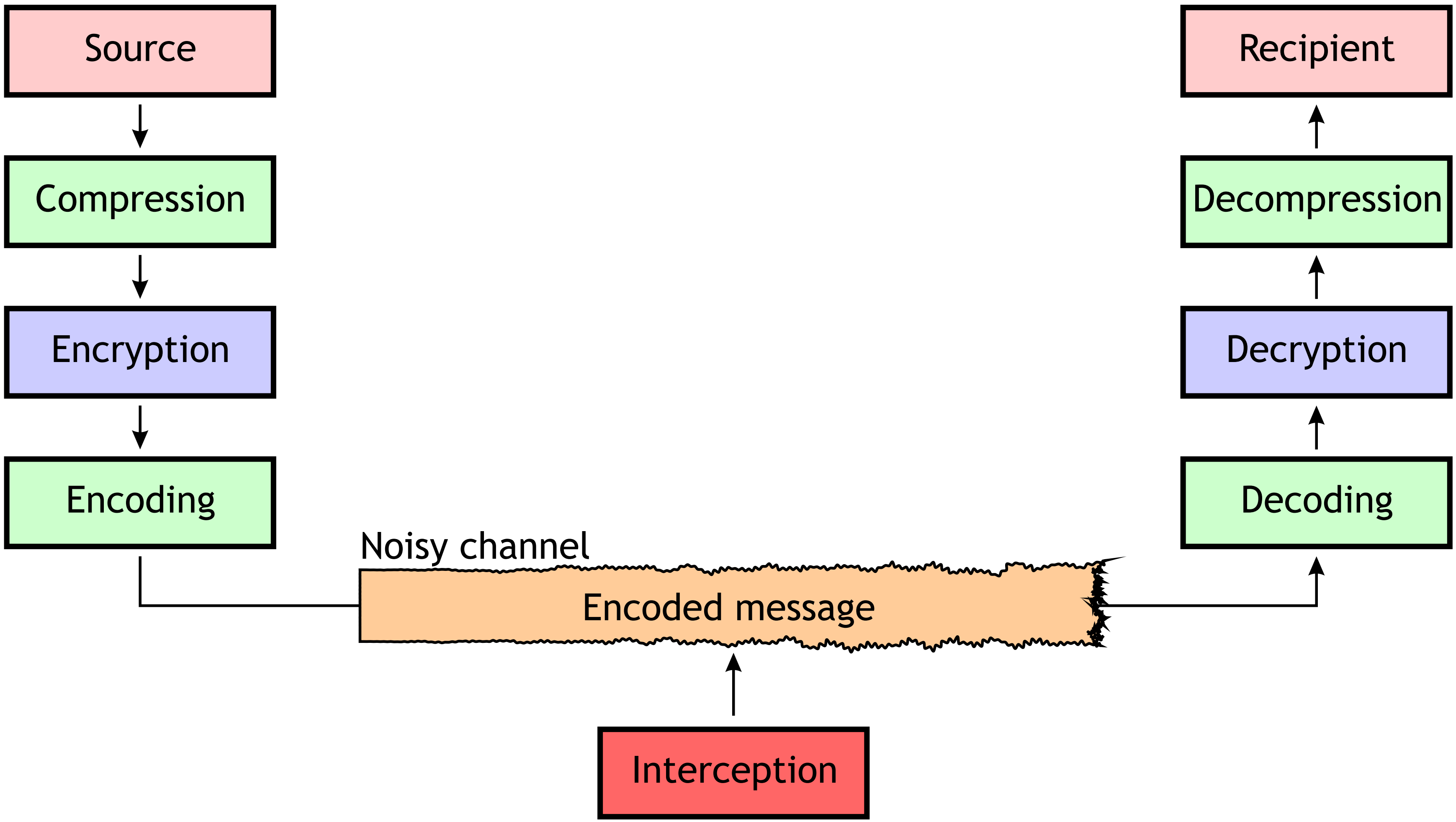
Methods for the Reconstruction of Parallel Turbo Codes

M. Cluzeau, M. Finiasz, and J.-P. Tillich

ENSTA ParisTech

INRIA

Overview of the problem



- We intercept a noisy bitstream and want to recover the (encrypted) information.

Overview of the problem

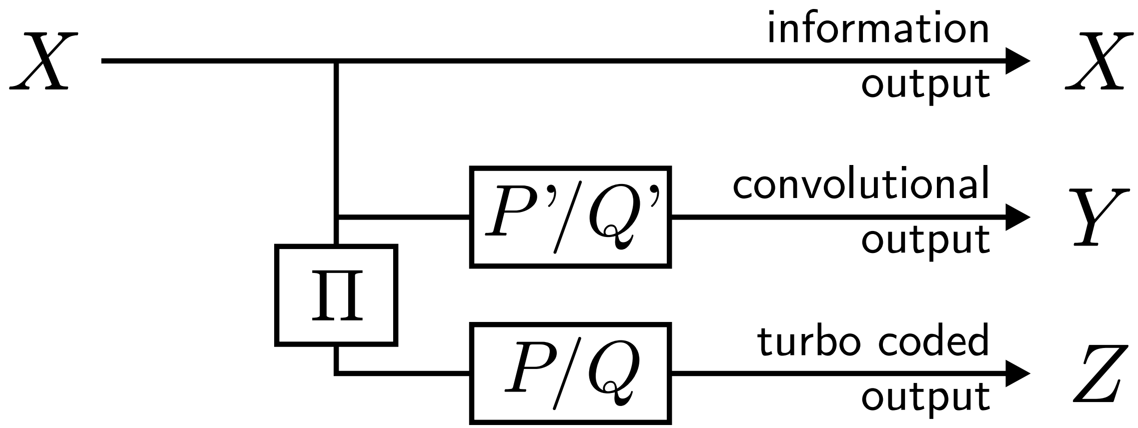
- Code reconstruction consists in finding the code and an efficient decoder for the intercepted bitstream,
  - if nothing is known about the encoder, this is generally a hard problem.

- 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 Π



- We want to find P, Q, P', Q' and Π from the interleaved outputs X, Y and Z, with some noise.

- 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 Z.

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

- We can find the block length by using linear block code reconstruction techniques:
  - again search for parity check equations,
    → longer equations involving bits of Z.

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

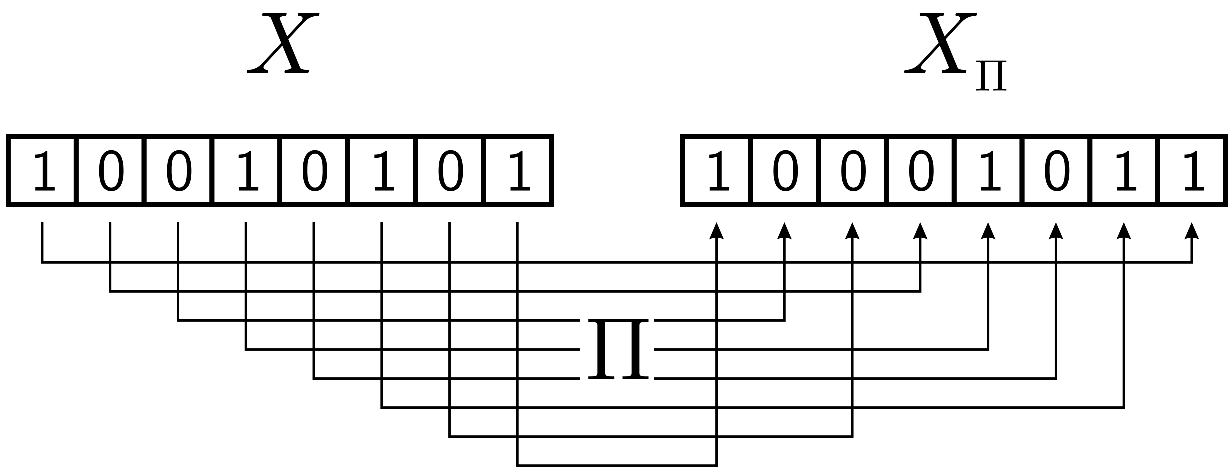
- N can be large, depending on the noise level this step can be very expensive,
  - synchronization patterns or other similar things can help guess the correct length.

Third Step of Reconstruction
Finding everything else...

- Now one has to recover P, Q and Π from X and Z with some noise.
  - P and Q can be exhaustively searched for,
  - recovering Π is the hard part.

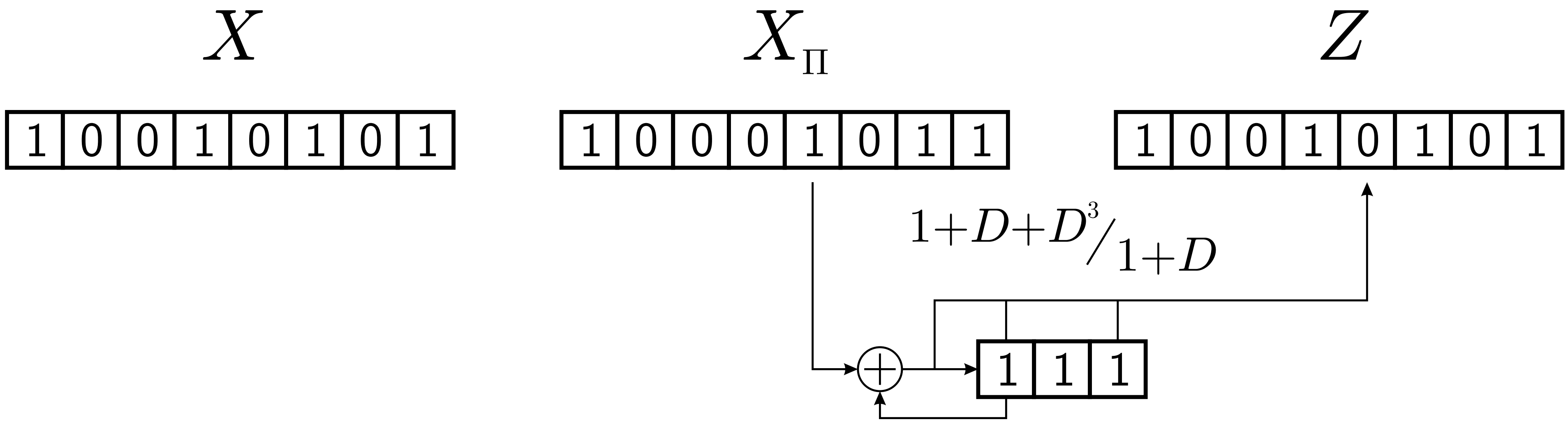
- We propose two methods:
  - search for low weight parity check equations,
  - guess the positions of Π one by one, using a “decoder” to decide which is correct.

Using Parity Checks



- The input X is first permuted...

Using Parity Checks



- …then encoded by P/Q.

Using Parity Checks

X

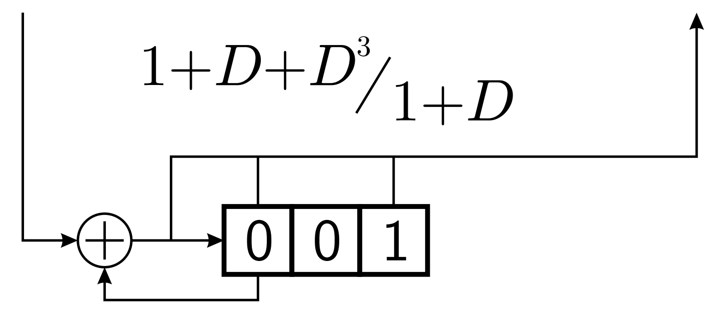
1	0	0	1	0	1	0	1
0	0	1	0	0	0	1	1
1	0	1	0	1	1	0	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	1	0
0	0	0	1	0	1	1	1

X_Π

1	0	0	0	1	0	1	1
1	1	1	0	0	0	0	0
0	1	0	0	0	1	1	1
1	1	1	1	1	1	0	0
0	0	1	0	0	1	0	1
1	0	1	0	1	0	1	0

Z

1	0	0	1	0	1	0	1
1	1	1	1	0	1	1	1
0	1	0	0	1	0	0	0
1	1	1	0	1	0	0	1
0	0	1	0	0	0	1	0
1	0	1	1	0	0	1	1



- The same process is applied to each block.

Using Parity Checks

X

1	1	0	1	0	1	0	1
0	0	1	0	0	0	1	1
1	0	1	1	1	1	0	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	0
0	0	0	1	0	1	1	1

X_Π

1	0	0	0	1	0	1	1
1	1	1	0	0	0	0	0
0	1	0	0	0	1	1	1
1	1	1	1	1	1	0	0
0	0	1	0	0	1	0	1
1	0	1	0	1	0	1	0

Z

1	0	0	1	0	1	0	1
1	1	1	1	0	1	1	1
0	0	0	0	1	0	0	0
1	1	1	0	1	1	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0

- We receive noisy versions of X and Z,
  - ▷ we want to recover Π.

Using Parity Checks

X

1	0	0	1	0	1	0	1
0	0	1	0	0	0	1	1
1	0	1	0	1	1	0	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	1	0
0	0	0	1	0	1	1	1

X_Π

1	0	0	0	1	0	1	1
1	1	1	0	0	0	0	0
0	1	0	0	0	1	1	1
1	1	1	1	1	1	0	0
0	0	1	0	0	1	0	1
1	0	1	0	1	0	1	0

0	0	0	1	0	1	1	0

Z

1	0	0	1	0	1	0	1
1	1	1	1	0	1	1	1
0	1	0	0	1	0	0	0
1	1	1	0	1	0	0	1
0	0	1	0	0	0	1	0
1	0	1	1	0	0	1	1

0	0	0	0	0	1	1	0

parity check

- X_Π and Z are linked by parity check equations.

Using Parity Checks

X

1	0	0	1	0	1	0	1
0	0	1	0	0	0	1	1
1	0	1	0	1	1	0	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	1	0
0	0	0	1	0	1	1	1

0	1	0	0	1	1	0	0

permuted parity check

X_Π

1	0	0	0	1	0	1	1
1	1	1	0	0	0	0	0
0	1	0	0	0	1	1	1
1	1	1	1	1	1	0	0
0	0	1	0	0	1	0	1
1	0	1	0	1	0	1	0

0	0	0	1	0	1	1	0

Z

1	0	0	1	0	1	0	1
1	1	1	1	0	1	1	1
0	1	0	0	1	0	0	0
1	1	1	0	1	0	0	1
0	0	1	0	0	0	1	0
1	0	1	1	0	0	1	1

0	0	0	0	0	1	1	0

- X_Π and Z are linked by parity check equations,
  - X and Z by permuted parity checks.

Using Parity Checks

X	X_Π	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_Π and Z are linked by parity check equations,
  - any shift is also valid.

Using Parity Checks

X

1	1	0	1	0	1	0	1
0	0	1	0	0	0	1	1
1	0	1	1	1	1	0	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	0
0	0	0	1	0	1	1	1

X_Π

1	0	0	0	1	0	1	1
1	1	1	0	0	0	0	0
0	1	0	0	0	1	1	1
1	1	1	1	1	1	0	0
0	0	1	0	0	1	0	1
1	0	1	0	1	0	1	0

Z

1	0	0	1	0	1	0	1
1	1	1	1	0	1	1	1
0	0	0	0	1	0	0	0
1	1	1	0	1	1	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0

0	1	0	1	0	1	1	0	$1+D^2+D^3+D^4$
0	0	1	0	0	1	1	0	$1+D^4+D^5$

$1+D^2$	0	0	0	0	1	0	1	0
$1+D^3$	0	0	0	1	0	0	1	0

- Each parity check we find gives us information
  - on P and Q and on Π.

- Each parity check found is of the form λP on the X_{Π} part and λQ on the Z part
  - one knows λQ and the weight of λP
  - it is possible to classify the P, Q pairs depending on their parity checks.
- Once P/Q is known, one knows λP too and gets even more information on Π.
- For low noise levels this technique is very efficient.
  - For higher noise levels, only some parity check equations are found, leaving parts of Π unknown.

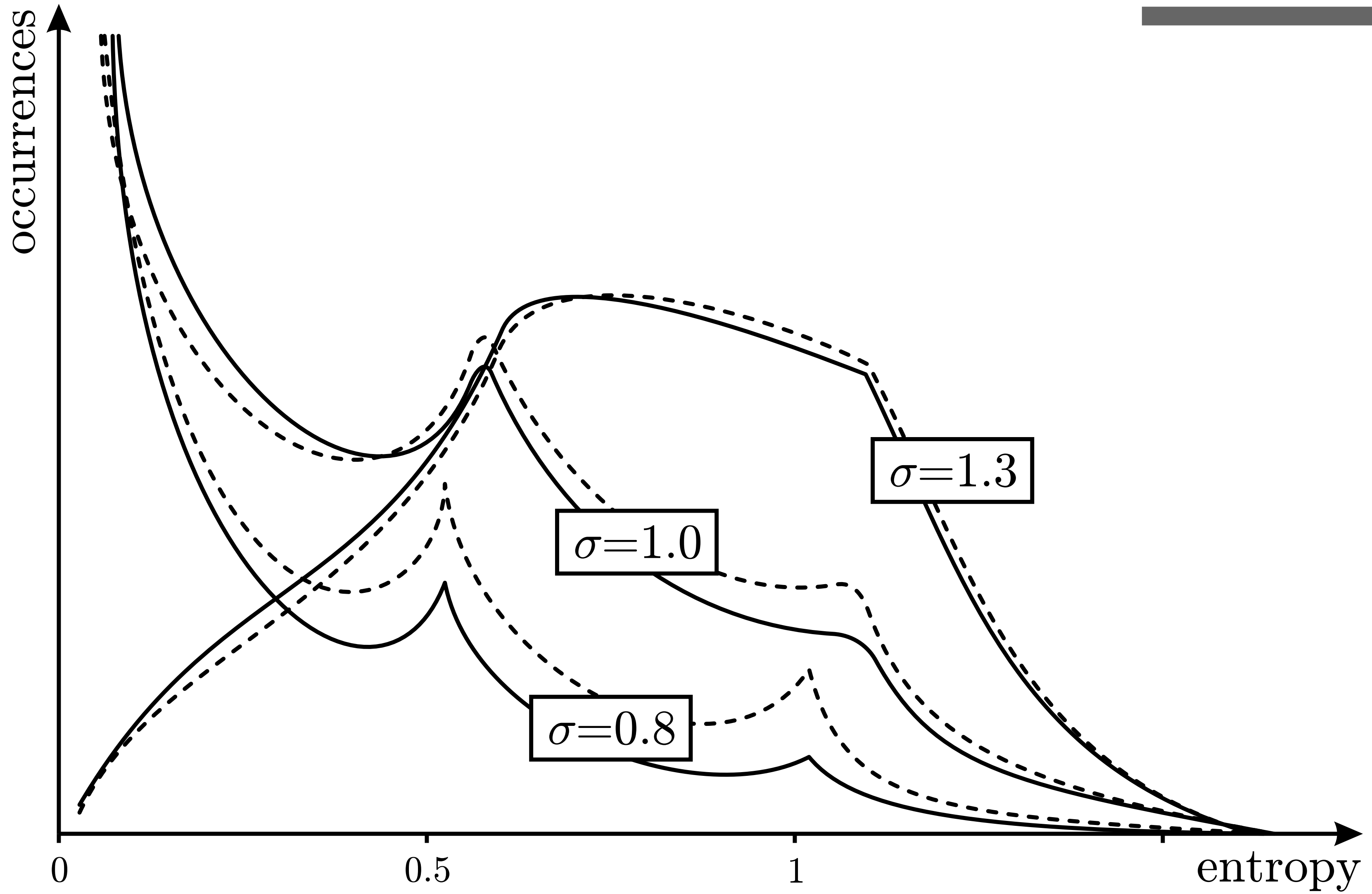
Using a Convolutional Decoder

Using a Convolutional Decoder

- For this technique, P/Q has to be known or guessed.
- One wants to find the first position x of Π: $\Pi(x) = 1$
  - ▷ there are N possibilities,
  - ▷ for each of the M intercepted blocks, one knows the first output bit of the convolutional encoder P/Q
    $\longrightarrow$ 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.

Using a Convolutional Decoder

- 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 trough sampling if the noise level is known.



- For a Gaussian noise of standard deviation σ quite high the "target" distributions can still be distinguished

- We use a straightforward algorithm:
  - ▷ the positions of Π 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 Π should remain.

N	σ	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	2 800	(2 736)	173 s
512	1.1	3 840	(3 837)	357 s
512	1.3	29 500	(29 448)	4 477 s
10 000	0.43	300	(163)	8 173 s
10 000	0.6	250	(249)	7 043 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.