SRE presentation : Convergence of iterative Decoding Methods

Sahasrajit Sarmasarkar
Guide: Prof Nikhil Karamchandani, Prof. Manoj Gopalkrishnan

Note: All material and figures in these slides have been taken from chapter 2, 3 and 6 of [1].

LDPC codes

Parity check matrix H is defined for a linear code such that $x.H^T = 0$ for every code word x. Typically we try to remove redundancies in the matrix H and have linearly independent rows.

LDPC codes

Parity check matrix H is defined for a linear code such that $x.H^T = 0$ for every code word x. Typically we try to remove redundancies in the matrix H and have linearly independent rows.
These codes are a class of linear codes where the parity check matrix has very few 1's.

LDPC codes

Parity check matrix H is defined for a linear code such that $x.H^T = 0$ for every code word x. Typically we try to remove redundancies in the matrix H and have linearly independent rows.
These codes are a class of linear codes where the parity check matrix has very few 1's.
This effectively translates to very few variables in each of the constraints on the variables in code-words.

LDPC codes

Parity check matrix H is defined for a linear code such that $x.H^T = 0$ for every code word x. Typically we try to remove redundancies in the matrix H and have linearly independent rows.
These codes are a class of linear codes where the parity check matrix has very few 1's.
This effectively translates to very few variables in each of the constraints on the variables in code-words.
For example consider matrix $H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$ and consider the every code-word to be of form $\mathrm{x} = (\begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \end{bmatrix})$

LDPC codes

Parity check matrix H is defined for a linear code such that $x.H^T = 0$ for every code word x. Typically we try to remove redundancies in the matrix H and have linearly independent rows.
These codes are a class of linear codes where the parity check matrix has very few 1's.
This effectively translates to very few variables in each of the constraints on the variables in code-words.
For example consider matrix $H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$ and consider the every code-word to be of form x $= (\begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \end{bmatrix})$
The constraints translate to the following
$x_1 + x_2 + x_4 = 0$, $x_3 + x_4 + x_6 = 0$,$x_4 + x_5 + x_7 = 0$

Communication Model assumption

We assume a memoryless, no-feedback communication system where we send the vector x satisfying the constraint $xH^T = 0$ and receive the vector y. We denote the set of all codewords satisfying the above constraint as C. Our aim is the recover vector x with the least error probability. We denote the vector x as $(x_1, x_2, ..., x_n)$ and the vector y as $(y_1, y_2, ..., y_n)$.

Communication Model assumption

We assume a memoryless, no-feedback communication system where we send the vector x satisfying the constraint $xH^T = 0$ and receive the vector y. We denote the set of all codewords satisfying the above constraint as C. Our aim is the recover vector x with the least error probability. We denote the vector x as $(x_1, x_2, ..., x_n)$ and the vector y as $(y_1, y_2, ..., y_n)$.
Formally we would like to calculate (also called block-MAP decoding)

$$\hat{x}^{\text{MAP}}(y) = \underset{x}{\operatorname{argmax}}\, p_{X|Y}(x|y)$$

Communication Model assumption

We assume a memoryless, no-feedback communication system where we send the vector x satisfying the constraint $xH^T = 0$ and receive the vector y. We denote the set of all codewords satisfying the above constraint as C. Our aim is the recover vector x with the least error probability. We denote the vector x as $(x_1, x_2, ..., x_n)$ and the vector y as $(y_1, y_2, ..., y_n)$.
Formally we would like to calculate (also called block-MAP decoding)

$$\hat{x}^{\text{MAP}}(y) = \underset{x}{\text{argmax}}\, p_{X|Y}(x|y)$$

To motivate the computation above, we would first calculate

$$\hat{x}_i^{\text{MAP}}(y) = \underset{x_i \in \{1,-1\}}{\text{argmax}}\, p_{X_i|Y}(x_i|y)$$

Communication Model assumption

We assume a memoryless, no-feedback communication system where we send the vector x satisfying the constraint $xH^T = 0$ and receive the vector y. We denote the set of all codewords satisfying the above constraint as C. Our aim is the recover vector x with the least error probability. We denote the vector x as $(x_1, x_2, ..., x_n)$ and the vector y as $(y_1, y_2, ..., y_n)$. Formally we would like to calculate (also called block-MAP decoding)

$$\hat{x}^{\text{MAP}}(y) = \underset{x}{\operatorname{argmax}}\, p_{X|Y}(x|y)$$

To motivate the computation above, we would first calculate

$$\hat{x}_i^{\text{MAP}}(y) = \underset{x_i \in \{1,-1\}}{\operatorname{argmax}}\, p_{X_i|Y}(x_i|y)$$

This decoding above is called Bit Wise MAP decoding where we estimate each bit such that the probability of error for that bit is minimised. Note that bit wise MAP decoding for every bit may be different from block wise MAP decoding.

Factor Graphs

Factor Graphs essentially help us to factorise sum of a complex expression into a few factors.

Factor Graphs

Factor Graphs essentially help us to factorise sum of a complex expression into a few factors.

Consider $\sum_{i,j} a_i.b_j$. This could be written as $(\sum_i a_i).(\sum_j b_j)$.

Factor Graphs

Factor Graphs essentially help us to factorise sum of a complex expression into a few factors.

Consider $\sum\limits_{i,j} a_i.b_j$. This could be written as $(\sum\limits_i a_i).(\sum\limits_j b_j)$. This idea may be pretty useful in the context on LDPC codes described above since there is pretty less variables in each constraint when we wish to construct marginals.

Factor Graphs

Factor Graphs essentially help us to factorise sum of a complex expression into a few factors.
Consider $\sum_{i,j} a_i.b_j$. This could be written as $(\sum_i a_i).(\sum_j b_j)$. This idea may be pretty useful in the context on LDPC codes described above since there is pretty less variables in each constraint when we wish to construct marginals.
For the example if

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = f_1(x_1, x_2, x_3).f_2(x_1, x_4, x_6)f_3(x_4)f_4(x_4, x_5) \qquad (1)$$

we could calculate the marginal for x_1 with respect to f as follows:

$$\sum_{\sim x_1} f(x_1, x_2, x_3, x_4, x_5, x_6)$$
$$= \Big[\sum_{x_2,x_3} f_1(x_1, x_2, x_3)\Big]\Big[\sum_{x_4} f_3(x_4)\Big(\sum_{x_6} f_2(x_1, x_4, x_6)\Big).\Big(\sum_{x_5} f_4(x_4, x_5)\Big)\Big]$$

Graphical representation of Factor Graphs

We represent the factor graphs as Tanner Bi-partite Graphs. Each constraint in $x.H^T = 0$ is denoted as a check-node and each variable in vector x is denoted as a variable node in the graph.
An edge exists from a check-node to a variable node in the Tanner Graph iff this variable node is present in the constraint corresponding to the check node.

Graphical representation of Factor Graphs

We represent the factor graphs as Tanner Bi-partite Graphs. Each constraint in $x.H^T = 0$ is denoted as a check-node and each variable in vector x is denoted as a variable node in the graph.
An edge exists from a check-node to a variable node in the Tanner Graph iff this variable node is present in the constraint corresponding to the check node.
We construct factor graphs corresponding to this Tanner graph under the assumption that Tanner Graph is a tree(cycle-free) by rooting it under a variable node. We denote variable nodes by circles and check nodes by squares.

Graphical representation of Factor Graphs

We represent the factor graphs as Tanner Bi-partite Graphs. Each constraint in $x.H^T = 0$ is denoted as a check-node and each variable in vector x is denoted as a variable node in the graph.
An edge exists from a check-node to a variable node in the Tanner Graph iff this variable node is present in the constraint corresponding to the check node.
We construct factor graphs corresponding to this Tanner graph under the assumption that Tanner Graph is a tree(cycle-free) by rooting it under a variable node. We denote variable nodes by circles and check nodes by squares.

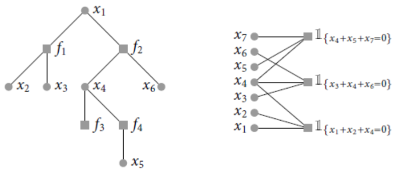

x_1
f_1
f_2
x_2
x_3
x_4
x_6
f_3
f_4
x_5
x_7
x_6
x_5
x_4
x_3
x_2
x_1
$\mathbb{1}_{\{x_4+x_5+x_7=0\}}$
$\mathbb{1}_{\{x_3+x_4+x_6=0\}}$
$\mathbb{1}_{\{x_1+x_2+x_4=0\}}$

Figure: Left: Factor graph as in 1 and Right : tanner graph for matrix H

Factor graph construction when no cycles

In these problems we assume that the factor graph is a bipartite tree i.e, there can be no two paths between any two nodes. We wish to compute $\sum_{\sim z} g(z, ...)$ where $g(z, ...)$ denotes all constraints corresponding to this factor graph.

Factor graph construction when no cycles

In these problems we assume that the factor graph is a bipartite tree i.e, there can be no two paths between any two nodes. We wish to compute $\sum_{\sim z} g(z, ...)$ where $g(z, ...)$ denotes all constraints corresponding to this factor graph.

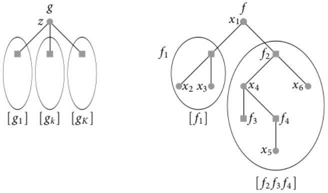


Figure: Generic representation of Factor Graph and Particular instance

Factor graph construction when no cycles

In these problems we assume that the factor graph is a bipartite tree i.e, there can be no two paths between any two nodes. We wish to compute $\sum_{\sim z} g(z, ...)$ where $g(z, ...)$ denotes all constraints corresponding to this factor graph.

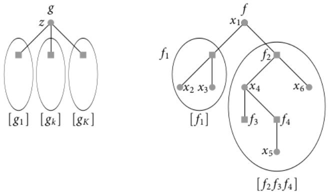


Figure: Generic representation of Factor Graph and Particular instance

Consider $g_k(z, ..)$ from figure above. Variable z cannot be shared since they form a tree.

$$\sum_{\sim z} g(z, ..) = \sum_{\sim z} \prod_{k=1}^{K} [g_k(z, ..)] = \prod_{k=1}^{K} \sum_{\sim z} g_k(z, ..)$$

Forney style factor graphs

- Forney style factor graph coverts the factor graph which is bipartite into a non bipartite graph by replacing variable nodes as (half-) edges.

Forney style factor graphs

- Forney style factor graph coverts the factor graph which is bipartite into a non bipartite graph by replacing variable nodes as (half-) edges.
- In case a variable node has degree greater than 2, we replicate by an equality factor. However note that the additional variables have to be equal to the parent by some equality factor.

Forney style factor graphs

- Forney style factor graph coverts the factor graph which is bipartite into a non bipartite graph by replacing variable nodes as (half-) edges.
- In case a variable node has degree greater than 2, we replicate by an equality factor. However note that the additional variables have to be equal to the parent by some equality factor.

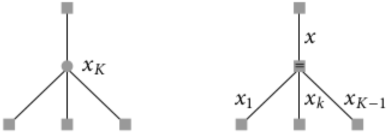


Figure: Conversion of variable node of degree K to FSFG from factor graph. Note that we need to ensure $x = x_1 = x_2 = .. = x_K$

Factor graphs continued

Factor graphs continued

Thus we may recursively calculate

$$\sum_{\sim z} g_k(z, ..) = \sum_{\sim z} h(z, z_1, ..z_J) \prod_{j=1}^{J} [h_j(z_j, ..)]$$

$$= \sum_{\sim z} h(z, z_1, ..z_J) \prod_{j=1}^{J} [\sum_{\sim z_j} h_j(z_j, ..)]$$

This idea may essentially be used for marginalisation by message passing.

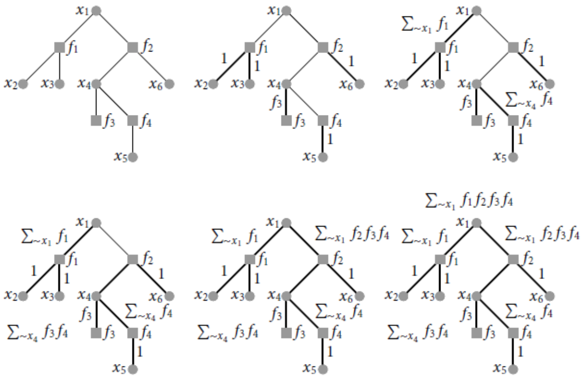


Figure: Sample message passing for marginalisation of f w.r.t x_1

This can be generalised for any other variable by rooting the tree at any other variable node.

Message processing rules

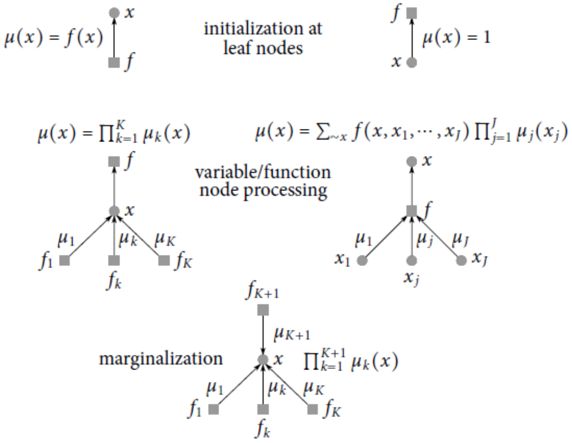


Figure: Formal message passing rules

Bit-Wise Decoding via Message passing

The bit wise MAP decoder reads that :

$$\begin{aligned}\hat{x}_i^{MAP}(y) &= \underset{x_i\in\{1,-1\}}{\operatorname{argmax}}\ p_{X_i|Y}(x_i|y) \\ &= \underset{x_i\in\{+1,-1\}}{\operatorname{argmax}} \sum_{\sim x_i} p_{Y|X}(y|x)p_X(x) \\ &= \underset{x_i\in\{+1,-1\}}{\operatorname{argmax}} \sum_{\sim x_i} (\prod_j p_{Y_j|X_j}(y_j|x_j))\mathbb{1}_{x\in C}\end{aligned}$$

Now $(\prod_j p_{Y_j|X_j}(y_j|x_j))\mathbb{1}_{x\in C}$ can again be represented as a factor graph $g(x_i, ...)$ after rooting it at a variable node x_i if Tanner graph corresponding to this code is a tree.

Bit-Wise Decoding via Message passing

The bit wise MAP decoder reads that :

$$\begin{aligned}\hat{x}_i^{MAP}(y) &= \underset{x_i\in\{1,-1\}}{\operatorname{argmax}}\ p_{X_i|Y}(x_i|y) \\ &= \underset{x_i\in\{+1,-1\}}{\operatorname{argmax}} \sum_{\sim x_i} p_{Y|X}(y|x)p_X(x) \\ &= \underset{x_i\in\{+1,-1\}}{\operatorname{argmax}} \sum_{\sim x_i} (\prod_j p_{Y_j|X_j}(y_j|x_j))\mathbb{1}_{x\in C}\end{aligned}$$

Now $(\prod_j p_{Y_j|X_j}(y_j|x_j))\mathbb{1}_{x\in C}$ can again be represented as a factor graph $g(x_i, ...)$ after rooting it at a variable node x_i if Tanner graph corresponding to this code is a tree.

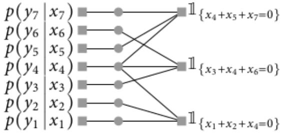

$p(y_7|x_7)$
$p(y_6|x_6)$
$p(y_5|x_5)$
$p(y_4|x_4)$
$p(y_3|x_3)$
$p(y_2|x_2)$
$p(y_1|x_1)$
$\mathbb{1}_{\{x_4+x_5+x_7=0\}}$
$\mathbb{1}_{\{x_3+x_4+x_6=0\}}$
$\mathbb{1}_{\{x_1+x_2+x_4=0\}}$

Figure: Factor graph for our example under bit-decoding

Blockwise MAP decoding

We can see that

$$\begin{aligned}\hat{x}^{MAP}(y) &= \operatorname*{argmax}_x p_{X|Y}(x|y) \\ &= \operatorname*{argmax}_x (\prod_j p_{Y_j|X_j}(y_j|x_j))\mathbb{1}_{\{x\in C\}}\end{aligned}$$

Now consider i^{th} bit of $\hat{x}^{MAP}(y)$ which might be written as

$$\begin{aligned}(\hat{x}^{MAP}(y))_i &= \operatorname*{argmax}_{x_i\in\{+1,-1\}} \max_{\sim x_i} \prod_j (p_{Y_j|X_j}(y_j|x_j))(\mathbb{1}_{\{x\in C\}}) \\ &= \operatorname*{argmax}_{x_i\in\{+1,-1\}} \max_{\sim x_i} \sum_j \log(p_{Y_j|X_j}(y_j|x_j)) + \log(\mathbb{1}_{\{x\in C\}})\end{aligned}$$

Blockwise MAP decoding

We can see that

$$\begin{aligned}\hat{x}^{MAP}(y) &= \operatorname*{argmax}_x p_{X|Y}(x|y) \\ &= \operatorname*{argmax}_x \Big(\prod_j p_{Y_j|X_j}(y_j|x_j)\Big)\mathbb{1}_{\{x \in C\}}\end{aligned}$$

Now consider i^{th} bit of $\hat{x}^{MAP}(y)$ which might be written as

$$\begin{aligned}(\hat{x}^{MAP}(y))_i &= \operatorname*{argmax}_{x_i \in \{+1,-1\}} \max_{\sim x_i} \prod_j (p_{Y_j|X_j}(y_j|x_j))(\mathbb{1}_{\{x \in C\}}) \\ &= \operatorname*{argmax}_{x_i \in \{+1,-1\}} \max_{\sim x_i} \sum_j \log(p_{Y_j|X_j}(y_j|x_j)) + \log(\mathbb{1}_{\{x \in C\}})\end{aligned}$$

We can show that the same distributive properties of (product,sum) hold for (sum,max) as well. Specifically, $\max\{x+y, y+z\} = x + \max\{y, z\}$.

Blockwise MAP decoding

We can see that

$$\begin{aligned}\hat{x}^{MAP}(y) &= \underset{x}{\operatorname{argmax}}\, p_{X|Y}(x|y) \\ &= \underset{x}{\operatorname{argmax}}\Big(\prod_j p_{Y_j|X_j}(y_j|x_j)\Big)\mathbb{1}_{\{x\in C\}}\end{aligned}$$

Now consider i^{th} bit of $\hat{x}^{MAP}(y)$ which might be written as

$$\begin{aligned}(\hat{x}^{MAP}(y))_i &= \underset{x_i\in\{+1,-1\}}{\operatorname{argmax}}\ \max_{\sim x_i} \prod_j (p_{Y_j|X_j}(y_j|x_j))(\mathbb{1}_{\{x\in C\}}) \\ &= \underset{x_i\in\{+1,-1\}}{\operatorname{argmax}}\ \max_{\sim x_i} \sum_j \log(p_{Y_j|X_j}(y_j|x_j)) + \log(\mathbb{1}_{\{x\in C\}})\end{aligned}$$

We can show that the same distributive properties of (product,sum) hold for (sum,max) as well. Specifically, $\max\{x+y, y+z\} = x + \max\{y,z\}$. The metric used here would be $\log(p_{Y_j|X_j}(y_j|x_j))$ instead of $(p_{Y_j|X_j}(y_j|x_j))$ in bit-wise decoding example. from each leaf check node.

Limitations of cycle free codes

Theorem : Let C be a binary linear code of rate r that admits a binary tanner graph that is a forest. Then C contains at least $\frac{2r-1}{2}n$ codewords of weight 2.

Limitations of cycle free codes

Theorem : Let C be a binary linear code of rate r that admits a binary tanner graph that is a forest. Then C contains at least $\frac{2r-1}{2}n$ codewords of weight 2.
This is effectively not a good news for us.

Limitations of cycle free codes

Theorem : Let C be a binary linear code of rate r that admits a binary tanner graph that is a forest. Then C contains at least $\frac{2r-1}{2}n$ codewords of weight 2.
This is effectively not a good news for us.
So we cannot be contended with cycle free graphs as they may have many pairs of code-words with small Hamming distance. Thus we have to look at codes whose Tanner graphs may have small cycles.

Limitations of cycle free codes

Theorem : Let C be a binary linear code of rate r that admits a binary tanner graph that is a forest. Then C contains at least $\frac{2r-1}{2}n$ codewords of weight 2.
This is effectively not a good news for us.
So we cannot be contended with cycle free graphs as they may have many pairs of code-words with small Hamming distance. Thus we have to look at codes whose Tanner graphs may have small cycles.
In the following slides, we attempt to do decoding under a preliminary binary erasure channel and look at the asymptotic behaviour as block length tends to ∞.

Binary erasure channel

This is an erasure channel where there is not bit flip.
The bit may get erased with probability ϵ denoted by BEC(ϵ).

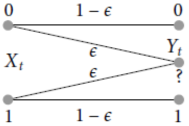


Figure: Binary symmetric channel

Binary erasure channel

This is an erasure channel where there is not bit flip.
The bit may get erased with probability ϵ denoted by BEC(ϵ).

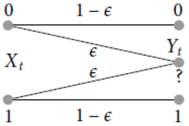


Figure: Binary symmetric channel

We can check Shannon capacity of this channel to be $C_{BEC}(\epsilon) = (1 - \epsilon)$

Binary erasure channel

This is an erasure channel where there is not bit flip.
The bit may get erased with probability ϵ denoted by BEC(ϵ).

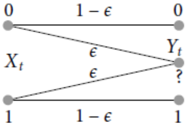
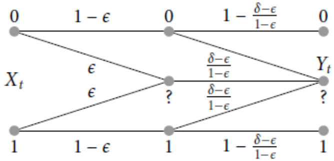


Figure: Binary symmetric channel

We can check Shannon capacity of this channel to be $C_{BEC}(\epsilon) = (1 - \epsilon)$
Also we may degrade BEC(ϵ) to BEC(δ) by coupling it with BEC($\frac{\delta-\epsilon}{\delta+\epsilon}$)


0
$1-\epsilon$
0
$1-\frac{\delta-\epsilon}{1-\epsilon}$
0
X_t
ϵ
ϵ
?
$\frac{\delta-\epsilon}{1-\epsilon}$
$\frac{\delta-\epsilon}{1-\epsilon}$
Y_t
?
1
$1-\epsilon$
1
$1-\frac{\delta-\epsilon}{1-\epsilon}$
1

Tanner graph and related symbols

We tend to denote an LDPC code by bipartite graph with 2 rows (check-nodes and variable nodes)

Tanner graph and related symbols

We tend to denote an LDPC code by bipartite graph with 2 rows (check-nodes and variable nodes)
i^{th} node in first row is connected to j^{th} node in second row if H_{ji}=1. Thus different parity check matrices may give different Tanner Graphs

Tanner graph and related symbols

We tend to denote an LDPC code by bipartite graph with 2 rows (check-nodes and variable nodes)

i^{th} node in first row is connected to j^{th} node in second row if $H_{ji}=1$. Thus different parity check matrices may give different Tanner Graphs

We often use the following symbols. Λ_i denotes the number of variable nodes of degree i and P_i denote the number of parity check nodes of degree i.

We use the following polynomials

$\Lambda(x) = \sum_{i=1}^{l_{max}} \Lambda_i x^i$ and $P(x) = \sum_{i=1}^{r_{max}} P_i . x^i$

Tanner graph and related symbols

We tend to denote an LDPC code by bipartite graph with 2 rows (check-nodes and variable nodes)
i^{th} node in first row is connected to j^{th} node in second row if H_{ji}=1. Thus different parity check matrices may give different Tanner Graphs
We often use the following symbols. Λ_i denotes the number of variable nodes of degree i and P_i denote the number of parity check nodes of degree i.
We use the following polynomials
$\Lambda(x) = \sum_{i=1}^{l_{max}} \Lambda_i x^i$ and $P(x) = \sum_{i=1}^{r_{max}} P_i.x^i$

Simple Relationships

$\Lambda(1) = n$, $P(1) = n(1 - r)$, $\Lambda'(1) = P'(1)$ $\sum_i i\Lambda_i = \sum_i iP_i$ where r is the design rate of code

Tanner graph and related symbols

We tend to denote an LDPC code by bipartite graph with 2 rows (check-nodes and variable nodes)

i^{th} node in first row is connected to j^{th} node in second row if H_{ji}=1. Thus different parity check matrices may give different Tanner Graphs

We often use the following symbols. Λ_i denotes the number of variable nodes of degree i and P_i denote the number of parity check nodes of degree i.

We use the following polynomials

$$\Lambda(x) = \sum_{i=1}^{l_{max}} \Lambda_i x^i \text{ and } P(x) = \sum_{i=1}^{r_{max}} P_i.x^i$$

Simple Relationships

$\Lambda(1) = n$, $P(1) = n(1 - \imath)$, $\Lambda'(1) = P'(1)$ $\sum_i i\Lambda_i = \sum_i iP_i$ where $\imath$ is the design rate of code

Normalised polynomials written as $L(x) = \frac{\Lambda(x)}{\Lambda(1)}$ $R(x) = \frac{P(x)}{P(1)}$

Symbols from an edge perspective

We define new symbols from an edge perspective.

$$\lambda(x) = \sum_i \lambda_i x^{i-1} = \frac{\Lambda'(x)}{\Lambda'(1)}$$

$$\rho(x) = \sum_i \rho_i x^{i-1} = \frac{P'(x)}{P'(1)}$$

Symbols from an edge perspective

We define new symbols from an edge perspective.

$$\lambda(x) = \sum_i \lambda_i x^{i-1} = \frac{\Lambda'(x)}{\Lambda'(1)}$$

$$\rho(x) = \sum_i \rho_i x^{i-1} = \frac{P'(x)}{P'(1)}$$

Here $\lambda_i(\rho_i)$ denote the fraction of edges that connect to variable(check) nodes of degree i respectively.

Symbols from an edge perspective

We define new symbols from an edge perspective.

$\lambda(x) = \sum_i \lambda_i x^{i-1} = \frac{\Lambda'(x)}{\Lambda'(1)}$

$\rho(x) = \sum_i \rho_i x^{i-1} = \frac{P'(x)}{P'(1)}$

Here $\lambda_i(\rho_i)$ denote the fraction of edges that connect to variable(check) nodes of degree i respectively.

We also write $l_{avg} = L'(1) = \frac{1}{\int_0^1 \lambda(x)dx}$ and $r_{avg} = R'(1) = \frac{1}{\int_0^1 \rho(x)dx}$

We can show design rate of code

$r(\Lambda, P) = 1 - \frac{L'(1)}{R'(1)} = 1 - \frac{l_{avg}}{r_{avg}}$

In terms of λ, ρ we may write it as

$r(\lambda, \rho) = 1 - \frac{\int_0^1 \rho(x)}{\int_0^1 \lambda(x)}$

Symbols from an edge perspective

We define new symbols from an edge perspective.

$\lambda(x) = \sum_i \lambda_i x^{i-1} = \frac{\Lambda'(x)}{\Lambda'(1)}$

$\rho(x) = \sum_i \rho_i x^{i-1} = \frac{P'(x)}{P'(1)}$

Here $\lambda_i(\rho_i)$ denote the fraction of edges that connect to variable(check) nodes of degree i respectively.

We also write $l_{avg} = L'(1) = \frac{1}{\int_0^1 \lambda(x)dx}$ and $r_{avg} = R'(1) = \frac{1}{\int_0^1 \rho(x)dx}$

We can show design rate of code

$r(\Lambda, P) = 1 - \frac{L'(1)}{R'(1)} = 1 - \frac{l_{avg}}{r_{avg}}$

In terms of λ, ρ we may write it as

$r(\lambda, \rho) = 1 - \frac{\int_0^1 \rho(x)}{\int_0^1 \lambda(x)}$

However it is important to note that the actual rate of the code may be be higher since some of the parity checks be redundant.

The standard ensemble LDPC(Λ, P)

Each graph in LDPC(Λ, P) has $\Lambda(1)$ variable nodes and $P(1)$ check nodes: Λ_i variable nodes of degree i and P_i check nodes of degree i.
A node of degree i is a node from where i edges emanate so in total there are $\Lambda'(1) = P'(1)$ sockets on each side. Label the sockets on each side with the set $[\Lambda'(1)] = \{1, ..., \Lambda'(1)\}$ in some arbitrary but fixed way.
Let σ be a permutation on $[\Lambda'(1)]$. Associate a bipartite graph by connecting i^{th} socket on variable side to the $\sigma(i)^{th}$ socket on check node side. Letting σ run over all permutations on $[\Lambda'(1)]$ generates a set of bipartite graphs. Finally, we place uniform probability distributions on the set of permutations to generate the ensemble LDPC(Λ, P).

Construction of parity check matrix H

Since there could be multiple edges between two node,

- $H_{ji} = 1$ iff there are an odd number of edges between i^{th} variable node and j^{th} check node.
- $H_{ji} = 0$ iff there are an even number of edges between i^{th} variable node and j^{th} check node.

Message passing decoder for binary erasure channel

By standard message passing rules, initial message is $=$ $(p_{Y_j|X_j}(y_j|0), p_{Y_j|X_j}(y_j|1))$. These messages could be (1-ϵ,0),(ϵ,ϵ) or (0,1-ϵ) corresponding to 0,erasure and 1.
Equivalently we work with a set of messages (1,0),(1,1) and (0,1)

Message passing decoder for binary erasure channel

By standard message passing rules, initial message is $=$ $(p_{Y_j|X_j}(y_j|0), p_{Y_j|X_j}(y_j|1))$. These messages could be (1-ϵ,0),(ϵ,ϵ) or (0,1-ϵ) corresponding to 0,erasure and 1.
Equivalently we work with a set of messages (1,0),(1,1) and (0,1)

Data processing rules

These rules are directly linked to the message propagation described earlier

- At each variable node the outgoing node is an erasure if all of the one incoming node is erasure.
- Similarly at each check node the outgoing is a erasure if any one of the incoming node is so.

For every variable nodes we could just directly take their products.

Message passing decoder for binary erasure channel

By standard message passing rules, initial message is $=$ $(p_{Y_j|X_j}(y_j|0), p_{Y_j|X_j}(y_j|1))$. These messages could be (1-ϵ,0),(ϵ,ϵ) or (0,1-ϵ) corresponding to 0,erasure and 1.
Equivalently we work with a set of messages (1,0),(1,1) and (0,1)

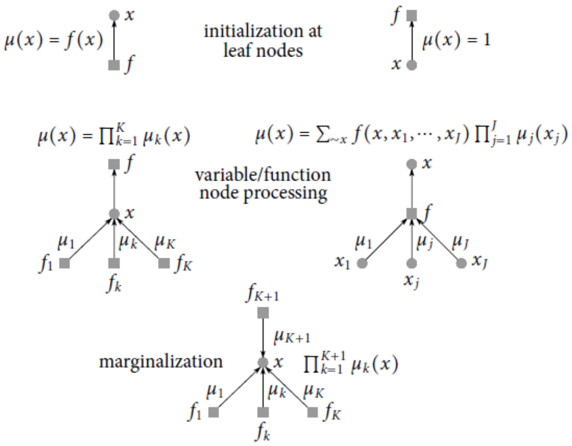
Data processing rules

These rules are directly linked to the message propagation described earlier

- At each variable node the outgoing node is an erasure if all of the one incoming node is erasure.
- Similarly at each check node the outgoing is a erasure if any one of the incoming node is so.

Formally for every check node we write $(\mu(0),\mu(1))=(\sum_{\{x_1,x_2\}} \mathbb{1}_{x_1+x_2=0}\mu_1(x_1)\mu_2(x_2), \sum_{\{x_1,x_2\}} \mathbb{1}_{x_1+x_2=1}\mu_1(x_1)\mu_2(x_2))$ as per the message processing rules
For every variable nodes we could just directly take their products.

Diagrams for message passing decoder under BEC



$\mu(x) = f(x)$
x
f
initialization at leaf nodes
f
$\mu(x) = 1$
x
$\mu(x) = \prod_{k=1}^{K} \mu_k(x)$
f
x
μ_1
μ_k
μ_K
f_1
f_k
f_K
variable/function node processing
$\mu(x) = \sum_{\sim x} f(x, x_1, \cdots, x_J) \prod_{j=1}^{J} \mu_j(x_j)$
x
f
μ_1
μ_j
μ_J
x_1
x_j
x_J
f_{K+1}
μ_{K+1}
marginalization
x
$\prod_{k=1}^{K+1} \mu_k(x)$
μ_1
μ_k
μ_K
f_1
f_k
f_K

Figure: Message passing rules under BEC

An example of message passing decoder

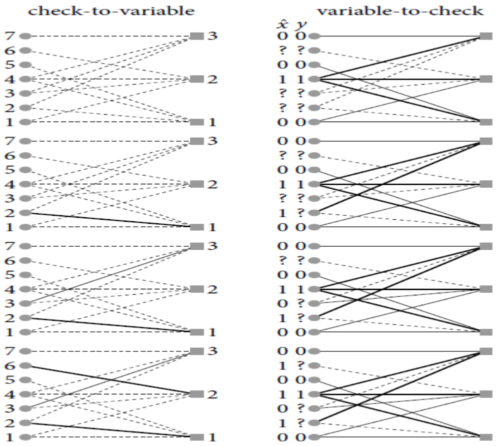


Figure: Message passing decoder for [7,4] Hamming code. A 0 message indicated by a thin line, 1 message by thick line , ? message by a dotted line. Received word $y = (0, ?, ?, 1, 0, ?, 0)$

Some simplifications

We denote $P_{\text{BP}}(G, \epsilon, l, x)$ as the conditional(bit or block) probability of erasure after l^{th} decoding iteration assuming x was sent, $x \in C$.
Note that each decoding iteration consists of messages being sent once from variable nodes to check nodes and once again being sent from check to variable nodes.

Some simplifications

We denote $P_{\mathrm{BP}}(G, \epsilon, l, x)$ as the conditional(bit or block) probability of erasure after l^{th} decoding iteration assuming x was sent, $x \in C$.
Note that each decoding iteration consists of messages being sent once from variable nodes to check nodes and once again being sent from check to variable nodes.

We can say that $P^{BP}(G, \epsilon, l, x) = \frac{\sum_{x' \in C} P^{BP}(G,\epsilon,l,x')}{|C|} = P^{BP}(G, \epsilon, l)$

Some simplifications

We denote $P_{\mathrm{BP}}(G, \epsilon, l, x)$ as the conditional(bit or block) probability of erasure after l^{th} decoding iteration assuming x was sent, $x \in C$.
Note that each decoding iteration consists of messages being sent once from variable nodes to check nodes and once again being sent from check to variable nodes.

We can say that $P^{BP}(G, \epsilon, l, x) = \frac{\sum_{x' \in C} P^{BP}(G,\epsilon,l,x')}{|C|} = P^{BP}(G, \epsilon, l)$
Here we can clearly observe the probability of error or success of message passing decoding won't change whether we send 0 or 1 as message bit So we can work with all zero code-word as message sent.

Concentration around ensemble averages

Theorem: Let G, uniformly chosen at random from LDPC(n,λ,ρ) be used for transmission over BEC(ϵ).Assume that decoder performs l rounds of decoding and $P_b^{BP}(G,\epsilon,l)$ denote bit erasure probability.Then for every δ, there exists an $\alpha > 0$, $\alpha = \alpha(\lambda,\rho,\epsilon,\delta,l)$ such that

$$\mathbb{P}\{|P^{BP}(G,\epsilon,l) - \mathbb{E}_{G'\in LDPC(n,\lambda,\rho)}[P^{BP}(G',\epsilon,l)]| > \delta\} \leq e^{-\alpha n}$$

This essentially shows all except an exponential decaying average behave within an arbitrarily small δ from ensemble.

Computation graph

In this model we expand the tanner graph over decoding iterations and tend to form a tree over several decoding iterations.

Computation graph

In this model we expand the tanner graph over decoding iterations and tend to form a tree over several decoding iterations.
We could do so over a node perspective (computation graph for a message at node randomly chosen) or over an edge perspective(computation for message sent over an edge randomly chosen) denoted by $\mathcal{C}_l^{\circ}(n, \lambda, \rho)$ or $\mathcal{C}_l^{\rightarrow}(n, \lambda, \rho)$
However note that we could have multiple occurrences of same node or edge over the computation graph. Note that $P_b^{BP}(T, \epsilon)$ denotes the probability of error given that the computation graph is T.

Computation graph

In this model we expand the tanner graph over decoding iterations and tend to form a tree over several decoding iterations.
We could do so over a node perspective (computation graph for a message at node randomly chosen) or over an edge perspective(computation for message sent over an edge randomly chosen) denoted by $\mathcal{C}_l^{\circ}(n, \lambda, \rho)$ or $\mathcal{C}_l^{\rightarrow}(n, \lambda, \rho)$
However note that we could have multiple occurrences of same node or edge over the computation graph. Note that $P_b^{BP}(T, \epsilon)$ denotes the probability of error given that the computation graph is T.
With this definition we have

$$\mathbb{E}_{LDPC(n,\lambda,\rho)}[P_b^{BP}(G, \epsilon, l)] = \sum_T \mathbb{P}\{T \in \mathcal{C}_l^{\circ}(n, \lambda, \rho)\} P_b^{BP}(T, \epsilon)$$

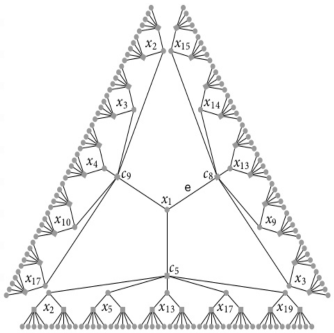
Computation graph

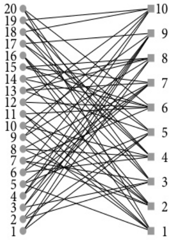
In this model we expand the tanner graph over decoding iterations and tend to form a tree over several decoding iterations.
We could do so over a node perspective (computation graph for a message at node randomly chosen) or over an edge perspective(computation for message sent over an edge randomly chosen) denoted by $\mathcal{C}_l^{\circ}(n,\lambda,\rho)$ or $\mathcal{C}_l^{\rightarrow}(n,\lambda,\rho)$
However note that we could have multiple occurrences of same node or edge over the computation graph. Note that $P_b^{BP}(T,\epsilon)$ denotes the probability of error given that the computation graph is T.
With this definition we have

$$\mathbb{E}_{LDPC(n,\lambda,\rho)}[P_b^{BP}(G,\epsilon,l)] = \sum_T \mathbb{P}\{T \in \mathcal{C}_l^{\circ}(n,\lambda,\rho)\} P_b^{BP}(T,\epsilon)$$

We now construct an ensemble of trees denoted by $\mathcal{T}_l^{\circ}(\lambda,\rho)$ and $\mathcal{T}_l^{\rightarrow}(\lambda,\rho)$ such that $\lim_{n\to\infty} \mathbb{E}_{LDPC(n,\lambda,\rho)}[P_b^{BP}(G,\epsilon,l)] = P^{BP}_{\mathcal{T}_l^{\circ}(\lambda,\rho)}(\epsilon)$ where $P^{BP}_{\mathcal{T}_l^{\circ}(\lambda,\rho)} = \sum_T \mathbb{P}\{T \in \mathcal{T}_l^{\circ}(\lambda,\rho)\} P_b^{BP}(T,\epsilon)$ whose construction is described in the following slides.

Some examples of computation graphs





$$\begin{array}{c|cccccccccccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
3 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
5 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
7 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
8 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
9 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
10 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0
\end{array} = H.$$

Figure: Computation graph of height 2(two iterations) for bit x_1 corresponding to H matrix and Tanner Graph on right. The computation graph of height 2 for edge e is the subtree consisting of edge e, variable node x_1 and two sub-trees rooted in check nodes c_5 and c_9.

Some intuition towards tree ensembles

We attempt to construct a tree ensemble such that asymptotically as n tends to ∞, the computation graph ensemble converges to this tree ensemble.

The essential idea behind this transformation is that the probability that the computation graph constructed has at least one node or edge appearing twice behave as $O(\frac{1}{n})$. Thus asymptotically, it behaves as a tree.

The following computation for $\mathcal{C}_1^\circ(n, \lambda(x) = x, \rho(x) = x^2)$ would demonstrate it.

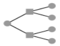
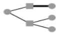
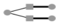
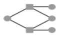
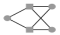
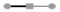
T	$P\{T \in \mathring{\mathcal{C}}_1(n, x, x^2)\}$	$P_b^{BP}(T, \epsilon)$
	$\frac{(2n-6)(2n-8)}{(2n-1)(2n-5)}$	$\epsilon(1-(1-\epsilon)^2)^2$
	$\frac{2(2n-6)}{(2n-1)(2n-5)}$	$\epsilon^2(1-(1-\epsilon)^2)$
	$\frac{1}{(2n-1)(2n-5)}$	ϵ^3
	$\frac{4(2n-6)}{(2n-1)(2n-5)}$	$\epsilon^2 + \epsilon^3(1-\epsilon)$
	$\frac{2}{(2n-1)(2n-5)}$	$\epsilon(1-(1-\epsilon)^2)$
	$\frac{2}{2n-1}$	ϵ^2

Figure: Elements of $\mathcal{C}_1^\circ(n, \lambda(x) = x, \rho(x) = x^2)$ with their probabilities Thick lines denote double edges

Tree ensemble

Let us define tree ensemble from edge and node perspective as $\mathcal{T}_l^{\circ}(\lambda, \rho)$ and $\mathcal{T}_l^{\rightarrow}(\lambda, \rho)$

Tree ensemble

Let us define tree ensemble from edge and node perspective as $\mathcal{T}_l^{\circ}(\lambda, \rho)$ and $\mathcal{T}_l^{\rightarrow}(\lambda, \rho)$

- The ensemble $\mathcal{T}_0^{\rightarrow}(\lambda, \rho)$ contains a single element- trivial tree consisting of root variable node.
- Define L(i) to be a bipartite tree rooted in a variable node with i(check-node) children and R(i) to be a bipartite tree rooted in a check-node with i variable nodes.
- To sample from $\mathcal{T}_l^{\rightarrow}(\lambda, \rho)$, l $\geq$ 1 first sample from $\mathcal{T}_{l-1}^{\rightarrow}(\lambda, \rho)$ and replace the following-
    - Each of its leaf variable nodes from a random element in $\{L(i)\}_{i\geq 1}$ where L(i) is chosen with probability λ_{i+1}
    - Each of its leaf check nodes from a random element in $\{R(i)\}_{i\geq 1}$ where R(i) is chosen with probability ρ_{i+1}

Tree ensemble

Let us define tree ensemble from edge and node perspective as $\mathcal{T}_l^{\circ}(\lambda, \rho)$ and $\mathcal{T}_l^{\rightarrow}(\lambda, \rho)$

- The ensemble $\mathcal{T}_0^{\rightarrow}(\lambda, \rho)$ contains a single element- trivial tree consisting of root variable node.
- Define L(i) to be a bipartite tree rooted in a variable node with i(check-node) children and R(i) to be a bipartite tree rooted in a check-node with i variable nodes.
- To sample from $\mathcal{T}_l^{\rightarrow}(\lambda, \rho)$, l $\geq$ 1 first sample from $\mathcal{T}_{l-1}^{\rightarrow}(\lambda, \rho)$ and replace the following-
    - Each of its leaf variable nodes from a random element in $\{L(i)\}_{i\geq 1}$ where L(i) is chosen with probability λ_{i+1}
    - Each of its leaf check nodes from a random element in $\{R(i)\}_{i\geq 1}$ where R(i) is chosen with probability ρ_{i+1}

This also implies the following recursive definition
To sample from $\mathcal{T}_l^{\rightarrow}(\lambda, \rho)$, sample from $\mathcal{T}_i^{\rightarrow}(\lambda, \rho)$, 0 $\leq$ i $\leq$ l and replace each of its leaf variable nodes by independent samples from $\mathcal{T}_{l-i}^{\rightarrow}(\lambda, \rho)$

Tree ensemble continued

- The ensemble $\mathcal{T}_0^\circ(\lambda, \rho)$ contains a single element- trivial tree consisting of root variable node.
- To sample from from $\mathcal{T}_1^\circ(\lambda, \rho)$, first choose an element randomly from $\{L(i)\}_{i\geq 1}$ where L(i) is chosen with probability L_i and then substitute each of its leaf check nodes from a random element in $\{R(i)\}_{i\geq 1}$ where R(i) is chosen with probability ρ_{i+1}
- The other things are same as that of previous definition

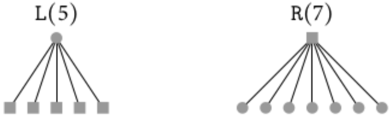


Figure: Basic trees $L(5)$ and $R(7)$

Examples

We give example of a ensemble where $\lambda(x) = (1/2)x + (1/2)\ x^2$, $\rho(x) = (1/5)x^5 + (4/5)\ x^4$

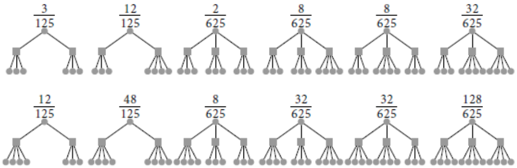


Figure: Ensembles corresponding to $\mathcal{T}_1^{\circ}(\lambda, \rho)$ with probabilities

Error probability for tree ensembles

As in computational graphs we can associate with each element of $\mathcal{T}_l^{\circ}(\lambda, \rho)$ a probability of error: we imagine each variable is initially labelled with 0 which could be erased independently with probability ϵ and the BP decoder tries to decode root node.

Error probability for tree ensembles

As in computational graphs we can associate with each element of $\mathcal{T}_l^\circ(\lambda, \rho)$ a probability of error: we imagine each variable is initially labelled with 0 which could be erased independently with probability ϵ and the BP decoder tries to decode root node.
We could write probability of error for tree ensemble as

$$P^{BP}_{\mathcal{T}_l^\circ(\lambda,\rho)} = \sum_{T} \mathbb{P}\{T \in \mathcal{T}_l^\circ(\lambda, \rho)\} P_b^{BP}(T, \epsilon)$$

Convergence to tree channel

Let us describe the tree channel and show how it links to LDPC decoding for large block lengths

- Consider the BEC characterised by erasure probability ϵ and tree ensemble $\mathcal{T}_l = \mathcal{T}_l(\lambda, \rho)$.
- The channel first picks a tree from the tree ensemble. The channel then takes binary inputs X $\in \{0{,}1\}$ with uniform probability and given X picks a code-word from $C_X(T)$ uniformly at random
- Transmits the code-word over the channel and the receiver sees (T,Y) and estimates X and $P^{BP}_{\mathcal{T}_l(\lambda,\rho)}(\epsilon)$ denotes the resultant bit error probability.

Convergence to tree channel

Let us describe the tree channel and show how it links to LDPC decoding for large block lengths

- Consider the BEC characterised by erasure probability ϵ and tree ensemble $\mathcal{T}_l = \mathcal{T}_l(\lambda, \rho)$.
- The channel first picks a tree from the tree ensemble. The channel then takes binary inputs X $\in \{0,1\}$ with uniform probability and given X picks a code-word from $C_X(T)$ uniformly at random
- Transmits the code-word over the channel and the receiver sees (T,Y) and estimates X and $P^{BP}_{\mathcal{T}_l(\lambda,\rho)}(\epsilon)$ denotes the resultant bit error probability.

For a given degree distribution pair (λ,ρ) for increasing block-lengths under l rounds of BP-decoding.Then

$$\lim_{n\to\infty} \mathbb{E}_{LDPC(n,\lambda,\rho)}[P_b^{BP}(G,\epsilon,l)] = P^{BP}_{\mathcal{T}_l^{\circ}(\lambda,\rho)}(\epsilon)$$

Convergence to tree channel

Let us describe the tree channel and show how it links to LDPC decoding for large block lengths

- Consider the BEC characterised by erasure probability ϵ and tree ensemble $\mathcal{T}_l = \mathcal{T}_l(\lambda, \rho)$.
- The channel first picks a tree from the tree ensemble. The channel then takes binary inputs X $\in \{0,1\}$ with uniform probability and given X picks a code-word from $C_X(T)$ uniformly at random
- Transmits the code-word over the channel and the receiver sees (T,Y) and estimates X and $P^{BP}_{\mathcal{T}_l(\lambda,\rho)}(\epsilon)$ denotes the resultant bit error probability.

For a given degree distribution pair (λ,ρ) for increasing block-lengths under l rounds of BP-decoding.Then

$$\lim_{n\to\infty} \mathbb{E}_{LDPC(n,\lambda,\rho)}[P_b^{BP}(G,\epsilon,l)] = P^{BP}_{\mathcal{T}_l^{\circ}(\lambda,\rho)}(\epsilon)$$

This effectively relates the decoding of LDPC code under increasing block-length to resemble the tree channel.

Density evolution

Consider a degree distribution pair (λ,ρ) with associated normalised degree distribution from node perspective L(x) Let ϵ be channel parameter Define x_{-1} =1 and for l $\geq$ 0 let $x_l = \epsilon\lambda(1-\rho(1-x_{l-1})$.
Then we say $P^{BP}_{\mathcal{T}^{\circ}_l(\lambda,\rho)}(\epsilon) = \epsilon L(1-\rho(1-x_{l-1})$, $P^{BP}_{\vec{\mathcal{T}}_l(\lambda,\rho)}(\epsilon)$= x_l

Density evolution

Consider a degree distribution pair (λ,ρ) with associated normalised degree distribution from node perspective L(x) Let ϵ be channel parameter Define x_{-1} =1 and for l $\geq$ 0 let $x_l = \epsilon\lambda(1-\rho(1-x_{l-1})$.
Then we say $P^{BP}_{\mathcal{T}_l^{\circ}(\lambda,\rho)}(\epsilon) = \epsilon L(1-\rho(1-x_{l-1})$, $P^{BP}_{\mathcal{T}_l^{\rightarrow}(\lambda,\rho)}(\epsilon) = x_l$
Some properties:

- Define f $(\epsilon,$x) $= \epsilon\lambda(1-\rho(1-x))$. Then f($\epsilon$,x) is increasing in both arguments for x,$\epsilon \in$ [0,1]
- If (λ,ρ) be a degree distribution pair and $\epsilon \in$ [0,1]. If $P^{BP}_{\mathcal{T}_l(\lambda,\rho)}(\epsilon)$ $\xrightarrow{l\rightarrow\infty}$ 0 then $P^{BP}_{\mathcal{T}_l(\lambda,\rho)}(\epsilon')$ $\xrightarrow{l\rightarrow\infty}$ 0 for $\forall$ 0 $\leq \epsilon' \leq \epsilon$

Density evolution

Consider a degree distribution pair (λ,ρ) with associated normalised degree distribution from node perspective L(x) Let ϵ be channel parameter Define x_{-1} =1 and for l $\geq$ 0 let $x_l = \epsilon\lambda(1-\rho(1-x_{l-1})$.
Then we say $P^{BP}_{T_l^\circ(\lambda,\rho)}(\epsilon) = \epsilon L(1-\rho(1-x_{l-1})$, $P^{BP}_{\vec{T_l}(\lambda,\rho)}(\epsilon)= x_l$
Some properties:

- Define f (ϵ,x) = $\epsilon\lambda(1-\rho(1-x))$. Then f($\epsilon$,x) is increasing in both arguments for x,$\epsilon \in$ [0,1]
- If (λ,ρ) be a degree distribution pair and $\epsilon \in$ [0,1]. If $P^{BP}_{T_l(\lambda,\rho)}(\epsilon)$ $\xrightarrow{l\to\infty}$ 0 then $P^{BP}_{T_l(\lambda,\rho)}(\epsilon')$ $\xrightarrow{l\to\infty}$ 0 for $\forall$ 0 $\leq \epsilon' \leq \epsilon$

Monotonicity w.r.t to iteration : Let $\epsilon, x_0 \in$ [0,1]. For l = 1,2,... define $x_l(x_0)$ = f(ϵ,$x_{l-1}(x_0)$).Then $x_l(x_0)$ is a monotone sequence converging to the nearest solution of equation x = f(ϵ,x)

Threshold

Here we are interested in ϵ where the probability of error no longer remains 0 even under infinitely large number of decoding iterations

More formally we define as

$\epsilon^{BP}(\lambda,\rho)= \sup\{\epsilon \in [0,1] : P^{BP}_{\mathcal{T}_l(\lambda,\rho)}(\epsilon') \xrightarrow{l\to\infty} 0\}$

Threshold

Here we are interested in ϵ where the probability of error no longer remains 0 even under infinitely large number of decoding iterations

More formally we define as

$\epsilon^{BP}(\lambda, \rho)= \sup\{\epsilon \in [0, 1] : P^{BP}_{\mathcal{T}_l(\lambda,\rho)}(\epsilon') \xrightarrow{l \to \infty} 0\}$

An equivalent definition: For a degree distribution (λ,ρ) and $\epsilon \in [0,1]$ let f $(\epsilon,x) = \epsilon\lambda(1 - \rho(1 - x))$

- $\epsilon^{BP}(\lambda, \rho) = \sup\{\epsilon \in [0, 1] : x = f(\epsilon, x)$ has no solution in (0,1) $\}$
- $\epsilon^{BP}(\lambda, \rho) = \inf\{\epsilon \in [0, 1] : x = f(\epsilon, x)$ has a solution in (0,1) $\}$

Threshold

Here we are interested in ϵ where the probability of error no longer remains 0 even under infinitely large number of decoding iterations

More formally we define as

$\epsilon^{BP}(\lambda, \rho)= \sup\{\epsilon \in [0,1] : P^{BP}_{\mathcal{T}_l(\lambda,\rho)}(\epsilon') \xrightarrow{l\to\infty} 0\}$

An equivalent definition: For a degree distribution (λ,ρ) and $\epsilon \in$ [0,1] let f (ϵ,x) = $\epsilon\lambda(1-\rho(1-x))$

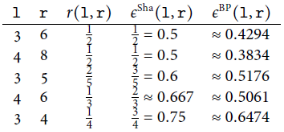
- $\epsilon^{BP}(\lambda, \rho) = \sup\{\epsilon \in [0,1] : x = f(\epsilon, x)$ has no solution in (0,1) $\}$
- $\epsilon^{BP}(\lambda, \rho) = \inf\{\epsilon \in [0,1] : x = f(\epsilon, x)$ has a solution in (0,1) $\}$

Critical point: Given a degree distribution pair (λ,ρ) we say it has a critical point x_{BP} if

f(ϵ^{BP},x^{BP}) = x^{BP} and $\left.\frac{\partial f(\epsilon^{BP},x^{BP})}{\partial x}\right|_{x=x^{BP}} = 1$

Some comparison

We compare the threshold on ϵ with Shannon threshold of the channel.



$\mathtt{l}$	$\mathtt{r}$	$r(\mathtt{l},\mathtt{r})$	$\epsilon^{\text{Sha}}(\mathtt{l},\mathtt{r})$	$\epsilon^{\text{BP}}(\mathtt{l},\mathtt{r})$
3	6	$\frac{1}{2}$	$\frac{1}{2} = 0.5$	≈ 0.4294
4	8	$\frac{1}{2}$	$\frac{1}{2} = 0.5$	≈ 0.3834
3	5	$\frac{2}{5}$	$\frac{3}{5} = 0.6$	≈ 0.5176
4	6	$\frac{1}{3}$	$\frac{2}{3} \approx 0.667$	≈ 0.5061
3	4	$\frac{1}{4}$	$\frac{3}{4} = 0.75$	≈ 0.6474

Figure: Comparison with Shannon threshold for regular distribution pairs

Some comparison

We compare the threshold on ϵ with Shannon threshold of the channel.

$\mathtt{l}$	$\mathtt{r}$	$r(\mathtt{l},\mathtt{r})$	$\epsilon^{\text{Sha}}(\mathtt{l},\mathtt{r})$	$\epsilon^{\text{BP}}(\mathtt{l},\mathtt{r})$
3	6	$\frac{1}{2}$	$\frac{1}{2} = 0.5$	≈ 0.4294
4	8	$\frac{1}{2}$	$\frac{1}{2} = 0.5$	≈ 0.3834
3	5	$\frac{2}{5}$	$\frac{3}{5} = 0.6$	≈ 0.5176
4	6	$\frac{1}{3}$	$\frac{2}{3} \approx 0.667$	≈ 0.5061
3	4	$\frac{1}{4}$	$\frac{3}{4} = 0.75$	≈ 0.6474

Figure: Comparison with Shannon threshold for regular distribution pairs

Here we just give example of (f(ϵ, x) - x) changes with ϵ around critical point plotted as a function of x

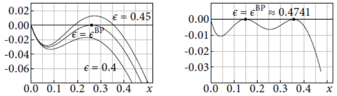


Figure: Graphical determination for two different degree distributions with $\epsilon^{\text{Shannon}} = \frac{1}{2}$ for both distributions

This figure attempts to demonstrate the fact that as $\epsilon^{\text{BP}}(\lambda, \rho)$ goes closer to $\epsilon^{\text{Shannon}}(\lambda, \rho)$, number of critical points tend to increase.

Introduction: Convolutional codes

Convolutional codes map infinite streams of data into infinite streams of data typically through a linear filter. Typically, we denote a linear filter $G(D) = \frac{p(D)}{q(D)}$ with memory $m = \max(\deg(p(D)), \deg(q(D)))$

Introduction: Convolutional codes

Convolutional codes map infinite streams of data into infinite streams of data typically through a linear filter. Typically, we denote a linear filter $G(D) = \frac{p(D)}{q(D)}$ with memory $m = \max(\deg(p(D)), \deg(q(D)))$
Without loss of generality, we assume $q_0 = 1$ where $q(D) = \sum_i q_i D^i$ and $p(D) = \sum_i p_i D^i$. Also we assume these codes to be over a binary finite field thus, the coefficients can be 0 or 1.

Introduction: Convolutional codes

Convolutional codes map infinite streams of data into infinite streams of data typically through a linear filter. Typically, we denote a linear filter $G(D) = \frac{p(D)}{q(D)}$ with memory $m = \max(\deg(p(D)), \deg(q(D)))$
Without loss of generality, we assume $q_0 = 1$ where $q(D) = \sum_i q_i D^i$ and $p(D) = \sum_i p_i D^i$. Also we assume these codes to be over a binary finite field thus, the coefficients can be 0 or 1.

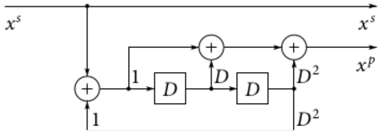


Figure: Binary symmetric recursive convolutional encoder with $m = 2$ and $G(D) = \frac{1+D+D^2}{1+D^2}$. The block D denotes a delay element.

Termination of convolutional codes

- We consider "terminated" schemes in this part, thus we consider the input codeword as $x^s = \{x_1^s, x_2^s, ..x_n^s, \underbrace{0, 0, ...0}_{\text{m times}}\}$. We keep the last m components to be zero. The output to this codeword is denoted as $x^p = (x_1^p, x_2^p, ..., x_{n+m}^p)$

Termination of convolutional codes

- We consider "terminated" schemes in this part, thus we consider the input codeword as $x^s = \{x_1^s, x_2^s, ..x_n^s, \underbrace{0, 0, ...0}_{\text{m times}}\}$. We keep the last m components to be zero. The output to this codeword is denoted as $x^p = (x_1^p, x_2^p, ..., x_{n+m}^p)$
- However, we consider a slight deviation in this terminated setting. For the first n steps the filter is $G(D)$ whereas for the next m steps the filter is $\tilde{G}(D) = p(D)$.

Termination of convolutional codes

- We consider "terminated" schemes in this part, thus we consider the input codeword as $x^s = \{x_1^s, x_2^s, ..x_n^s, \underbrace{0, 0, ...0}_{\text{m times}}\}$. We keep the last m components to be zero. The output to this codeword is denoted as $x^p = (x_1^p, x_2^p, ..., x_{n+m}^p)$
- However, we consider a slight deviation in this terminated setting. For the first n steps the filter is $G(D)$ whereas for the next m steps the filter is $\tilde{G}(D) = p(D)$.
- We consider the so-called state space model where state σ_{i-1} denotes the content of the shift register just before i^{th} bit x_i^s for $i = 1, 2, ..n + m$. Thus, we have the sequence $(x_i^s, \sigma_{i-1}) \rightarrow (x_i^p, \sigma_{i-1})$ where x_i^p denotes the i^{th} bit of the output sequence x^p.

State space representation contn..

The initial state is $\sigma_0 = (0, 0, .., 0)$ For $1 \ \leq i \leq m$, the evolution is described by $\sigma_i = \sigma_{i-1}A + x_i^s C$, $x_i^p = \sigma_{i-1}B^T + x_i^s p_0$ where matrices A,B and C are defined below.

State space representation contn..

The initial state is $\sigma_0 = (0, 0, .., 0)$ For $1 \leq i \leq m$, the evolution is described by $\sigma_i = \sigma_{i-1}A + x_i^s C$, $x_i^p = \sigma_{i-1}B^T + x_i^s p_0$ where matrices A,B and C are defined below.

$$A = \begin{bmatrix} q_1 & 1 & 0 & .... & 0 \\ q_2 & 0 & 1 & .... & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ q_{m-1} & 0 & 0 & 0 & 1 \\ q_m & 0 & 0 & 0 & 0 \end{bmatrix}_{m \times m} \quad B = \begin{bmatrix} p_1 + p_0.q_1 \\ p_2 + p_0.q_2 \\ \vdots \\ p_{m-1} + p_0.q_{m-1} \\ p_m + p_0 q_m \end{bmatrix}_{m \times 1} \quad C^T = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}_{m \times 1}$$

State space representation contn..

The initial state is $\sigma_0 = (0, 0, .., 0)$ For $1 \ \le i \le m$, the evolution is described by $\sigma_i = \sigma_{i-1}A + x_i^s C$, $x_i^p = \sigma_{i-1}B^T + x_i^s p_0$ where matrices A,B and C are defined below.

$$A = \begin{bmatrix} q_1 & 1 & 0 & .... & 0 \\ q_2 & 0 & 1 & .... & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ q_{m-1} & 0 & 0 & 0 & 1 \\ q_m & 0 & 0 & 0 & 0 \end{bmatrix}_{m \times m} \quad B = \begin{bmatrix} p_1 + p_0.q_1 \\ p_2 + p_0.q_2 \\ \vdots \\ p_{m-1} + p_0.q_{m-1} \\ p_m + p_0 q_m \end{bmatrix}_{m \times 1} \quad C^T = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}_{m \times 1}$$

We denote the encoding map as $x^p = \gamma(x^s)$. Thus, the code $C(G, n)$ is denoted as $C(G, n) = \{(x^s, \gamma(x^s)) : x^s = (x_1^s, x_2^s, ..., x_n^s, \underbrace{0, 0..0}_{\text{m times}})\}$,$x_i^s \in \mathbb{F}_2$

Bit-wise MAP decoding

We consider the code-word from the codebook $C(G, n)$ as (X^s, X^p) and (Y^s, Y^p) taking values (y^s, y^p) denotes the output of the channel (assumed memoryless, without feedback) considering (X^s, X^p) was transmitted. We thus denote the decoding function as $\hat{x}_i^s = \hat{x}_i^s(y^s, y^p)$

Bit-wise MAP decoding

We consider the code-word from the codebook $C(G, n)$ as (X^s, X^p) and (Y^s, Y^p) taking values (y^s, y^p) denotes the output of the channel (assumed memoryless, without feedback) considering (X^s, X^p) was transmitted. We thus denote the decoding function as $\hat{x}_i^s = \hat{x}_i^s(y^s, y^p)$

$$\begin{aligned}
\hat{x}_i^s &= \underset{x_i^s \in \{0,1\}}{\operatorname{argmax}}\, p(x_i^s | y^s, y^p) \\
&= \underset{x_i^s \in \{0,1\}}{\operatorname{argmax}} \sum_{\sim x_i^s} p(x^s, x^p, \sigma | y^s, y^p) \\
&= \underset{x_i^s \in \{0,1\}}{\operatorname{argmax}} \sum_{\sim x_i^s} p(y^s, y^p | x^s, x^p, \sigma) p(x^s, x^p, \sigma) \\
&= \underset{x_i^s \in \{0,1\}}{\operatorname{argmax}} \sum_{\sim x_i^s} p(y^s, y^p | x^s, x^p, \sigma) p(x^s, x^p, \sigma) \\
&= \underset{x_i^s \in \{0,1\}}{\operatorname{argmax}}\, p(\sigma_0) \sum_{\sim x_i^s} \prod_{j=1}^{n+m} \underbrace{p(y_j^s | x_j^s) p(y_j^p | x_j^p)}_{\text{channel}} \underbrace{p(x_j^s)}_{\text{prior}} \underbrace{p(x_j^p, \sigma_j | x_j^s, \sigma_{j-1})}_{\text{allowed transitions}}
\end{aligned}$$

Forney style factor graph representation

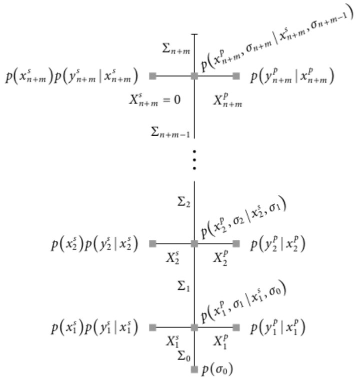


Figure: FSFG for MAP decoding of C(G,n)

Forney style factor graph representation

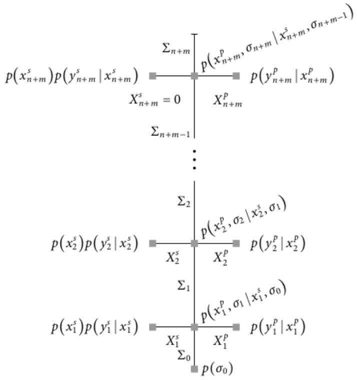


Figure: FSFG for MAP decoding of C(G,n)

This is essentially the BCJR algorithm and the bitwise MAP estimate can be computed at each step using 2 flows of messages one from the top and one from bottom (denoted by α flow and β flow) and the decision step which is called the γ-step.

Flows of information in BCJR algorithm

- Let us define the α-flow. The term $\alpha_{\Sigma_i}(\sigma_i)$ is defined as $p(\sigma_i, y_1^s, y_2^s, ..y_i^s, y_1^p, y_2^p, ..., y_i^s)$ which can be shown to be $\sum_{x_i^s, x_i^p, \sigma_{i-1}} \underbrace{p(x_i^p, \sigma_i | x_i^s, \sigma_{i-1})}_{\text{kernel}} \underbrace{p(y_i^s|x_i^s)p(y_i^p|x_i^p)p(x_i^s)\alpha_{\Sigma_{i-1}}(\sigma_{i-1})}_{\text{product of incoming messages}}$

Flows of information in BCJR algorithm

- Let us define the α-flow. The term $\alpha_{\Sigma_i}(\sigma_i)$ is defined as $p(\sigma_i, y_1^s, y_2^s, ..y_i^s, y_1^p, y_2^p, ..., y_i^s)$ which can be shown to be $\sum_{x_i^s, x_i^p, \sigma_{i-1}} \underbrace{p(x_i^p, \sigma_i | x_i^s, \sigma_{i-1})}_{\text{kernel}} \underbrace{p(y_i^s|x_i^s) p(y_i^p|x_i^p) p(x_i^s) \alpha_{\Sigma_{i-1}}(\sigma_{i-1})}_{\text{product of incoming messages}}$
- Let us define the β-flow. The term $\beta_{\Sigma_i}(\sigma_i)$ is defined as $p(y_{i+1}^s, ..., y_{n+m}^s, y_{i+1}^p, ..., y_{n+m}^p | \sigma_i)$ which can be shown to be

$$\beta_{\Sigma_{i-1}}(\sigma_{i-1}) = \sum_{x_i^s, x_i^p, \sigma_{i-1}} \underbrace{p(x_i^p, \sigma_i | x_i^s, \sigma_{i-1})}_{\text{kernel}} \underbrace{p(y_i^s|x_i^s) p(y_i^p|x_i^p) p(x_i^s) \beta_{\Sigma_i}(\sigma_i)}_{\text{product of incoming messages}}$$

Flows of information in BCJR algorithm

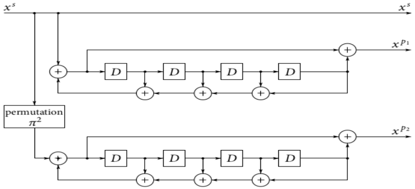
- Let us define the α-flow. The term $\alpha_{\Sigma_i}(\sigma_i)$ is defined as $p(\sigma_i, y_1^s, y_2^s, ..y_i^s, y_1^p, y_2^p, ..., y_i^s)$ which can be shown to be $\sum_{x_i^s, x_i^p, \sigma_{i-1}} \underbrace{p(x_i^p, \sigma_i | x_i^s, \sigma_{i-1})}_{\text{kernel}} \underbrace{p(y_i^s|x_i^s) p(y_i^p|x_i^p) p(x_i^s) \alpha_{\Sigma_{i-1}}(\sigma_{i-1})}_{\text{product of incoming messages}}$
- Let us define the β-flow. The term $\beta_{\Sigma_i}(\sigma_i)$ is defined as $p(y_{i+1}^s, ..., y_{n+m}^s, y_{i+1}^p, ..., y_{n+m}^p | \sigma_i)$ which can be shown to be

$$\beta_{\Sigma_{i-1}}(\sigma_{i-1}) = \sum_{x_i^s, x_i^p, \sigma_{i-1}} \underbrace{p(x_i^p, \sigma_i | x_i^s, \sigma_{i-1})}_{\text{kernel}} \underbrace{p(y_i^s|x_i^s) p(y_i^p|x_i^p) p(x_i^s) \beta_{\Sigma_i}(\sigma_i)}_{\text{product of incoming messages}}$$

- Let us define the decision step at each bit. We can show that the decision step can be written at every bit as $p(x_i^s, y_1^s, ..y_n^s, y_1^p, ..y_n^p) =$ $p(x_i^s) p(y_i^s|x_i^s) \sum_{\sim x_i^s} \underbrace{p(x_i^p, \sigma_i | x_i^s, \sigma_{i-1})}_{\text{kernel}} p(y_i^p|x_i^p) \beta_{\Sigma_i}(\sigma_i) \alpha_{\Sigma_{i-1}}(\sigma_{i-1})$

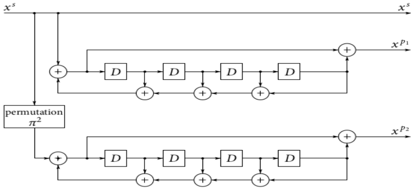
Concatenated codes

We consider a parallel con-catenated code. Here we permute the input sequence through permutations denoted by π^1 and π^2. Similar to the previous example we denote $x = (x_1, x_2, ..., x_n, \underbrace{0, 0..0}_{\text{m times}})$ as input to the filter $G(D)$ and the feedback is removed for the last m steps.



Concatenated codes

We consider a parallel con-catenated code. Here we permute the input sequence through permutations denoted by π^1 and π^2. Similar to the previous example we denote $x = (x_1, x_2, ..., x_n, \underbrace{0, 0..0}_{\text{m times}})$ as input to the filter $G(D)$ and the feedback is removed for the last m steps.
Thus, we denote the code as $C(G, n, \pi) = \{(x, \gamma(\pi^1(x)), \gamma(\pi^2(x))) : x = (x_1, ..x_n, \underbrace{0, ..., 0}_{\text{m times}}); x_i \in \mathbb{F}_2\}$. For a fixed G and n, $\mathcal{P}(G, n)$ denotes the ensemble of codes generated by varying π over all the ensembles on $n + m$ bits that fix the last m bits and ensure uniform probability distribution. Thus $C(G, n, \pi)$ is a code chosen at random from the ensemble.



Punctured turbo codes

- Often, we wish to puncture(delete) certain bits to adjust the rate of the code. For example if every second bit of x^{p1} and x^{p2} is deleted, we get a rate $\frac{1}{2}$ code.
- More generally, we wish to have codes in range $\frac{1}{3} \leq r \leq 1$. For this, we may puncture each bit of parity stream with probability $\frac{3r-1}{2r}$.

Punctured turbo codes

- Often, we wish to puncture(delete) certain bits to adjust the rate of the code. For example if every second bit of x^{p1} and x^{p2} is deleted, we get a rate $\frac{1}{2}$ code.
- More generally, we wish to have codes in range $\frac{1}{3} \leq r \leq 1$. For this, we may puncture each bit of parity stream with probability $\frac{3r-1}{2r}$.

This ensemble of codes is denoted by $\mathcal{P}(G, n, \pi, r)$

Bit- Wise MAP Decoding

The bit-wise MAP decoding can be decoded as:

$$\begin{aligned}
&\hat{x}_i^{\mathrm{MAP}}(y^s, y^{p1}, y^{p2}) \\
&= \operatorname*{argmax}_{x_i^s \in \{0,1\}} \sum_{\sim x_i^s} p(x^s, x^{p1}, x^{p2}, \sigma^1, \sigma^2, y^s, y^{p1}, y^{p2}) \\
&= \operatorname*{argmax}_{x_i^s \in \{0,1\}} \sum_{\sim x_i^s} \prod_{j=1}^{n+m} \underbrace{p(x_j^s)}_{\text{prior}} \underbrace{p(y_j^s|x_j^s) p(y_j^{p1}|x_j^{p1}) p(y_j^{p2}|x_j^{p2})}_{\text{channel}} \\
&\quad p(\sigma_0^1) p(\sigma_0^2) \prod_{j=1}^{n+m} \underbrace{p(x_j^{p1}, \sigma_j | x_j^{s1}, \sigma_{j-1})}_{\text{code 1}} \underbrace{p(x_j^{p}, \sigma_j | x_j^{s2}, \sigma_{j-1})}_{\text{code 2}}
\end{aligned}$$

Note that we denote $x^{s1} = \pi^1(x^s)$ and $x^{s2} = \pi^2(x^s)$

FSFG denoting the decoding equation

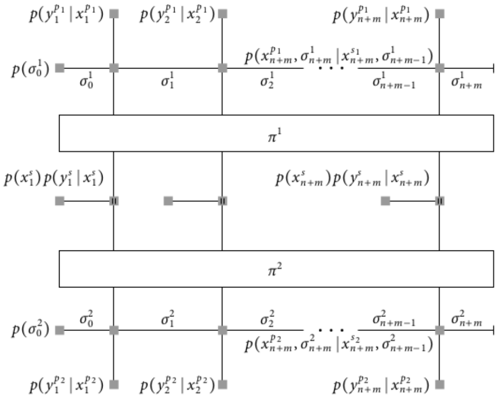


Figure: FSFG for the optimum bit wise decoding

However, it is important to note that this graph is not a tree, thus contains cycles, hence the schedule of decoding process might matter which we shall see in the subsequent sections.

Turbo schedule

- Let us denote each of the convolutional code as a component. In each iteration, we freeze the messages in one component and decode the other component by running a complete iteration of BCJR algorithm. We initialise both the messages with zero.

Turbo schedule

- Let us denote each of the convolutional code as a component. In each iteration, we freeze the messages in one component and decode the other component by running a complete iteration of BCJR algorithm. We initialise both the messages with zero.
- For example, in the first iteration,we freeze the messages in the second component and run BCJR algorithm in the first component.

Turbo schedule

- Let us denote each of the convolutional code as a component. In each iteration, we freeze the messages in one component and decode the other component by running a complete iteration of BCJR algorithm. We initialise both the messages with zero.
- For example, in the first iteration,we freeze the messages in the second component and run BCJR algorithm in the first component.
- In the second iteration, we freeze the messages in the first component and run BCJR algorithm over the first component.

Turbo schedule

- Let us denote each of the convolutional code as a component. In each iteration, we freeze the messages in one component and decode the other component by running a complete iteration of BCJR algorithm. We initialise both the messages with zero.
- For example, in the first iteration,we freeze the messages in the second component and run BCJR algorithm in the first component.
- In the second iteration, we freeze the messages in the first component and run BCJR algorithm over the first component.
- However, note that that in this iteration, the messages that flow through permutation π^1 are no longer zero unlike the previous iteration.

Some simplifications

- The decoder is symmetric since MAP decoding is performed on a linear code. Thus, output is symmetric given all input densities are symmetric if transmission takes over a BMS channel.

Some simplifications

- The decoder is symmetric since MAP decoding is performed on a linear code. Thus, output is symmetric given all input densities are symmetric if transmission takes over a BMS channel.
- We consider a windowed decoding algorithm i.e. instead of running the BCJR algorithm over the whole section, we run it extending w sections to the left and w sections to the right.

Some simplifications

- The decoder is symmetric since MAP decoding is performed on a linear code. Thus, output is symmetric given all input densities are symmetric if transmission takes over a BMS channel.
- We consider a windowed decoding algorithm i.e. instead of running the BCJR algorithm over the whole section, we run it extending w sections to the left and w sections to the right.

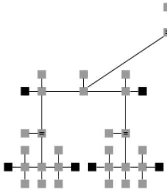


Figure: Computation graph corresponding to (w=1) iterative decoding

Density evolution

- As we discussed in the previous example with fixed trellis, the quantity that would be computed is $\lim_{l\to\infty}\lim_{w\to\infty}\lim_{n\to\infty}\mathbb{E}_{\mathcal{P}(\mathcal{G},n)}[P_b(C,a,l)]$ which can actually be shown to be equal to $\lim_{w\to\infty}\lim_{l\to\infty}\lim_{n\to\infty}\mathbb{E}_{\mathcal{P}(\mathcal{G},n)}[P_b(C,a,l)]$

Density evolution

- As we discussed in the previous example with fixed trellis, the quantity that would be computed is $\lim_{l\to\infty}\lim_{w\to\infty}\lim_{n\to\infty}\mathbb{E}_{\mathcal{P}(\mathcal{G},n)}[P_b(C,a,l)]$ which can actually be shown to be equal to $\lim_{w\to\infty}\lim_{l\to\infty}\lim_{n\to\infty}\mathbb{E}_{\mathcal{P}(\mathcal{G},n)}[P_b(C,a,l)]$
- Let us define the following maps on a an infinite trellis defined by $G(D)$ as shown in Fig. 26. Note that the systematic bits pass through a channel of L-density a and parity bits pass through channel of L-density b. The resulting outgoing densities for the systematic and parity bits are denoted by c and d.

Density evolution

- As we discussed in the previous example with fixed trellis, the quantity that would be computed is $\lim_{l\to\infty}\lim_{w\to\infty}\lim_{n\to\infty}\mathbb{E}_{\mathcal{P}(\mathcal{G},n)}[P_b(C,a,l)]$ which can actually be shown to be equal to $\lim_{w\to\infty}\lim_{l\to\infty}\lim_{n\to\infty}\mathbb{E}_{\mathcal{P}(\mathcal{G},n)}[P_b(C,a,l)]$
- Let us define the following maps on a an infinite trellis defined by $G(D)$ as shown in Fig. 26. Note that the systematic bits pass through a channel of L-density a and parity bits pass through channel of L-density b. The resulting outgoing densities for the systematic and parity bits are denoted by c and d.

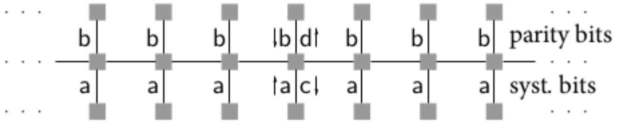


Figure: Definition of $c = \Gamma_G^s(a,b)$ and $d = \Gamma_G^p(a,b)$ of a bi-directional channel of rational function $G(D)$

Density evolution(contn..)

Consider a binary polynomial $p(D)$ of $\deg(p)$. We consider the time-reversed polynomial $\hat{p}(D) = D^{\deg(p)} p(\frac{1}{D})$ and it is extended for $q(D)$ as well and thus define $\hat{G}(D) = \frac{\hat{p}(D)}{\hat{q}(D)}$.
The following theorem characterises density evolution for parallel concatenated codes.

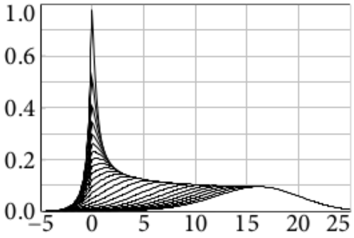
Theorem

Consider a density evolution $\mathcal{P}(G, r)$ when transmission takes place over a BMS-channel with L-density a_{BMSC} and the turbo schedule is used. Let c_l denote the density from the trellis towards systematic bits in the l^{th} iteration. Then for $c_0 = \Delta_0$, we have $c_l = \Gamma_G^s(a_{BMSC} \circledast c_{l-1}, a_{BMSC})$

.

This theorem can be proven using the definition of the algorithm of turbo schedule However, it is difficult to compute Γ_G^s as the states might be in $2^m - 1$ dimensions where m is the memory of channel, hence often determined by sampling.

Example of density evolution



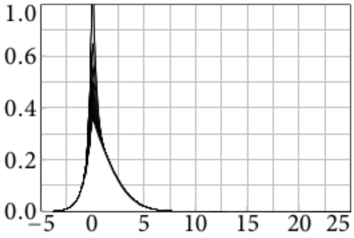


Figure: Evolution of c_l for ensemble $\mathcal{P}(G = \frac{21}{37}, r = \frac{1}{2})$, an alternating puncturing pattern,transmission over BAWGNC(σ). For the left image σ=0.093 where the densities move to the right towards Δ_∞ whereas the right image has $\sigma = 0.95$ where the densities converge to a fixed point density.

Conjecture on stability criterion

Conjecture

Consider the ensemble $\mathcal{P}(G, r)$ and let $D(x, y) = \sum_i d_i(y)x^i$ denote the detour generating function associated with G(taking appropriate puncturing into account). Assume the transmission takes place over BMS channel with $L-$ density a. Then the fixed point corresponding to correct decoding is stable iff $2\mathcal{B}(a)d_2(\mathcal{B}(a)) < 1$.

Recall that detour denote the codewords which originate from state zero at time zero , diverge in the first transition from the zero state and stops the time it returns in the zero state.

EXIT charts

- Note that the densities c_l as defined in previous theorem may have complex representation.
- To understand how fast the decoding process is taking place,we often look at the entropy emitted in every iteration.

Definition

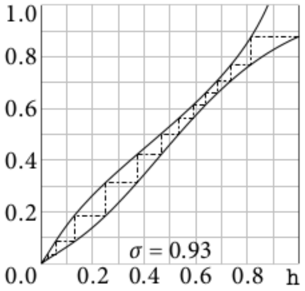
The density evolution process according to the EXIT chart method with respect to channel family $\{a_{\mathrm{BAWGNC(h)}}\}$ can be specified as follows. Let $h_0 = 1$. For $l \geq 1$,

$$h_l = H(\Gamma_G^s(a_{\mathrm{BMSC}} \circledast a_{BAWGNC(h_{(l-1)})}, a_{\mathrm{BMSC}}))$$

. We define h_l as the entropy emitted in the l^{th} iteration according to EXIT chart method The condition for convergence may be stated as:

$$H(\Gamma_G^s(a_{\mathrm{BMSC}} \circledast a_{BAWGNC(h)}, a_{\mathrm{BMSC}})) < h, h \in (0,1)$$

EXIT charts contn..



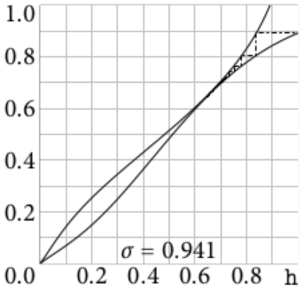


Figure: Exit chart method $\mathcal{P}(G = 21/37, r = 1/2)$ on BAWGNC channel. In the left picture the parameter $\sigma = 0.93$ whereas the right side has $\sigma = 0.941$

Note that the EXIT charts denote the "actions" of each component code during the turbo decoding process, hence it is symmetric unlike the example for LDPC codes.

GEXIT curves

In the previous example, we plotted EXIT curves to visualise the decoding process. Similarly, we can also plot the code as a response to a channel family $\{a_{\mathrm{BMSC(h)}}\}$ which is more difficult to compute thus we corresponding BP-GEXIT curve i.e., we use the densities the output of iterative decoder instead of MAP densities.

Lemma

Consider the ensemble $\mathcal{P}(G, r)$. Assume that transmission takes place over a smooth family of BMS channels ordered by degradation characterised by their L-densities $\{a_{BMSC(h)}\}$ where h denotes the entropy of the channel. For each h, let c_h denote the fixed point density of the density evolution process i.e., $c_h = \Gamma^s(a_{BMSC(h)} \circledast c_h, a_{BMSC(h)})$. Then the BP GEXIT curve is given in parametric form as
$(h, \int (rc_h \circledast c_h + (1-r)\Gamma^p(a_{BMSC(h)} \circledast c_h, a_{BMSC(h)}) l^{BMSC}(y)dy)$ where $l^{BMSC}(y)$ is the GEXIT kernel associated with BMS channel.

GEXIT Curve example

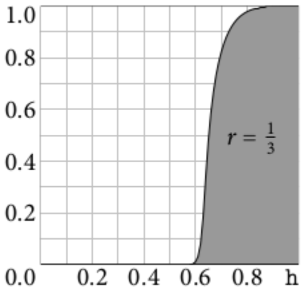


Figure: BP GEXIT curve for the ensemble $\mathcal{P}(G = 7/5, r = 1/3)$ over BAGNC(h). The BP and MAP thresholds coincide. Note that $h^{BP/MAP} = 0.559(\sigma^{MAP/BP} = 1.073)$

Weight distribution

Let $C(G,n)$ denote the convolutional code of input length n defined by the binary rational function G. Let $a_{i,o,n}$ count the number of codewords of input weight i and output weight o. We define the associating function $A(x,y,z) = \sum a_{i,o,n} x^i y^o z^n$. We define $A(x,y,z)$ as the input-output weight generating function of the code $C(G,n)$.

Weight distribution

Let $C(G,n)$ denote the convolutional code of input length n defined by the binary rational function G. Let $a_{i,o,n}$ count the number of codewords of input weight i and output weight o. We define the associating function $A(x,y,z) = \sum a_{i,o,n} x^i y^o z^n$. We define $A(x,y,z)$ as the input-output weight generating function of the code $C(G,n)$.
Consider the transfer matrix $M(x,y)$. Note that it encodes the state transition of the trellis corresponding to rational function G. For example, consider the transfer function of rational function $G = 7/5$.

$$M(x,y) = \begin{array}{c} \\ (00) \\ (01) \\ (10) \\ (11) \end{array} \begin{array}{cccc} (00) & (01) & (10) & (11) \\ \end{array}$$

$$M(x,y) = \begin{array}{c|cccc} & (00) & (01) & (10) & (11) \\ \hline (00) & 1 & xy & 0 & 0 \\ (01) & 0 & 1 & y & x \\ (10) & xy & 1 & 0 & 0 \\ (11) & 0 & 0 & x & y \end{array}$$

Weight distribution

Let $C(G, n)$ denote the convolutional code of input length n defined by the binary rational function G. Let $a_{i,o,n}$ count the number of codewords of input weight i and output weight o. We define the associating function $A(x, y, z) = \sum a_{i,o,n} x^i y^o z^n$. We define $A(x, y, z)$ as the input-output weight generating function of the code $C(G, n)$.
Consider the transfer matrix $M(x, y)$. Note that it encodes the state transition of the trellis corresponding to rational function G. For example, consider the transfer function of rational function $G = 7/5$.

$$M(x,y) = \begin{array}{c} \\ (00) \\ (01) \\ (10) \\ (11) \end{array} \begin{array}{cccc} (00) & (01) & (10) & (11) \\ \end{array}$$

$$M(x,y) = \begin{array}{c|cccc} & (00) & (01) & (10) & (11) \\ \hline (00) & 1 & xy & 0 & 0 \\ (01) & 0 & 1 & y & x \\ (10) & xy & 1 & 0 & 0 \\ (11) & 0 & 0 & x & y \end{array}$$

Recall that for the last m steps the feedback is eliminated and we denote the transfer function as $\bar{M}$ corresponding to the transformer function $p/1$. Thus, $A(x, y, z) = \sum_n [M^n(x, y)\bar{M}(y)]_{0,0} z^n = [(I - zM(x, y))^{-1} \bar{M}^m(y)]_{0,0}$

Lemma on detour generating functions

We can define the regular weight distribution of the code $C(G, n)$ as follows where $A(x, z) = A(x, y = x, z) = \sum_{w,n} x^w z^n$ with coefficients $a_{w,n} = \sum_{i,o:i+o=w} a_{i,o,n}$ which counts $a_{w,n}$ counts the number of codewords of length n and weight w.

Lemma on detour generating functions

We can define the regular weight distribution of the code $C(G, n)$ as follows where $A(x, z) = A(x, y = x, z) = \sum_{w,n} x^w z^n$ with coefficients $a_{w,n} = \sum_{i,o:i+o=w} a_{i,o,n}$ which counts $a_{w,n}$ counts the number of codewords of length n and weight w.

Lemma

Consider the binary rational function G and let $M(x, y)$ be the corresponding function where x encodes the input weight and y the output weight. Let $M^{\cdot}(x, y)$ be equal to $M(x, y)$ except for entry $(0, 0)$ which should be set equal to 0. Let $D(x, y)$ be the generating function counting detours. Then we have

$$D(x, y) = 1 - \frac{1}{[(I - M^{\cdot}(x, y))^{-1}]_{0,0}}$$

Recall that detour denote codeowords that start in state zero, diverges in the first transition state at time zero and stops the first time it reaches state zero.

Concatenated ensembles

Let us talk about the expected weight distribution of convolutional codes where

Concatenated ensembles

Let us talk about the expected weight distribution of convolutional codes where

Theorem

Consider the ensemble $\mathcal{P}(n, G, r)$.Let $A_{\frac{3r-1}{2r}}(x, y, z) = \sum_{i,o,n} x^i y^o z^n$ denote the generating function of input-output weight distribution of $C(G, n)$ under the puncturing of rate $\frac{3r-1}{2r}$ so that the overall rate of the ensemble $\mathcal{P}$ is r. Let $P(x, y, z) = \sum_{i,o,n} p_{i,o,n} x^i y^o z^n$ denote the generating function of the expected input-output weight distribution of $\mathcal{P}(G, n, r)$ where the expectation is taken over all possible interleavers. Then, $p_{i,o,n} = \frac{\sum_j a_{i,j,n} a_{i,o-j,n}}{\binom{n}{i}}$

References

[1] Richardson, T., and Urbanke, R. *Modern Coding Theory*. Cambridge University Press, USA, 2008.

Thank You

THANK
YOU
GRACIAS
ARIGATO
SHUKURIA
DANKSCHEEN
TASHAKKUR ATU
YAQHANYELAY
SUKSAMA
EKHMET
GRAZIE
MEHRBANI
PALDIES
BOLZİN
MERCI
BİYAN
SHUKRIA
TINGKI
GOZAIMASHITA
EFCHARISTO
JUSPAXAR
KOMAPSUMNIDA
MAAKE