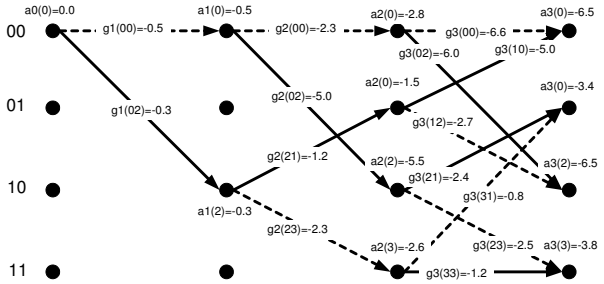
Max-Log-MAP decoding Algorithm

- In summation of probabilities in Log-MAP algorithm we are using $max*(\cdot)$ operation.
- The $max*(\cdot)$ requires to convert LLR value into exponential and after adding 1 to move back into log domain.
- Simplifications
  - We can replace $\log\left(1+e^{-|x-y|}\right)$ by a lookup table.
  - We can skip the term $\Rightarrow$ Max-Log-Map.

Max-Log-MAP decoding Algorithm

$$\begin{aligned}
\alpha_{k,i} &= \log\left(\sum_{i_1} A_{k-1,i_1}\cdot M_{k,i_1,i}\right) = \log\left(\sum_{i_1} e^{\alpha_{k-1,i_1}+\gamma_{k,i_1,i}}\right)\\
&= \max*\left(\forall_{i_1} e^{\alpha_{k-1,i_1}+\gamma_{k,i_1,i}}\right) \approx \max\left(\forall_{i_1} e^{\alpha_{k-1,i_1}+\gamma_{k,i_1,i}}\right)\\
&\Rightarrow \text{Max-Log-MAP}\\
\beta_{k,j} &= \log\left(\sum_{j_1} M_{k+1,j,j_1}\cdot B_{k+1,j_1}\right) = \log\left(\sum_{j_1} e^{\gamma_{k+1,j,j_1}+\beta_{k+1,j_1}}\right)\\
&= \max*\left(\forall_{j_1} e^{\gamma_{k+1,j,j_1}+\beta_{k+1,j_1}}\right) \approx \max\left(\forall_{j_1} e^{\gamma_{k+1,j,j_1}+\beta_{k+1,j_1}}\right)\\
&\Rightarrow \text{Max-Log-MAP}
\end{aligned}$$

Example: Max-Log-MAP approximation



$$\begin{aligned}
\alpha_{k,i} &= \log\left(\sum_{i_1} A_{k-1,i_1}\cdot M_{k,i_1,i}\right)\\
\alpha_{3,0} &= \log\left(e^{(\alpha_{2,0}+\gamma_{3,0,0})}+e^{(\alpha_{2,1,0}+\gamma_{3,1,0})}\right)\\
&\approx \max\left(e^{(-2.8-6.0)}+e^{(-1.5-2.7)}\right) = \max(-8.8,-6.5) = -6.5
\end{aligned}$$

Soft-output Viterbi algorithm

Two modifications compared to the classical Viterbi algorithm

- Ability to accept extrinsic information from other decoder
  - The path metric is modified to account the extrinsic information. This is similar to the metric calculation in Max-Log-MAP algorithm.
- Modification for generating soft outputs

SOVA

- For each state in the trellis the metric $M_{k,i,j}$ is calculated for both merging paths.
- The path with the highest metric is selected to be the survivor.
- For the state (at this stage) a pointer to the previous state along the surviving path is stored.
- The following information that is later used for calculating $L\left(b_k|\underline{y}\right)$ is stored:
  - The difference Δ_k^s between the discarded and surviving path.
  - The binary vector containing $\delta+1$ bits, indicating last $\delta+1$ bits that generated the discarded path.
- After Maximum Likely path is found the update sequences and metric differences are used to calculate $L\left(b_k|\underline{y}\right)$.

SOVA

Calculation of $L\left(b_k|\underline{y}\right)$.

- For each bit b_k^{ML} in the ML path we try to find the path merging with ML path that had compared to the b_k^{ML} in ML different bit value b_k at state k and this path should have minimal distance with ML path.
- We go trough $\delta+1$ merging paths that follow stage k i.e. the $\Delta_i^{s_i}$ $i=k...(k+\delta)$
- For each merging path in that set we calculate back to find out which value of the bit b_k generated this path.
- If the bit b_k in this path is not b_k^{ML} and $\Delta_i^{s_i}$ is less than current Δ_k^{min} we set $\Delta_k^{min}=\Delta_i^{s_i}$
- $L\left(b_k|\underline{y}\right) \approx b_k \min\limits_{\substack{i=k...k+\sigma \\ b_k^{ML}\neq b_k^i}} \Delta_i^{s_i}$

Comparison of the soft decoding algorithms

MAP

- The MAP algorithm is the optimal component codes decoder algorithm.
- It finds the probability of each bit b_k of either being $+1$ or -1 by summing the probabilities of all the codewords where the given bit is $+1$ and where the bit is -1.
- Complex.
- Because of the exponent in probability calculations in practice the MAP algrithm often suffers numerical stability problems.

LogMAP

- LogMAP is theorectically identical to MAP, the calculation only are made in logarithmic domain.
- Multiplications are replaced by addition and summation with $\max*(\cdot)$ operation.
- Numerical problems that occure in MAP are cirmcumvented.

Max-Log-MAP

- Similar to LogMAP but replaces the maxlog operation $(\max*(\cdot))$ with taking maximum.
- Because at each state in forward and backward calcualtions only the path with maximum value is considered the probabilities are not calculated over all the codewords.
  - In recursive calculation of α and β only the best transition is considered.
  - The algorithm gives the logarithm of the probability that only the most likely path reaches the state.
- In the MaxLogMAP $L\left(b_k|\underline{y}\right)$ is comparison of probability of most likely path giving $b_k = -1$ to the most likely path giving $b_k = +1$.
  - In calcualtions of loglikelihood ratio only two codewords are considered (two transitions): The best transition that would give $+1$ and the best transition that would give -1.
- MaxLogMAP performs worse than MAP or LogMAP

SOVA

- the SOVA algorithms founds the ML path.
  - The recursion used is identical to the one used for calcuating α in MaxLogMAP algorithm.
- Along the ML path hard decision on the bit b_k is made.
- $L\left(b_k|\underline{y}\right)$ is the minimum metric difference between the ML path and the path that merges with ML path and is generated with different bit value b_k.
  - In $L\left(b_k|\underline{y}\right)$ calcualtions accordingly to MaxLogMAP one path is ML path and other is the most likely path that gives the different b_k.
  - In SOVA the difference is calculated between the ML and the most likely path that merges with ML path and gives different b_k.
    This path but the other may not be the most likely one for giving different b_k.

SOVA

- The output of SOVA just more noisy compared to MaxlogMAP output (SOVA does not have bias).
- The SOVA and MaxLogMAP have the same output
  - The magnitude of the soft decisions of SOVA will either be identical of higher than those of MaxLogMAP.
  - If the most likely path that gives the different hard decision for b_k has survived and merges with ML path the two algorithms are identical.
  - If that path does not survive the path on what different b_k is made is less likely than the path which should have been used.

Algorithms complexity

Table: Comparison of complexity of different decoding algorithms

Operations	maxlogMAP	logMAP	SOVA
max-ops	$5 \times 2^M - 2$	$5 \times 2^M - 2$	$3(M+1) + 2^M$
additions	$10 \times 2^M + 11$	$10 \times 2^M + 11$	$2 \times 2^M + 8$
mult. by ± 1	8	8	8
bit comps			$6(M+1)$
look-ups		$5 \times 2^M - 2$	

M is the length of the code memory.

Table accordingly to reference [1]

Algorithms complexity ...

If to assume that each operation is comparable we can calculate the totat amount of operations per bit every algorithm demands for decoding one code in one iteration.

Table: Number of reguired operations per bit for different decoding algorithms

memory (M)	MaxLogMAP	LogMAP	Sova
2	77	95	55
3	137	175	76
4	257	335	109
5	497	655	166

References

1 P. Robertson, E. Villebrun, P. Hoeher, "Comparison of Optimal and Suboptimal MAP decoding algorithms", ICC, 1995 page 1009-1013.