Parallel Concatenated Codes
– “Turbo Codes”

- Ingredients
- Parallel Code Concatenation
- Iterative Decoder (“Turbo Decoder”)
Motivation

- **Product codes**
  - Encoder: Arrangement of info symbols in matrix, horizontal and vertical encoding
  - Decoder: Iteration between vertical and horizontal decoding

- Generalization of these concepts

- **Parallel Concatenated codes**
  - Encoder: Encoding of original and of interleaved (permuted) info word
  - Decoder: Iteration between constituent decoders using soft-values
Ingredients

- Recursive systematic convolutional (RSC) encoders
- Soft-in soft-out decoders
- Interleavers
**Convolutional Encoder**

**Example:** $(7, 5)_8$ convolutional encoder

![Convolutional Encoder Diagram](image)

Definition via generator polynomials:

- "7" → $g_1(D) = 1 + D + D^2$
- "5" → $g_2(D) = 1 + D^2$

- Memory length $m = 2$
- Encoding rate $R^* = 1/2$
Convolutional Encoder

Example: \((7, 5)_8\) convolutional encoder

Definition via generator polynomials:

- "7" \(\rightarrow g_1(D) = 1 + D + D^2\)
- "5" \(\rightarrow g_2(D) = 1 + D^2\)

- Memory length \(m = 2\)
- Encoding rate \(R^* = 1/2\)
- Encoder state \(s_t = [s_{t,1}, s_{t,2}]\)
Convolutional Encoder

- Trellis Diagram

**Encoder starts in the zero state**

- **Error event**: detour from the all-zero path (zero-state at the beginning and the end but not in between)

- **Free distance** $d_{\text{free}} = 5$: minimum weight of any single error event

- $u_t = 0$: solid line; $u_t = 1$: dashed line
**Convolutional Encoder**

Encoder starts and ends in the zero-state ("trellis termination")

Information word length  \( K = 4 \)
Trellis length  \( T = K + m = 6 \)
Codeword length  \( N = (K + m)/R^* = 12 \)
Code rate  \( R = R^* \frac{K}{K+m} = 1/2 \cdot 4/6 = 4/12 \)
Minimum distance  \( d_{\text{min}} = d_{\text{free}} \)
Convolutional Encoder

Formulation of Encoding via Polynomials

- Information-word polynomial:
  \[ u(D) = u_1 + u_2 D + u_3 D^2 + u_4 D^3 \]

- Generator polynomials:
  \[ g_1(D) = 1 + D + D^2 \]
  \[ g_2(D) = 1 + D^2 \]

- Code-word polynomials:
  \[ x_1(D) = u(D) g_1(D) = x_{1,1} + x_{2,1} D^2 + \cdots + x_{6,1} D^5 \]
  \[ x_2(D) = u(D) g_2(D) = x_{1,2} + x_{2,2} D^2 + \cdots + x_{6,2} D^5 \]
Recursive Systematic Convolutional Encoder

Example: \((1, 5/7)_8\) convolutional encoder

Definition via generator functions:

- “1” \[ \rightarrow \quad g_1(D) = 1 \]
- "5/7" \[ \rightarrow \quad g_2(D) = \frac{1 + D^2}{1 + D + D^2} \]
RSC Encoder

Properties of an RSC Encoder

- Generates the same code as a non-recursive encoder
- Information words of weight 1 generate large (infinite) codeword weights
- Finite codeword weights require information weights of at least 2

Effective free distance $d_{f,\text{eff}}$:

minimum codeword weight for information weight 2
APP decoding

For each info symbol, the decoder computes the a-posteriori probabilities:

\[
Pr(U_k = 0 | y) = \sum_{x \in C: u_k = 0} \alpha \cdot \prod_{n=0}^{N-1} p(y_n | x_n)
\]

\[
Pr(U_k = 1 | y) = \sum_{x \in C: u_k = 1} \alpha \cdot \prod_{n=0}^{N-1} p(y_n | x_n)
\]

BCJR algorithm

Bahl, Cocke, Jelinek, and Raviv found an efficient way to compute the APPs on the code trellis (similar to the Viterbi algorithm)
LogAPP decoding

For each info symbol, the decoder computes the a-posteriori L-value:

\[ L(U_k|y) = \ln \frac{\Pr(U_k = 0|y)}{\Pr(U_k = 1|y)} \]

The logarithmic implementation of the BCJR algorithm has numerical advantages and operates on L-values and log-probabilities.

A sub-optimum variant of LogAPP decoding is MaxLogAPP decoding.

Remark: Using the approximation of boxplus is MaxLogAPP decoding of single-parity-check codes.
Generalized Decoding Model

\[ l_{U,k,\text{ch}} = L(U_k|z_k) \]
\[ l_{U,k,\text{p}} = L(U_k|z, y) = L(U_k|l_{U,\text{ch}}, l_{X,\text{ch}}) \]
\[ l_{U,k,\text{e}} = L(U_k|z_{\backslash k}, y) = L(U_k|l_{U,\text{ch},\backslash k}, l_{X,\text{ch}}) \]
\[ l_{X,n,\text{ch}} = L(X_n|y_n) \]
\[ l_{X,n,\text{p}} = L(X_n|z, y) = L(X_n|l_{U,\text{ch}}, l_{X,\text{ch}}) \]
\[ l_{X,n,\text{e}} = L(X_n|z, y_{\backslash n}) = L(X_n|l_{U,\text{ch}}, l_{X,\text{ch},\backslash n}) \]
Generalized Decoding Model

Relation to “Conventional” Decoding Model

- Extension of encoder by a systematic branch
  (an RSC encoder is extended by an additional(!) systematic branch)

```
\[
\begin{align*}
    \mathbf{u}_t & \quad \rightarrow \quad \mathbf{s}_{t,1} \quad \rightarrow \quad \mathbf{s}_{t,2} \quad \rightarrow \quad \mathbf{x}_{t,0} \\
    \quad & \quad \downarrow \quad \quad \downarrow \quad \quad \downarrow \quad \quad \downarrow \quad \quad \downarrow \\
    \quad & \quad \quad \quad \quad \quad \quad \quad \quad \rightarrow \quad \mathbf{x}_{t,1} \\
    \quad & \quad \downarrow \quad \quad \downarrow \quad \quad \downarrow \quad \quad \downarrow \quad \quad \downarrow \\
    \quad & \quad \quad \quad \quad \quad \quad \quad \quad \rightarrow \quad \mathbf{x}_{t,2}
\end{align*}
\]
```

- “Conventional” decoding of virtual $R^\bullet = 1/3$ encoder
- Notice: A new algorithm is not necessary!
Puncturing may be used to increase the code/encoder rate

- **Operation at the encoder side:**
  puncturing (removing) of some of the symbols

- **Operation at the decoder side:**
  (i) insertion of L-value zero ("don’t know nothing") at the positions of the punctured symbols
  (ii) use of the decoder for the original code/encoder

**Advantage:** the code rate can be increased while the same devices for encoding and decoding are used
Interleavers

- An interleaver permutes the order of a sequence of symbols. The corresponding de-interleaver performs the reverse operation.

- Some authors prefer the term permuter. (Which is actually a better name.)

- Formulation using a function $\pi(.)$:
  
  original sequence $\mathbf{u} = [u_0, u_1, u_2, u_3]$
  
  permuted sequence $\mathbf{v} = [u_{\pi(0)}, u_{\pi(1)}, u_{\pi(2)}, u_{\pi(2)}]$

- Formulation using a matrix $P$:

  $[v_0, v_1, v_2, v_3] = [u_0, u_1, u_2, u_3] \cdot \begin{bmatrix}
  0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 1 \\
  0 & 0 & 1 & 0 \\
  1 & 0 & 0 & 0 
\end{bmatrix}$
**Types of Interleavers**

- **Block Interleaver**
  Symbols are written into a matrix column-wise and read out row-wise.

- **Random Interleaver**
  The permutation pattern is chosen randomly (but only once then it is fixed, of course).

- **S-rand Interleaver**
  The permutation pattern is chosen randomly with a “spreading” constraint given by the parameter $s$:

  $$|\pi(i) - \pi(k)| > s \quad \text{for all } i, k \text{ with } |i - k| < s$$
Parallel Concatenated Codes

**Question:**
Given two LogAPP-decoder modules ... What can you do with them?
Parallel Concatenated Codes

**Question:**
Given two LogAPP-decoder modules ... What can you do with them?

**Answer:**
Invent turbo codes.
Parallel Concatenated Codes

Question:
Given two LogAPP-decoder modules ... What can you do with them?

Answer:
Invent turbo codes.

Next Question:
What encoder constructions are reasonable?
**Typical Settings**

- **Encoder 1**: rate-1/2 RSC convolutional encoder
- **Encoder 2**: rate-1 RSC convolutional encoder (without systematic output)
- Parallel-to-serial device includes puncturing to increase rate from 1/3 to 1/2
**PC Codes - Decoder**

- Serial-to-parallel device includes insertion of \( L \)-values being zero for punctured symbols
- **Decoder 1** and **Decoder 2** are LogAPP decoders
PC Codes - Decoder

L-values for Decoder 1 (for Decoder 2 analogous)

\[
\begin{align*}
l^{(1)}_{n,ch} &= L(X^{(1)}_n | y^{(1)}_n) \\
e_k^{(1)} &= L(U^{(1)}_k | l^{(1)}_{ch}, a^{(1)}_{\backslash k}) \\
l^{(1)}_k &= L(U^{(1)}_k | l^{(1)}_{ch}, a^{(1)})
\end{align*}
\]
Some Remarks

- The constituent decoders perform **local decoding**, i.e., they know only their code constraints.

- The LogAPP decoder 1 computes correct L-values only if all input L-values are **conditionally independent**, i.e., if

$$p(l^{(1)}_{ch}, a^{(1)} | x^{(1)}, u^{(1)}) = \prod_{n=1}^{N^{(1)}} p(l^{(1)}_{n, ch} | x^{(1)}_{n}) \cdot \prod_{k=1}^{K} p(a^{(1)}_{k} | u^{(1)}_{k})$$

(analogous for decoder 2)

- As the independence is fulfilled only for the first few iterations, the decoder outputs are only approximate L-values.


**PC Codes - Design Issues**

- Constituent encoders
- Interleaver (spreading properties)
- Puncturing pattern
- Constituent decoders (LogAPP, MaxLogAPP, ...)
- Number of iterations / stopping criteria
The Original Turbo Encoder

Berrou, Glavieux, Thitimajshima, 1993.

Fig. 2 Recursive Systematic codes with parallel concatenation.
The Original Turbo Decoder

Berrou, Glavieux, Thitimajshima, 1993.

Fig. 3b Feedback decoder (under 0 internal delay assumption).
Performance of the First Turbo-Code

Berrou, Glavieux, Thitimajshima, 1993.

Fig. 5: Binary error rate given by iterative decoding ($p=1, \ldots, 16$) of code of Fig. 2 [rate:1/2; Interleaving 255x255].