Error-correcting codes.

Lecture 5. SISO decoding. BCJR algorithm. Suboptimal decoding
• For memoryless channel ML decoding is reduced to computing and comparing additive metrics of codewords. Metric formula depends on the channel model (Hamming distance, scalar product, etc.)

• Trellis representation of the code reduces ML decoding to search of the shortest path over the graph.

• Minimal trellis of the code, is unique.

• ML decoding is implemented by using the Viterbi algorithm.
Agenda

- Motivation
- Symbol MAP
- Computing symbol probabilities using trellises.
  - BCJR algorithm
- Suboptimal decoding
Assume that the message is encoded by two or more systematic encoders. At the channel output soft decisions from demodulator enter the decoder of the first code. First decoder’s decisions
Assume that the message is encoded by two or more systematic encoders. At the channel output soft decisions from demodulator enter the decoder of the first code. First decoder’s decisions are sent to the second decoder, then again to the first decoder as it would be output of another channel, probably, with less noise since the first decoder corrected some errors ...

In practice, tens of such iterations can be used. This approach is called belief propagation.
Assume that the message is encoded by two or more systematic encoders. At the channel output soft decisions from demodulator enter the decoder of the first code. First decoder’s decisions are sent to the second decoder, then again to the first decoder as it would be output of another channel, probably, with less noise since the first decoder corrected some errors ...

In practice, tens of such iterations can be used. This approach is called belief propagation.

To apply this method we need SISO (soft input, soft output) decoders. The best SISO decoders are maximum a posteriori probability (MAP) decoders.
Let events $H_1, \ldots, H_M$ be non-intersecting, $H_i \cap H_j = \emptyset$, and $P(\bigcup_{m=1}^{M} H_m) = 1$. Then they are considered as a set of hypothesis with respect to event $A$. Law of total probability

$$P(A) = \sum_{m=1}^{M} P(A|H_m)P(H_m)$$

and Bayes’ a posteriori probability formula

$$P(H_j|A) = \frac{P(A, H_j)}{P(A)} = \frac{P(A|H_j)P(H_j)}{\sum_{m=1}^{M} P(A|H_m)P(H_m)}.$$
Let \( C = \{c_m, m = 1, \ldots, M\} \subseteq \{0, 1\}^n \), be a binary block code and \( y = (y_1, \ldots, y_n) \in Y^n \) be a channel output sequence. A posteriori probability of \( c \in \{0, 1\} \) at position \( t \) is

\[
p(c_t = c|y) = \frac{p(c_t = c, y)}{p(y)},
\]

where

\[
p(c_t = c, y) = \sum_{c \in C_t(c)} p(c, y),
\]

and \( C_t(c) \) is a set of codewords which have \( c \) at position \( t \).
Let $C = \{c_m, m = 1, \ldots, M\} \subseteq \{0, 1\}^n$, be a binary block code and $y = (y_1, \ldots, y_n) \in Y^n$ be a channel output sequence. A posteriori probability of $c \in \{0, 1\}$ at position $t$ is

$$p(c_t = c | y) = \frac{p(c_t = c, y)}{p(y)},$$

where

$$p(c_t = c, y) = \sum_{c \in C_t(c)} p(c, y),$$

and $C_t(c)$ is a set of codewords which have $c$ at position $t$. **Question:** What is complexity of computing $p(c_t = c | y)$ in general?
Computing probabilities by using trellises

Hint: compute probabilities for one section taking into account both “past” and “future” sections.
Computing probabilities by using trellises

Hint: compute probabilities for one section taking into account both “past” and “future” sections.

Computational complexity is determined by the trellis complexity (not by the size of the code)
Computing probabilities by using trellises

If we know the probability of each branch, then we will be able to find LLRs by summing up those which correspond to 0 and those which correspond to 1.
Let $s_t$ be a node at layer $t$.

$$
\Pr(s_{t-1} = m; s_t = m' | y) = \frac{\Pr(s_{t-1} = m; s_t = m', y)}{p(y)}.
$$

Only numerator is important for evaluating log-likelihood ratio

$$
\sigma_t(m, m') = \Pr(s_{t-1} = m, s_t = m', y).
$$

Denote $y^j_i = (y_i, y_{i+1}, \ldots, y_j)$
Decompose \((s_{t-1} = m, s_t = m', y)\) to past, current and future as

\[
\sigma_t(m, m') = \Pr \left( (s_{t-1} = m, y_1^{t-1}); (s_t = m', y_t); (y_{t+1}^n) \right).
\]

For memoryless channel from
\[
\Pr(A_1A_2A_3) = \Pr(A_1) \Pr(A_2|A_1) \Pr(A_3|A_1A_2)
\]
follows

\[
\begin{align*}
\Pr(A_1) &= \Pr(s_{t-1} = m, y_1^{t-1}); \\
\Pr(A_2|A_1) &= \Pr(s_t = m', y_t|s_{t-1} = m, y_1^{t-1}) = \Pr(s_t = m', y_t|s_{t-1} = m); \\
\Pr(A_3|A_1A_2) &= \Pr(y_{t+1}^n|s_{t-1} = m, s_t = m', y_1^t) = \Pr(y_{t+1}^n|s_t = m').
\end{align*}
\]
Computing probabilities by using trellises

New notations:

\[ \alpha_t(m) = \text{Pr}(A_1 = s_t = m', y^t_1); \]
\[ \gamma_t(m', m) = \text{Pr}(s_t = m', y_t | s_{t-1} = m); \]
\[ \beta_t(m) = \text{Pr}(y^n_{t+1} | s_t = m') \]

In these terms

\[ \sigma_t(m, m') = \alpha_{t-1}(m) \gamma_t(m, m') \beta_t(m'). \]
We need to know $\alpha_t$, $\gamma_t$, and $\beta_t$. Due to law of total probability

$$\alpha_t(m) = \sum_{m'} \underbrace{\Pr(s_{t-1} = m', y_{1}^{t-1})}_{\alpha} \underbrace{\Pr(s_t = m, y_t|s_{t-1} = m', y_{1}^{t-1})}_{\gamma}.$$ 

Condition $y_1^{t-1}$ not needed given $s_{t-1}$. Thereby, we have recursion

$$\alpha_t(m) = \sum_{m'} \alpha_{t-1}(m') \gamma_t(m', m).$$ 

with initial conditions

$$\alpha_0(m) = \begin{cases} 1, & m = 0; \\ 0, & m \neq 0. \end{cases}$$
Similarly, in inverse direction

\[ \beta_t(m) = \sum_{m'} \Pr(s_{t+1} = m', y_{t+1}^n | s_t = m) = \]

\[ = \sum_{m'} \Pr(s_{t+1} = m', y_{t+1}, y_{t+2}^n | s_t = m) = \]

\[ = \sum_{m'} \Pr(s_{t+1} = m', y_{t+1} | s_t = m, y_{t+2}^n) \times \]

\[ \times \Pr(y_{t+2}^n | s_t = m, s_{t+1} = m', y_{t+1}) = \]

\[ = \sum_{m'} \Pr(s_{t+1} = m', y_{t+1} | s_t = m) \underbrace{\gamma}_{\beta} \Pr(y_{t+2} | s_{t+1} = m'). \]

Recursion for \( \beta \)

\[ \beta_t(m) = \sum_{m'} \beta_{t+1}(m') \gamma_{t+1}(m', m) \]

with initial conditions

\[ \beta_n(m) = \begin{cases} 1, & m = 0; \\ 0, & m \neq 0. \end{cases} \]
Computing $\gamma$:

$$\gamma_t(m, m') = \sum_{c_t} \Pr(s_t = m', c_t, y_t | s_{t-1} = m) = \sum_{c_t} p(c_t | m', m) p(y_t | c_t)$$

and $\gamma_t(m, m') = 0$ for those $(m, m')$, which are not connected in the trellis. For bit-wise trellis

$$\gamma_t(m, m') = p(c_{t,m,m'} ) p(y_t | c_{t,m,m'} )$$

where $c_{t,m,m'}$ is the code symbol associated, with edge $m \rightarrow m'$ at layer $t$. Let $S_t(c)$ be the set of pairs $(m, m')$, to which $c_{t,m,m'} = c$ is assigned.

$$p(c_t = c | y) = \frac{\sum_{(m,m') \in S_t(c)} \sigma_t(m, m')}{p(y)}.$$

Similarly, a posteriori probability for information bits can be written.
Usually, \(p(y)\) are not needed since the SISO decoder output are bit LLRs:

\[
\lambda_t = \ln \frac{p(c_t = 1|y)}{p(c_t = 0|y)} = \frac{\sum_{(m,m') \in S_t(1)} \sigma_t(m,m')}{\sum_{(m,m') \in S_t(0)} \sigma_t(m,m')}
\]

\[
\sigma_t(m,m') = \alpha_{t-1}(m) \gamma_t(m,m') \beta_t(m'), \ m, m' \in \{0, 1, 2, 3\}
\]

\[
\lambda_t = \log \frac{\sigma_t(0,1)+\sigma_t(1,0)+\sigma_t(2,1)+\sigma_t(3,3)}{\sigma_t(0,0)+\sigma_t(1,2)+\sigma_t(2,3)+\sigma_t(3,1)}
\]
**BCJR algorithm**

**Input:** Channel output sequence $y$, a priori probability $u_1, \ldots, u_k$ or codeword symbols $c_1, \ldots, c_n$.

**Output:** LLRs $\lambda_1, \ldots, \lambda_n$.

**Initialization:** For all $t = 1, 2, \ldots, n$ and pairs $(m, m')$ compute $\gamma_t(m, m')$.

1. **Forward pass:** Starting from layer 0, for all layers and all nodes at each layer compute $\alpha_t(m)$.

2. **Backward pass:** Starting from layer $n$, moving backwards for all layers and all nodes at each layer compute $\beta_t(m)$.

3. Compute all $\sigma_t(m, m')$.

4. Compute all $\lambda_t, t = 1, 2, \ldots, n$. 

Steps 2-4 are performed at one backward pass. Values $\beta_t(m)$ are not kept in decoder memory. Only current values $\beta_t(m)$ are updated.
**BCJR algorithm**

**Input:** Channel output sequence $y$, a priori probability $u_1, \ldots, u_k$ or codeword symbols $c_1, \ldots, c_n$.

**Output:** LLRs $\lambda_1, \ldots, \lambda_n$.

Initialization: For all $t = 1, 2, \ldots, n$ and pairs $(m, m')$ compute $\gamma_t(m, m')$.

1. **Forward pass:** Starting from layer 0, for all layers and all nodes at each layer compute $\alpha_t(m)$.

2. **Backward pass:** Starting from layer $n$, moving backwards for all layers and all nodes at each layer compute $\beta_t(m)$.

3. Compute all $\sigma_t(m, m')$.

4. Compute all $\lambda_t$, $t = 1, 2, \ldots, n$.

Steps 2-4 are performed at one backward pass. Values $\beta_t(m)$, are not kept in decoder memory. Only current values $\beta(m)$ are updated.
\[ G = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix} \]

Consider BSC with \( p_0 = 0.01 \). Channel output \( y = (0, 0, 1, 0, 1) \).

| Information symbols \( u_1, u_2 \) | \( c = (c_1, \ldots, c_5) \) | \( p(y|c) \) | \( p(c|y) = \frac{p(y|c)p(c)}{p(y)} \) |
|-----------------------------------|-----------------|----------------|----------------|
| 00                               | 00000           | \( p_0^2(1-p_0)^3 = 0.00729 \) | 0.45            |
| 01                               | 01011           | \( p_0^3(1-p_0)^2 = 0.00081 \) | 0.05            |
| 10                               | 11100           | \( p_0^3(1-p_0)^2 = 0.00081 \) | 0.05            |
| 11                               | 10111           | \( p_0^2(1-p_0)^3 = 0.00729 \) | 0.45            |

\[
p(y) = \sum_{m=0}^{3} p(y|c_m)p(c_m) = 0.00405
\]

| Символы | Output LLRs | \( \ln \frac{p(1|y)}{p(0|y)} \) |
|---------|-------------|----------------|
| \( u_1 \) |             | 0              |
| \( u_2 \) |             | 0              |
| \( c_1, c_3, c_4, c_5 \) |     | 0              |
| \( c_2 \) |             | -2.1972        |
\[ y = \begin{array}{cccc}
0 & 0 & 1 & 01 \\
0 & 00 & 0 & 00 \\
1 & 10 & 11 & 11 \\
1 & 01 & 1 & 11 \\
\end{array} \]

\( a) \) The trellis and channel output sequence \( y \)

\[ \begin{array}{ccc}
0 & 0 & 0 \\
0 & 00 & 00 \\
1 & 10 & 11 \\
1 & 01 & 11 \\
\end{array} \]

\[ \begin{array}{c}
\frac{1}{2}(1 - p_0) \\
\frac{1}{2}p_0^2 \\
\frac{1}{2}(1 - p_0)p_0 \\
(1 - p_0)p_0 \\
\end{array} \]

\( b) \) Formulas for \( \gamma \)

\[ \begin{array}{ccc}
0.45 & 0.045 & 0.09 \\
0.05 & 0.005 & 0.045 \\
0.405 & 0.09 & 0.09 \\
\end{array} \]

\( b) \) Counting \( \gamma \)
Example

б) Counting $\gamma$

g) Counting $\alpha$

д) Counting $\beta$
Example

Counting $\alpha$

Counting $\beta$

Counting $\sigma$
Thick edges correspond to information symbol 1.

Code symbols processing:
For example, $c_2 = 0$ is assigned to edges $0 \rightarrow 0$ and $1 \rightarrow 1$. Other two edges of section 2 correspond to $c_2 = 1$. Therefore,

$$\ln \frac{0.0002025 + 0.0002025}{0.0018225 + 0.0018225} = -2.1972,$$
• SISO decoding is used for exchange information between decoders used in different concatenated constructions: product codes, concatenated codes, turbo-codes, LDPC codes, etc.

• Optimal SISO decoder is symbol-MAP decoder.

• For codes represented by trellises symbol-MAP decoding is performed by using BCJR decoding algorithm

• Complexity of the BCJR decoding is 2 times higher than that of the Viterbi decoding. It makes 2 passes, forward and backward.

• Unlike ML decoding, BCJR decoding requires additions and multiplications. In practice, simplified versions operating in log domain are used.
Suboptimum decoding. Overview

For the methods above, the log(decoding complexity) for a $(n, k)$-code is of order

$$\kappa = \min\{k, n - k, \log \text{trellis complexity}\}$$

both for soft and hard decisions.

Reduced complexity near-ML soft-input decoders

- Information set decoding, Ordered statistic decoding.
- Meet-in-the-middle, Box-and-Match, BEAST.
- Using algebraic decoding with reliability information (Generalized minimum distance (Forney), Chase decoding)
- Reduced-complexity search over trellises: M-algorithm, Tail-biting codes (WAVA).
- Zero-neighbors algorithm
Asymptotically, among long random codes there exist near optimum codes (in sense of $d_{\text{min}}$) such that there minimal trellis has complexity at most

$$\kappa(R) \geq \begin{cases} H(2\delta) - H(\delta), & R > 1 - H(0.25), \\ 1 - H(\delta), & R \leq 1 - H(0.25) \end{cases}$$

where $H(x) = -x \log 2x - (1 - x) \log 2(1 - x)$ and $R = 1 - H(\delta)$, which means that the code achieves Varshamov-Gilbert bound on minimum distance.
Definitions: A set $S_{\text{inf}} = \{i_1, \ldots, i_k\}$ of size $k$ such that the columns of $G$ with indices from $S_{\text{inf}}$ are linearly independent is called \textit{information set (IS)}. Complement to information set $S_{\text{inf}} = \{1, \ldots, n\} \setminus S_{\text{check}}$ is called \textit{check set}.

Or:

A set $S_{\text{check}} = \{i_1, \ldots, i_r\}$ of size $r = n - k$ such that the columns of $H$ with indices from $S_{\text{check}}$ are linearly independent is called \textit{check set (CS)}. Complement to check set $S_{\text{inf}} = \{1, \ldots, n\} \setminus S_{\text{check}}$ is called \textit{information set}.

Idea: IS decoding algorithm exploits the fact that the entire codeword can be recovered from code symbols of information set.
Decoding is reduced to finding the error-free IS.

Options

- Try many ISs at random. If you are lucky, you will find one which is error-free.
- Take one IS and try to flip some positions with a hope that these positions are in error.
- Use both listed options together.

If there are expected $t$ errors in total, then IS of size $k$ bits with high probability contains approximately $kt/n$ errors.
Algorithm 1: Information set hard decision decoding algorithm

Input: Channel output \( y \), number of errors \( t \), Number of attempts \( I_{\text{max}} \)

Output: Candidate codeword \( c \)

1. Let \( c' \leftarrow 0, I \leftarrow 0 \)

2. while \( (d_H(y, c') > t) \& (I < I_{\text{max}}) \) do

3. Choose at random an information set \( S_{\text{inf}} \).

4. for all possible error combinations of weight \( w \leq \frac{tk}{n} \) on \( S_{\text{inf}} \) do

5. Form a new candidate \( c' \) by flipping \( w \) positions in \( S_{\text{inf}} \).

6. Try to recover the codeword from bits in \( S_{\text{inf}} \);

7. if \( d_H(y, c') \leq t \) then

8. \hspace{1em} break

9. \hspace{1em} \( I \leftarrow I + 1 \);

10. return \( c' \)
For one attempt success probability is \( \frac{{n-k \choose t}}{n \choose t} \)

The probability of I non-successful IS selecting attempts

\[
P_e(I) = \left( 1 - \frac{{n-k \choose t}}{n \choose t} \right)^I
\]

If tends to zero if \( I \) grows with \( n \) fast enough. It is easy to derive corresponds complexity exponent

\[
\kappa_{\text{IS}} = H(\delta) - (1 - R) H \left( \frac{\delta}{1 - R} \right)
\]

where \( \delta = t/n \), \( H(x) = -x \log_2 x - (1 - x) \log_2 (1 - x) \). For ML decoding of codes satisfying VG bound, \( t = d_{\text{min}} \), \( \delta = H^{-1}(1 - R) \).
IS decoding is simpler but it is HD decoding.
We IS decoding to soft-decision channel output. Which IS we should select as a first candidate?
Ordered statistic decoding (OSD)

We IS decoding to soft-decision channel output. Which IS we should select as a first candidate? Obviously, we should start with $k$ most reliable bits (MRB).

Depending on the code rate, decoding may be implemented either in $G$ domain or $H$ domain. If $H$ is in systematic form then error combination on the parity-check part is equal to partial syndrome computed for information part (see Algorithm 2).
Ordered statistic decoding (OSD)

\[ y = (y_1, \ldots, y_n) \]

Parity-check matrix \( H \)

Generator matrix \( G \)

Check set

Information set

decoded codeword

\[ c_L = (c_1, \ldots, c_{n-k}) \quad c_M = (c_{n-k+1}, \ldots, c_n) \]
Algorithm 2: OSD algorithm

**Input:** Channel output $y$, parity-check matrix $H$, reprocessing order $w_{\text{max}}$

**Output:** Candidate codeword $c$

1. Permute $y$ and columns of $H$ in ascending order of reliabilities $|y_i|$
2. Reduce $H$ to systematic form with identity matrix on LRB.
3. Compute hard decisions vector $v = (v_L, v_M)$ and its syndrome $s = Hv^T$
4. $c \leftarrow (v_L + s, v_M)$, $E_0 \leftarrow E(c, y)$.
5. **for all possible error combinations of weight $w \leq w_{\text{max}}$ on MRB do**
6. Form a new $v'_M$ by flipping $w$ positions in MRB $v_M$.
7. Modify (recompute) syndrome $s$.
8. Create a new candidate $c' = (v_L + s, v'_M)$.
9. Compute new metric $E(c', y)$.
10. **if** $E(c', y) < E_0$ **then**
11. Update decision: $E_0 \leftarrow E(c', y)$, $c \leftarrow c'$
12. **return** $c$
The OSD is applied to submatrix $B$ with exploiting the meet-in-the-middle principle.

\[ y = (y_1, \ldots, y_n) \]
\[ v = (v_1, \ldots, v_n) = (v_L, v_M) \]

Parity-check matrix $H$

\[ H_0 \]

Decoded codeword

\[ c_L = (c_1, \ldots, c_{n-k}) \quad c_M = (c_{n-k+1}, \ldots, c_n) \]
We start with split syndrome decoding algorithm. Assume that \( t = \delta n \) errors are to be corrected, by \((n, k)\)-code of rate \( R = k/n \). Let us split the received sequence \( y \) into two parts, \( y = (y_1, y_2) \). With high probability two parts of codeword contain \( t/2 \) errors each. Compute two sets \( S_1 \) and \( S_2 \) of partial syndromes, corresponding to all possible combinations of weight \( t/2 \) at each half-codeword. Next, sort and match two sets to find half-words such that corresponding partial syndromes satisfy equality

\[
    s = s_1 + s_2 = 0.
\]

Asymptotically, with high probability, both memory and search complexity are of order

\[
    \mu \approx \left( \frac{n/2}{\delta n/2} \right) \sim \exp_2 \left\{ \frac{n}{2} H(\delta) \right\}
\]

Clearly, more than one attempt of splitting codeword into two parts may be needed.
For codes satisfying VG bound for $t = d_{\text{min}}$ the complexity exponent is

$$\kappa = \frac{1 - R}{2}.$$
Assume that $t$ errors among MSB are to be corrected. The straightforward search complexity is

$$\binom{k + l}{t}.$$

Due to meet-in-the-middle idea complexity drops to

$$\sqrt{\binom{k + l}{t}},$$

both memory and computations.
Algorithm 3: Advanced OSD algorithm

Input: Channel output \( y \), parity-check matrix \( H \), parameter \( l \), search order \( t = w_{\text{max}} \).
Output: Candidate codeword \( c \).

1. Permute \( y \) and columns of \( H \) in ascending order of reliabilities \( |y_i| \).
2. Reduce \( H \) to “almost systematic” form with identity matrix on LRB.

\[
H = \begin{pmatrix} I_{r-l} & 0 & A \\ 0 & B_1 & B_2 \end{pmatrix}
\]

3. By zero-order OSD decoding obtain the first candidate \( c \) and its metric \( E_0 \leftarrow E(c, y) \).
4. Compute sub-syndrome \( s_0 \).
5. Generate a list of candidate MRB sequences for which \( s_0 = 0 \) by using syndrome-split decoding correcting \( w_{\text{max}} \) errors at positions \( r - l + 1, \ldots, n \).

6. for each candidate MSB do
7.     Recover a codeword \( c' \)
8.     Compute metric \( E(c', y) \).
9.     if \( E(c', y) < E_0 \) then
10.        Update decision: \( E_0 \leftarrow E(c', y) \), \( c \leftarrow c' \)
11. return \( c \)
BEAST=bidirectional efficient algorithm for searching trees.

Idea:
Let $\mu(c, y)$ be nonnegative distance function between channel output $y$ and codeword $c$. Let $T = \arg \min_c \mu(c, y)$. Then there exists a trellis level $\ell$ such that

\[
\mu(\hat{c}_t^1, y_t^1) < T/2; \\
\mu(\hat{c}_{t+1}^n, y_{t+1}^n) \geq T/2,
\]

BEAST moves in two directions (forward from root and backward from root) and constructs full trees of codewords with metric $\approx T/2$. If two halves match (reach the same node of the trellis) then we have a codeword candidate.
If $T$ is known then

- Generate forward set $F$ of paths with $\mu(\hat{c}_1^t, y_1^t) < T/2$
- Generate backward set $B$ such that $\mu(\hat{c}_{t+1}^n, y_{t+1}^n) \geq T/2$.
- Sort $F$ and $B$ in ascending number of reached trellis state.
- Find matching: state the same and sum of lengths is equal to $n$.

**Sorting** is a key point. If sizes of two lists are $M_f, M_b$ then matching complexity is

$$\begin{cases} M_f \times M_b & \text{without sorting} \\ M_f \log M_f + M_b \log M_b & \text{with sorting} \end{cases}$$

BEAST has the “peak” complexity equal to the trellis complexity, but extremely low average complexity at high SNRs.
Consider linear \((n, k)\)-code with minimum distance \(d_{\text{min}}\) and assume that for this code there exists low-complexity algorithm correcting all error patterns of Hamming weight up to \(\lfloor (d_{\text{min}} - 1)/2 \rfloor\) errors. In general, the following steps are performed

1. Make **hard decisions** and sort positions in ascending order of their reliabilities.

2. Create **list of candidate** codewords by flipping or erasing some of unreliable positions.

3. For each candidate **decode** corresponding hard-decision vector in order to obtain the candidate codeword.

4. For each candidate codeword **compute its metric** and choose one with the best metric.
Chase and GMD decoding

The step 2 determines complexity/error probability tradeoff.

**Chase 1:** Try all possible $2^{d_{\text{min}}-1}$ error patterns on $d_{\text{min}} - 1$ LRBs.

**Chase 2:** Try all possible $2^{\lfloor d_{\text{min}}/2 \rfloor}$ error patterns on $\lfloor d_{\text{min}}/2 \rfloor$ LRBs.

**Chase 3:** If $d_{\text{min}}$ is even then flip $0, 1, 3, \ldots$, $d_{\text{min}} - 1$ LRBs. Otherwise, flip $0, 2, 4, \ldots$, $d_{\text{min}} - 1$ LRBs.

**GMD:** The same as Chase 3 with erasures instead of flipping LRBs.

For Chase 3 and GMD only $\approx d_{\text{min}}/2$ decoding attempts are performed. Chase 2 is practical only for very small $d_{\text{min}}$.

**Remark:** All versions admit earlier termination based on some optimality criteria.
For \((n, k)\)-code \(C\) let \(R(c)\) be domain of \(c \in C\), and \(D(c)\) be vicinity, i.e. the set of sequences \(x \in F_2^n \setminus R(c)\) such that by flipping one bit we obtain \(x' \in R(c)\).

The set of codewords \(Z \subseteq \{C \setminus 0\}\) called the set of zero neighbors if it is minimal set such that

\[
D(0) \subseteq \bigcup_{c \in Z} R(c).
\]

Thus, domains of zero neighbors either overlap with zero-domain or differ from zero-domain members in at most one bit.
Zero neighbors (ZN) algorithm
The ZN decoder keeps the full list of zero neighbors (ZNs), which is subset of the code. For channel output sequence $y$ decoder finds such ZN (shift) that adding to $y$ produce a codeword of weight smaller than Hamming weight of $y$. Continue until the weight can be further reduced. Final decision is a codeword equal to the sum of all shifts. The maximum number of steps (shifts) grows \textit{linearly} with codelength. (Why?) The complexity of a single step is proportional to the number of ZNs which grows \textit{exponentially} with codelength.
ZN algorithm

Input: Channel output $y$;
Output: Information sequence;

Initialization: Flag=1; $\hat{c} = 0$;

while Flag==1 do

Flag=0;

foreach $v \in Z$ do

if $w(x + v) < w(x)$ then

$\hat{c} \leftarrow \hat{c} + v$;

$x \leftarrow x + v$;

Flag=1;

break;

return: Information bits of $\hat{c}$
Surprisingly, asymptotic complexity is exactly the same as for trellis-base decoding

$$\kappa_Z(R) \leq \begin{cases} H(2\delta) - H(\delta), & R > 1 - H(0.25), \\ 1 - H(\delta), & R \leq 1 - H(0.25). \end{cases}$$

for codes satisfying Varshamov-Gilbert bound.

$$H(x) = -x \log_2(x) - (1 - x) \log_2(1 - x)$$

**Remark:** The ZN algorithm is analyzed only for hard-decision channel. Reformulating the algorithm and its analysis for soft decisions is an open problem.
Among considered algorithms only BCJR algorithm produces soft decisions. However, most of them produce a list of most probable codewords.

Let \( C = \{c_m, m = 1, \ldots, M\} \subseteq \{0, 1\}^n \), be a binary block code and \( y = (y_1, \ldots, y_n) \in Y^n \) be a channel output sequence. The LLR of of \( c \in \{0, 1\} \) at position \( t \) given \( y \) is

\[
LLR(c_t) = \log \frac{p(c_t = 1|y)}{p(c_t = 0|y)},
\]

where

\[
p(c_t = c, |y) = \sum_{c \in C_t(c)} \frac{p(c, y)}{p(y)},
\]

and \( C_t(c) \) is a set of codewords which have \( c \) at position \( t \).
List-based approximate SISO decoding

Let for a given $\mathbf{y}$, $\mathcal{L} \subset \mathbb{C}$ denote a list of most probable codewords provided by a suboptimal soft-input decoder. Then

$$LLR(c_t) \approx \log \frac{\sum_{\mathbf{c} \in \mathcal{L}_t(1)} p(\mathbf{c}, \mathbf{y})}{\sum_{\mathbf{c} \in \mathcal{L}_t(0)} p(\mathbf{c}, \mathbf{y})},$$

where $\mathcal{L}_t(\mathbf{c})$ are subsets of $\mathcal{L}$ including codewords with $\mathbf{c}$ at position $t$.

The problem appears if for some pair $(t, \mathbf{c})$ the subset $\mathcal{L}_t(\mathbf{c})$ is empty.
Example solutions:

- In case of empty $\mathcal{L}_t(c)$ approximate

\[ p(c_t = c, |y) \approx p(c_t = c, |y_t). \]

- Approximate by least-reliable list entry:

\[ \sum_{c \in \mathcal{L}_t(c)} p(c, y) \approx \min_{c \in \mathcal{L}} p(c, y). \]

These approximations are used in combination with weighting coefficients which depend on the specific code and channel SNR.
1. SISO decoding is needed for implementing iterative decoding algorithms.

2. BCJR algorithm provides optimal (MAP) SISO decoding. It requires two passes. Complexity of each pass is equal to the trellis complexity of the code.

3. Suboptimal simplified SISO decoding can be implemented as list decoding followed by approximated estimating of output LLRs.

4. Suboptimum decoding can be based on information set decoding, Chase or GMD decoding, BEAST, WAVA, etc.

5. Suboptimum decoding is an area for further research.