Methods for the Reconstruction of Parallel Turbo Codes

M. Cluzeau, M. Finiasz, and J.-P. Tillich
We intercept a noisy bitstream and want to recover the (encrypted) information.
Overview of the problem

- Code reconstruction consists in finding the code and an efficient decoder for the intercepted bitstream,
  - if nothing is known about the encoder, this is generally a hard problem.

- Depending on the type of code, some techniques exist:
  - convolutional codes,
  - linear block codes,
  - LDPC codes.

  [Valembois, Filliol, Barbier, Sendrier, Côte…]

- Here we focus on parallel turbo codes.
We consider rate $\frac{1}{3}$ parallel turbo codes using 2 systematic convolutional encoders and a permutation $\Pi$.

We want to find $P$, $Q$, $P'$, $Q'$ and $\Pi$ from the interleaved outputs $X$, $Y$ and $Z$, with some noise.
First Step of Reconstruction
Isolating the outputs

We apply convolutional code reconstruction techniques:

▷ search short parity check equations valid for offsets of any multiple of $n$ ($n = 3$ for standard interleaving).

▷ they will only involve bits of $X$ and $Y$
  ➔ we can isolate $Z$,
  ➔ with enough equations we can recover $P'$ and $Q'$.

Deciding which of the reconstructed $X$ and $Y$ was indeed $X$ is impossible:

▷ Reconstruction only works for the correct choice:
  ➔ in case of failure we start over.
We can find the block length by using linear block code reconstruction techniques:

- again search for parity check equations,
- longer equations involving bits of $\mathbb{Z}$.

For a permutation of length $N$ and no puncturing, the shortest block length with parity checks equations involving bits of $\mathbb{Z}$ is equal to $3N$. 

Second Step of Reconstruction
Finding the block/permutation length

We can find the block length by using linear block code reconstruction techniques:

▷ again search for parity check equations,
  ➔ longer equations involving bits of $\mathbb{Z}$.

For a permutation of length $N$ and no puncturing, the shortest block length with parity checks equations involving bits of $\mathbb{Z}$ is equal to $3N$.

$N$ can be large, depending on the noise level this step can be very expensive,

▷ synchronization patterns or other similar things can help guess the correct length.
Now one has to recover $P$, $Q$ and $\Pi$ from $X$ and $Z$ with some noise.

- $P$ and $Q$ can be exhaustively searched for,
- recovering $\Pi$ is the hard part.

We propose two methods:

- search for low weight parity check equations,
- guess the positions of $\Pi$ one by one, using a “decoder” to decide which is correct.
Using Parity Checks
The input $X$ is first permuted...
Using Parity Checks

\[ \begin{align*}
X & = 10010101 \\
X_\Pi & = 10001011 \\
Z & = 10010101
\end{align*} \]

...then encoded by \( P/Q \).

\[ \frac{1+D+D^3}{1+D} \]
Using Parity Checks

The same process is applied to each block.
We receive noisy versions of $X$ and $Z$, we want to recover $\Pi$. 
Using Parity Checks

| X        | $X_{II}$  | Z        |
|----------|-----------|----------|
| 10010101 | 100010111 | 10010101 |
| 00100011 | 111000000 | 11110111 |
| 10101100 | 01000111  | 01001000 |
| 01111101 | 11111000  | 11110101 |
| 10001010 | 00100101  | 00100010 |
| 00010111 | 10101010  | 10110011 |

$X_{II}$ and $Z$ are linked by parity check equations.
Using Parity Checks

$X$ and $Z$ are linked by parity check equations,

$X_{\Pi}$ and $Z$ by permuted parity checks.
### Using Parity Checks

| $X$   | $X_{II}$ | $Z$   |
|-------|----------|-------|
| 1 0 0 1 0 1 0 1 | 1 0 0 0 1 0 1 1 | 1 0 0 1 0 1 0 1 |
| 0 0 1 0 0 0 1 1 | 1 1 1 0 0 0 0 0 | 1 1 1 1 0 1 1 1 |
| 1 0 1 0 1 1 0 0 | 0 1 0 0 0 1 1 1 | 0 1 0 0 1 0 0 0 |
| 0 1 1 1 1 0 1 1 | 1 1 1 1 1 1 0 0 | 1 1 1 0 1 0 0 1 |
| 1 0 0 0 1 0 1 0 | 0 0 1 0 0 1 0 1 | 0 0 1 0 0 0 1 0 |
| 0 0 0 1 0 1 1 1 | 1 0 1 0 1 0 1 0 | 1 0 1 1 0 0 1 1 |
|               |           |       |
| 0 1 0 0 1 1 0 0 | 0 0 0 1 0 1 1 0 | 0 0 0 0 0 1 1 0 |
| 0 0 0 1 1 0 1 0 | 0 0 1 0 1 1 0 0 | 0 0 0 0 1 1 0 0 |
| 0 1 1 1 0 0 0 0 | 0 1 0 1 1 0 0 0 | 0 0 0 1 1 0 0 0 |

- **permutation shifts**
- **parity check shifts**

$X_{II}$ and $Z$ are linked by parity check equations, and any shift is also valid.
Each parity check we find gives us information on $P$ and $Q$ and on $\Pi$. 

- $X$
  - $\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \ \end{bmatrix}$
  - $X_{\Pi}$
    - $\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \ \end{bmatrix}$
  - $Z$
    - $\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \ 1 & 1 & 1 & 1 & 0 & 1 & 1 \ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \ \end{bmatrix}$

- $1 + D^2 + D^3 + D^4$
  - $1 + D^2$
    - $\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \ \end{bmatrix}$

- $1 + D^4 + D^5$
  - $1 + D^3$
    - $\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \ \end{bmatrix}$
Each parity check found is of the form $\lambda P$ on the $X_\Pi$ part and $\lambda Q$ on the $Z$ part

- one knows $\lambda Q$ and the weight of $\lambda P$
- it is possible to classify the $P, Q$ pairs depending on their parity checks.

Once $P/Q$ is known, one knows $\lambda P$ too and gets even more information on $\Pi$.

For low noise levels this technique is very efficient.

For higher noise levels, only some parity check equations are found, leaving parts of $\Pi$ unknown.
Using a Convolutional Decoder
For this technique, $P/Q$ has to be known or guessed.

One wants to find the first position $x$ of $\Pi$: $\Pi(x) = 1$

- there are $N$ possibilities,
- for each of the $M$ intercepted blocks, one knows the first output bit of the convolutional encoder $P/Q$ → the first “column” of $Z$
- each of the $N$ “columns” of $X$ corresponds to a different set of input bits.

For each possible value of $x$, one computes the entropy of the internal state of the convolutional encoder $P/Q$,

- $N$ distributions of $M$ samples each.
When guessing $x$ two cases can occur:

- For the correct choice ($\Pi(x) = 1$), the entropy on the encoder state should be quite low
  - directly related to the noise level
- For an incorrect choice ($\Pi(x) \neq 1$), this entropy will be higher
  - equivalent to having an unrelated input bit.

Among the $N$ computed distributions:

- $N - 1$ will follow a “bad” distribution,
- $1$ will follow the “good” distribution.

The “bad” and “good” distributions can be computed through sampling if the noise level is known.
For a Gaussian noise of standard deviation $\sigma$ quite high, the "target" distributions can still be distinguished.
We use a straightforward algorithm:

- the positions of $\Pi$ are recovered sequentially,
- at each step the most “probable” positions are selected using a Neyman-Pearson test:
  - we fix a threshold and keep all candidates above this threshold,
  - at step $i$, we consider the $i - 1$ previous steps were successful:
    - if no position is above the threshold, the candidate is discarded,
- once we reach the end, only a few candidates for $\Pi$ should remain.
Using a Convolutional Decoder

Practical results

| $N$  | $\sigma$ | $M$  | (theory) | running time |
|------|----------|------|----------|--------------|
| 64   | 0.43     | 50   | (48)     | 0.2 s        |
| 64   | 0.6      | 115  | (115)    | 0.3 s        |
| 64   | 1        | 1380 | (1380)   | 12 s         |
| 512  | 0.6      | 170  | (169)    | 11 s         |
| 512  | 0.8      | 600  | (597)    | 37 s         |
| 512  | 1        | 2800 | (2736)   | 173 s        |
| 512  | 1.1      | 3840 | (3837)   | 357 s        |
| 512  | 1.3      | 29500| (29448)  | 4477 s       |
| 10000| 0.43     | 300  | (163)    | 8173 s       |
| 10000| 0.6      | 250  | (249)    | 7043 s       |

Complexity in $\Theta(N^2 M 2^m)$:

- however, the larger $N$, the larger $M$ must be.
We can predict the number of intercepted words required to reconstruct the turbo code:

- for low noise levels only few words are required.

Particularly efficient technique for Gaussian noise:

- the distributions are quite messy for a BSC

Recovery can fail for two reasons:

- the number of candidates explodes
  - happens when $M$ is too small.
- the number of candidates drops to 0
  - bad choice for $P/Q$, or bad luck with the noise distribution.
Further Improvements

- Both techniques can be adapted to punctured turbo codes
  - the complexity will increase significantly (at least by a factor $N$).

- Both methods can be combined:
  - one should always spend a few seconds/minutes searching for low weight parity checks,
  - it helps find $P/Q$, and decreases the cost of the second algorithm.