Design and Decoding of Polar Codes with Large Kernels

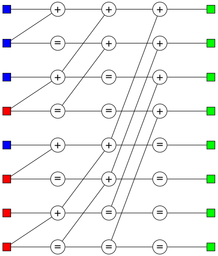
Peter Trifonov, Grigorii Trofimiuk
{pvtrifonov,gtrofimiuk}@itmo.ru

ITMO University

October 17, 2021

1. Motivation
   - Arikan polar codes and their limitations
   - What is possible with large kernels?
2. Decoding polar codes with large kernels
   - Successive cancellation decoding
   - Kernel processing (marginalization)
   - Trellis representation of linear codes
   - Recursive trellis processing
   - Window processing
3. Design of polar codes
   - Finding good polarization kernels
   - Code design for the BEC
   - Code design for the AWGN channel
   - Codes with Improved Distance Properties
4. Conclusions

Polar Codes



$$G=\begin{pmatrix}1&1&1&1&0&0&0&0\\1&1&0&0&1&1&0&0\\1&0&1&0&1&0&1&0\\1&1&1&1&1&1&1&1\end{pmatrix}$$

- Let $A_m = K_l^{\otimes m}$, where K_l is an $l \times l$ matrix
- Encoding: $c_0^{n-1} = u_0^{n-1} A_m$, $u_i = 0, i \in \mathcal{F}$, where $\mathcal{F}$ is the set of indices of low-capacity bit subchannels
  - A $(n = l^m, l^m - |\mathcal{F}|)$ linear code
- The successive cancellation decoding algorithm

$$\widehat{u}_i = \begin{cases} 0 & i \in \mathcal{F} \\ \arg\max_{u_i} W_m^{(i)}(\widehat{u}_0^{i-1}, u_i | y_0^{n-1}) & i \notin \mathcal{F} \end{cases}, i = 0, 1, \dots, l^m - 1$$

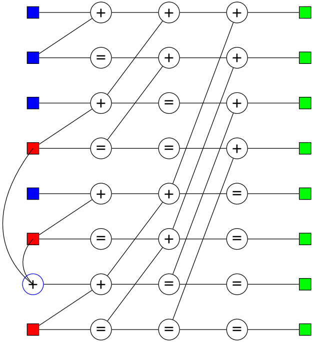
Arikan kernel $K_2 = F = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$:

$$\begin{aligned} W_\lambda^{(2\psi)}(u_0^{2\psi}|y_0^{2^\lambda-1}) &= \sum_{u_{2\psi+1}=0}^{1} W_{\lambda-1}^{(\psi)}(u_{0,even}^{2\psi+1} + u_{0,odd}^{2\psi+1}|y_{0,even}^{2^\lambda-1}) W_{\lambda-1}^{(\psi)}(u_{0,odd}^{2\psi+1}|y_{0,odd}^{2^\lambda-1}) \\ W_\lambda^{(2\psi+1)}(u_0^{2\psi+1}|y_0^{2^\lambda-1}) &= W_{\lambda-1}^{(\psi)}(u_{0,even}^{2\psi+1} + u_{0,odd}^{2\psi+1}|y_{0,even}^{2^\lambda-1}) W_{\lambda-1}^{(\psi)}(u_{0,odd}^{2\psi+1}|y_{0,odd}^{2^\lambda-1}) \end{aligned}$$

Polar Codes: Weak and Powerful

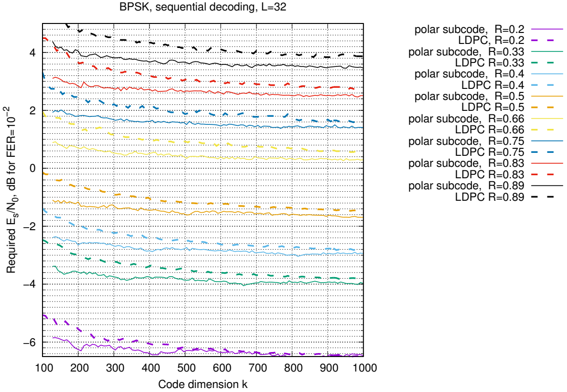
- SC decoding complexity is $O(n \log n)$
  - SC algorithm is highly suboptimal
  - List SC decoding is needed with complexity $O(Ln \log n)$
- Polar codes achieve the capacity
  - Error probability $O(2^{-\sqrt{n}})$, minimum distance $O(\sqrt{n})$
  - Polar codes with CRC, polar subcodes
- Highly regular encoder structure
  - High decoding latency
  - Poor hardware utilization

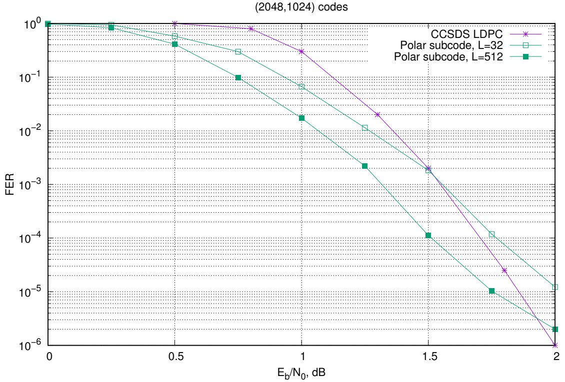
Improved Polar Codes



- Polar codes with CRC
  - CRC is used to select codewords from the list obtained by the Tal-Vardy decoder
- Polar subcodes
  - Dynamic frozen symbols $u_{i_j} = \sum_{s=0}^{i_j-1} u_s V_{js}, i_j \in \mathcal{F}$
  - V is the constraint matrix
- Polarization adjusted convolutional codes

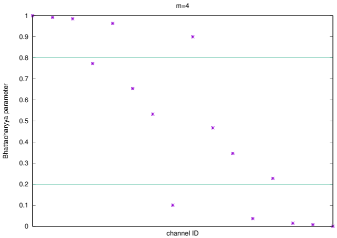
Performance issues



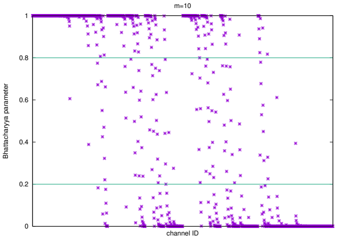


- The gain of polar (sub)codes with respect to LDPC diminishes with code length
- The slope of the FER curve is inferior to LDPC

Mediocre Subchannels



6 (37.5%) mediocre subchannels



94 (9.2%) mediocre subchannels

- The fraction of mediocre subchannels decreases with code length
- The number of mediocre subchannels increases with code length
- List size needed to cope with errors in mediocre subchannels grows exponentially with their number

How to mitigate exponential growth of the decoding complexity needed for near-ML decoding of polar (sub)codes?

Bit Subchannels

- Consider a binary input memoryless channel with transition probability function $W(y|c)$
- Transition probability functions for synthetic bit subchannels

$$W_m^{(i)}(y_0^{n-1}, u_0^{i-1}|u_i) = \frac{1}{2^{n-1}} \sum_{u_{i+1}^{n-1}} \prod_{j=0}^{n-1} W(y_j|(u_0^{n-1} K_l^{\otimes m})_j)$$

- With $m \to \infty$, the capacities of subchannels $W_m^{(i)}$ converge to 0 and 1, and the fraction of subchannels with capacity close to 1 converges to the capacity of W

The Bhattacharyya Parameter

- Consider a binary input channel with transition probability function $W(y|c)$
- An upper bound on BER for a maximum likelihood receiver is the Bhattacharyya parameter

$$Z(W) = \sum_y \sqrt{W(y|0)W(y|1)}$$

- Symmetric capacity $I(W)$ and the Bhattacharyya parameter satisfy

$$\begin{aligned} I(W)^2 + Z(W)^2 &\leq 1 \\ I(W) + Z(W) &\geq 1 \end{aligned}$$

- With $m \to \infty$, the Bhattacharyya parameters of $W_m^{(i)}$ converge to 0 and 1

Rate of Polarization

How good are subchannels $W_m^{(i)}$ obtained by the polarization process?

Definition

Matrix K_l has rate of polarization $E(K_l)$, if for any binary input channel $W: 0 \leq I(W) < 1$:

- For any $\beta < E(K_l)$
$$\liminf_{m\to\infty} P\left\{Z(W_m^{(i)}) \leq 2^{-l^{m\beta}}\right\} = I(W)$$
- For any $\beta > E(K_l)$
$$\liminf_{m\to\infty} P\left\{Z(W_m^{(i)}) \geq 2^{-l^{m\beta}}\right\} = 1$$

SC decoding error probability for an $(n = l^m, k)$ polar code based on K_l

$$P \leq 2^{-l^{\beta}}, \beta < E(K_l)$$

Arikan kernel: $E(K_2) = 0.5$

Partial Distances

Definition

Partial distances $\mathcal{D}_i, 0 \leq i < l$, of an $l \times l$ matrix $K = \begin{pmatrix} K[0] \\ \vdots \\ K[l-1] \end{pmatrix}$ are defined as

$$\mathcal{D}_i = d_H(K[i], \langle K[i+1], \ldots, K[l-1] \rangle), 0 \leq i < l-1,$$

$$\mathcal{D}_{l-1} = \mathrm{wt}(K[l-1]), 0 \leq i < l-1,$$

where $\langle a_1, \ldots, a_k \rangle$ is an (l, k) linear code generated by vectors $a_1, \ldots, a_k$, and $d_H(a, C)$ is the minimum Hamming distance between vector a and codewords of C.

Rate of polarization of kernel K is given by $E(K) = \frac{1}{l} \sum_{i=0}^{l-1} \log_l \mathcal{D}_i$

Scaling Exponent

- Scaling exponent μ for a family of codes with rate R shows the length $n = O(\frac{1}{(I(W)-R)^\mu})$ needed to achieve some fixed target FER on channel W
- Random codes: $\mu = 2$
- Scaling assumption for polar codes: for any ϵ there exists

$$f = \lim_{m\to\infty} \beta_m l^{\frac{m}{\mu(W,K_l)}-m}, 0 < f < \infty,$$

  where β_m is the number of subchannels $W_m^{(i)} : \epsilon \leq Z(W_m^{(i)}) \leq 1-\epsilon$
- Arikan polar codes on BEC: $\mu(BEC, K_2) = 3.627$

Some Asymptotic Results

- There exist $l \times l$ kernels K_l with rate of polarization[1] $E(K_l) \xrightarrow{l\to\infty} 1$
- There exist kernels with scaling exponent[2] $\mu(BEC, K_l) \xrightarrow{l\to\infty} 2$
- For any $E < 1$ and $\mu > 2$ there exist[3] polar codes with sufficiently large kernels K_l of rate $R = I(W) - \delta$ and length $O(\frac{1}{\delta^\mu})$ that enable reliable communication on a binary-input memoryless symmetric channel W with quasi-linear time encoding and decoding
- For any discrete memoryless channel with capacity $I(W)$, any $E, \mu > 0 : E + 2/\mu < 1$ there exist polar codes[4] with block error rate e^{-n^E}, code rate $R = I(W) - N^{-1/\mu}$, and decoding complexity $O(n \log n)$

[1] S. B. Korada, E. Sasoglu, and R. Urbanke, "Polar codes: Characterization of exponent, bounds, and constructions," IEEE Transactions on Information Theory, vol. 56, no. 12, December 2010

[2] A. Fazeli, H. Hassani, M. Mondelli and A. Vardy, "Binary Linear Codes with Optimal Scaling: Polar Codes with Large Kernels," IEEE Transactions on Information Theory, 87(9), September 2021

[3] V. Guruswami, A. Riazanov M. Ye. Arikan meets Shannon: Polar codes with near-optimal convergence to channel capacity, In proc. of 52nd Annual ACM Symposium on Theory of Computing, 2020.

[4] H.P. Wang and I. M. Duursma, "Polar Codes' Simplicity, Random Codes' Durability," IEEE Transactions on Information Theory, 67(3), 2021

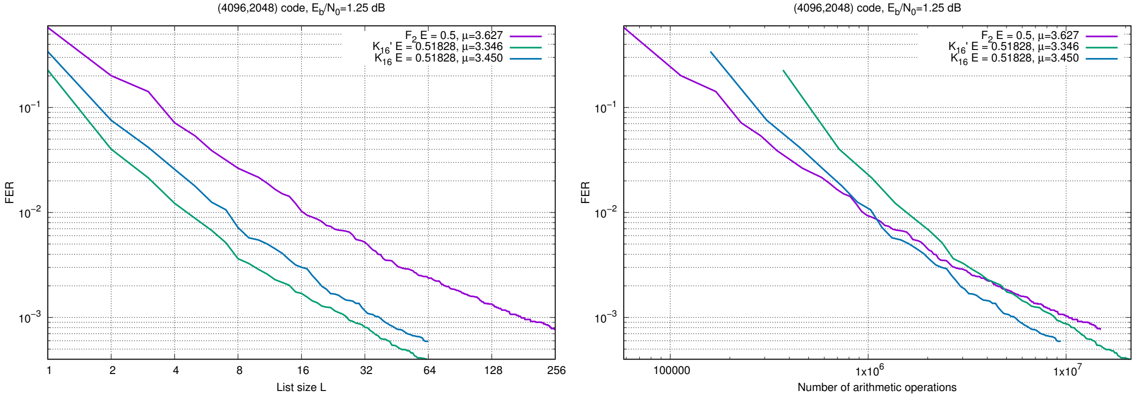
Some Constructions

- Spatially coupled LDPC codes were heuristically shown to have scaling exponent[5] $\mu_{SC-LDPC} \approx 3$
- There exists[6] a kernel of size $l = 64$ with scaling exponent $\mu \approx 2.87$. Already better than SC-LDPC!

[5] M. Mondelli, S. H. Hassani, and R. L. Urbanke, "How to achieve the capacity of asymmetric channels," IEEE Trans. Inf. Theory, vol. 64, no. 5, May 2018

[6] H. Yao, A. Fazeli, A. Vardy. Explicit Polar Codes with Small Scaling Exponent. In Proc. of ISIT 2019

Some Simulation Results: (4096, 2048) Code, SCL Decoding



- Large kernel based codes require smaller list size to achieve the same performance as the codes based on Arikan kernel
- Lower decoding complexity compared to codes based on Arikan kernel to achieve target FER

The Successive Cancellation Decoding Algorithm

- For $i=0,1,\dots,l^m-1$: $\widehat{u}_i=\begin{cases}0 & i\in\mathcal{F}\\ \arg\max_{u_i} W_m^{(i)}(y_0^{n-1},\widehat{u}_0^{i-1}|u_i) & i\notin\mathcal{F}\end{cases}$
- Bit subchannels with transition probability function $W_m^{(i)}(y_0^{n-1},u_0^{i-1}|u_i)$
- It is convenient to use probabilities

$$W_m^{(i)}(u_0^i|y_0^{n-1})=\frac{W_m^{(i)}(y_0^{n-1},\widehat{u}_0^{i-1}|u_i)}{2W(y_0^{n-1})}=\sum_{u_{i+1}^{n-1}}\prod_{j=0}^{n-1}W((u_0^{n-1}K_l^{\otimes m})_j|y_j)$$

- Complexity $O(n\log_l n)$ operations of kernel processing, i.e. computing

$$W_m^{(li+s)}(u_0^{li+s}|y_0^{n-1})=\sum_{u_{li+s+1}^{li+l-1}}\prod_{j=0}^{l-1}W_{m-1}^{(i)}\left((u_{lt}^{l(t+1)-1}K_l)_j,0\le t\le i|y_{j,l}^{n-1}\right),\qquad(1)$$

where $r_{j,l}^{n-1}=(r_j,r_{j+l},\dots,r_{j+n-l}),0\le s<l,0\le i<\frac{n}{l},W_0^{(0)}(c|y)=W(c|y)$

The Successive Cancellation Decoding Algorithm

- For $i = 0, 1, \ldots, l^m - 1$: $\widehat{u}_i = \begin{cases} 0 & i \in \mathcal{F} \\ \arg\max_{u_i} W_m^{(i)}(\widehat{u}_0^{i-1}, u_i | y_0^{n-1}) & i \notin \mathcal{F} \end{cases}$
- Bit subchannels with transition probability function $W_m^{(i)}(y_0^{n-1}, u_0^{i-1} | u_i)$
- It is convenient to use probabilities

$$W_m^{(i)}(u_0^i | y_0^{n-1}) = \frac{W_m^{(i)}(y_0^{n-1}, \widehat{u}_0^{i-1} | u_i)}{2W(y_0^{n-1})} = \sum_{u_{i+1}^{n-1}} \prod_{j=0}^{n-1} W((u_0^{n-1} K_l^{\otimes m})_j | y_j)$$

- Complexity $O(n \log_l n)$ operations of kernel processing, i.e. computing

$$W_m^{(li+s)}(u_0^{li+s} | y_0^{n-1}) = \sum_{u_{li+s+1}^{li+l-1}} \prod_{j=0}^{l-1} W_{m-1}^{(i)} \left((u_{lt}^{l(t+1)-1} K_l)_j, 0 \le t \le i | y_{j,l}^{n-1} \right), \quad (1)$$

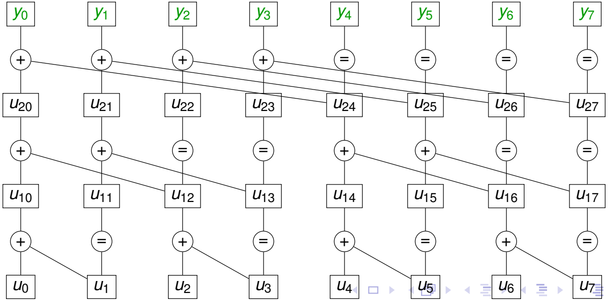
where $r_{j,l}^{n-1} = (r_j, r_{j+l}, \ldots, r_{j+n-l}), 0 \le s < l, 0 \le i < \frac{n}{l}, W_0^{(0)}(c|y) = W(c|y)$

Example: Arikan kernel $F_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$

$$W_\mu^{(2i)}(u_0^{2i}|y_0^{n-1}) = \sum_{u_{2i+1}} W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i|y_{0,2}^{n-1}) W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i|y_{1,2}^{n-1})$$

$$W_\mu^{(2i+1)}(u_0^{2i+1}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i|y_{0,2}^{n-1}) W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i|y_{1,2}^{n-1})$$

- The same co-factors $W_{\mu-1}^{(i)}$ are used at phases $2i$ and $2i+1$. They can be reused
- Complexity $O(n \log_2 n)$
- Memory size $O(n)$

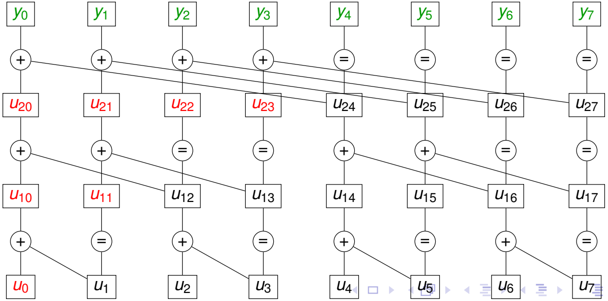


Example: Arikan kernel $F_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$

$$W_\mu^{(2i)}(u_0^{2i}|y_0^{n-1}) = \sum_{u_{2i+1}} W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0\le t\le i|y_{0,2}^{n-1})W_{\mu-1}^{(i)}(u_{2t+1}, 0\le t\le i|y_{1,2}^{n-1})$$

$$W_\mu^{(2i+1)}(u_0^{2i+1}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0\le t\le i|y_{0,2}^{n-1})W_{\mu-1}^{(i)}(u_{2t+1}, 0\le t\le i|y_{1,2}^{n-1})$$

- The same co-factors $W_{\mu-1}^{(i)}$ are used at phases $2i$ and $2i+1$. They can be reused
- Complexity $O(n\log_2 n)$
- Memory size $O(n)$

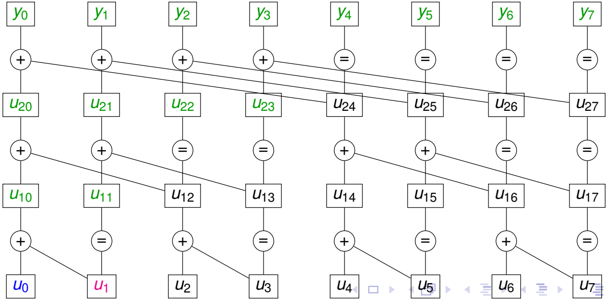


Example: Arikan kernel $F_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$

$$W_{\mu}^{(2i)}(u_0^{2i}|y_0^{n-1}) = \sum_{u_{2i+1}} W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i|y_{0,2}^{n-1}) W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i|y_{1,2}^{n-1})$$

$$W_{\mu}^{(2i+1)}(u_0^{2i+1}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i|y_{0,2}^{n-1}) W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i|y_{1,2}^{n-1})$$

- The same co-factors $W_{\mu-1}^{(i)}$ are used at phases $2i$ and $2i+1$. They can be reused
- Complexity $O(n \log_2 n)$
- Memory size $O(n)$

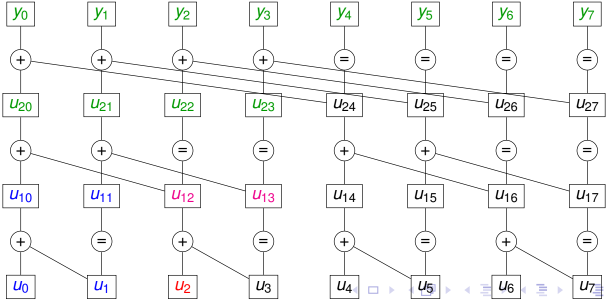


Example: Arikan kernel $F_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$

$$W_\mu^{(2i)}(u_0^{2i}|y_0^{n-1}) = \sum_{u_{2i+1}} W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i|y_{0,2}^{n-1})\, W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i|y_{1,2}^{n-1})$$

$$W_\mu^{(2i+1)}(u_0^{2i+1}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i|y_{0,2}^{n-1})\, W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i|y_{1,2}^{n-1})$$

- The same co-factors $W_{\mu-1}^{(i)}$ are used at phases $2i$ and $2i+1$. They can be reused
- Complexity $O(n \log_2 n)$
- Memory size $O(n)$

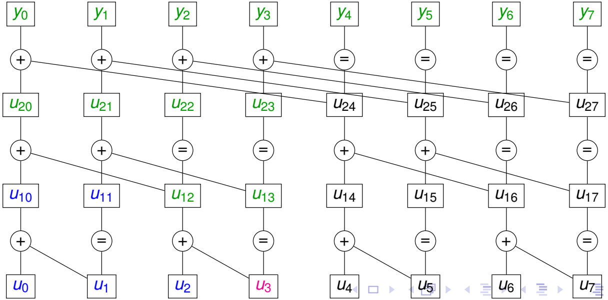


Example: Arikan kernel $F_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$

$$W_{\mu}^{(2i)}(u_0^{2i}|y_0^{n-1}) = \sum_{u_{2i+1}} W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i|y_{0,2}^{n-1})W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i|y_{1,2}^{n-1})$$

$$W_{\mu}^{(2i+1)}(u_0^{2i+1}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i|y_{0,2}^{n-1})W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i|y_{1,2}^{n-1})$$

- The same co-factors $W_{\mu-1}^{(i)}$ are used at phases $2i$ and $2i+1$. They can be reused
- Complexity $O(n \log_2 n)$
- Memory size $O(n)$

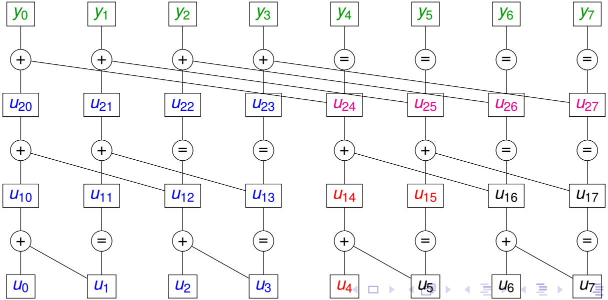


Example: Arikan kernel $F_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$

$$W_\mu^{(2i)}(u_0^{2i}|y_0^{n-1}) = \sum_{u_{2i+1}} W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i|y_{0,2}^{n-1}) W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i|y_{1,2}^{n-1})$$

$$W_\mu^{(2i+1)}(u_0^{2i+1}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i|y_{0,2}^{n-1}) W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i|y_{1,2}^{n-1})$$

- The same co-factors $W_{\mu-1}^{(i)}$ are used at phases $2i$ and $2i+1$. They can be reused
- Complexity $O(n \log_2 n)$
- Memory size $O(n)$

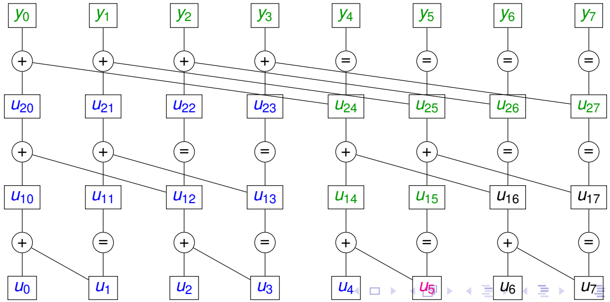


Example: Arikan kernel $F_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$

$$W_\mu^{(2i)}(u_0^{2i}|y_0^{n-1}) = \sum_{u_{2i+1}} W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0\le t\le i|y_{0,2}^{n-1})W_{\mu-1}^{(i)}(u_{2t+1}, 0\le t\le i|y_{1,2}^{n-1})$$

$$W_\mu^{(2i+1)}(u_0^{2i+1}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0\le t\le i|y_{0,2}^{n-1})W_{\mu-1}^{(i)}(u_{2t+1}, 0\le t\le i|y_{1,2}^{n-1})$$

- The same co-factors $W_{\mu-1}^{(i)}$ are used at phases $2i$ and $2i+1$. They can be reused
- Complexity $O(n\log_2 n)$
- Memory size $O(n)$

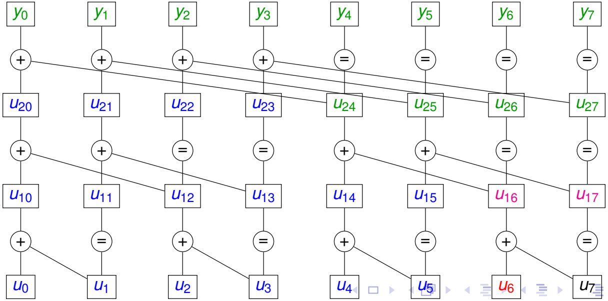


Example: Arikan kernel $F_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$

$$W_\mu^{(2i)}(u_0^{2i}|y_0^{n-1}) = \sum_{u_{2i+1}} W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i|y_{0,2}^{n-1}) W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i|y_{1,2}^{n-1})$$

$$W_\mu^{(2i+1)}(u_0^{2i+1}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i|y_{0,2}^{n-1}) W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i|y_{1,2}^{n-1})$$

- The same co-factors $W_{\mu-1}^{(i)}$ are used at phases $2i$ and $2i+1$. They can be reused
- Complexity $O(n \log_2 n)$
- Memory size $O(n)$

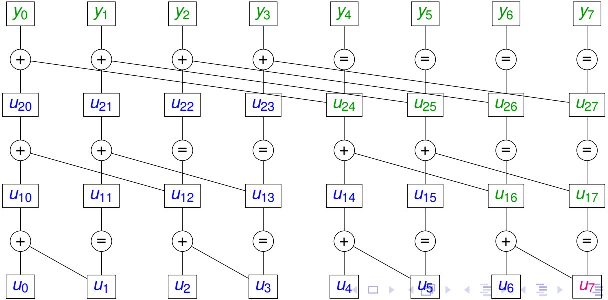


Example: Arikan kernel $F_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$

$$W_\mu^{(2i)}(u_0^{2i}|y_0^{n-1}) = \sum_{u_{2i+1}} W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i|y_{0,2}^{n-1}) W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i|y_{1,2}^{n-1})$$

$$W_\mu^{(2i+1)}(u_0^{2i+1}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i|y_{0,2}^{n-1}) W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i|y_{1,2}^{n-1})$$

- The same co-factors $W_{\mu-1}^{(i)}$ are used at phases $2i$ and $2i+1$. They can be reused
- Complexity $O(n \log_2 n)$
- Memory size $O(n)$

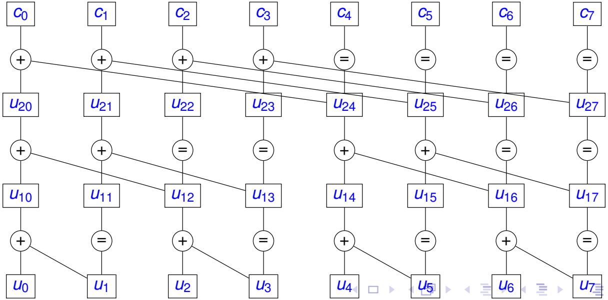


Example: Arikan kernel $F_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$

$$W_\mu^{(2i)}(u_0^{2i}|y_0^{n-1}) = \sum_{u_{2i+1}} W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i | y_{0,2}^{n-1}) W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i | y_{1,2}^{n-1})$$

$$W_\mu^{(2i+1)}(u_0^{2i+1}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{2t}+u_{2t+1}, 0 \le t \le i | y_{0,2}^{n-1}) W_{\mu-1}^{(i)}(u_{2t+1}, 0 \le t \le i | y_{1,2}^{n-1})$$

- The same co-factors $W_{\mu-1}^{(i)}$ are used at phases $2i$ and $2i+1$. They can be reused
- Complexity $O(n \log_2 n)$
- Memory size $O(n)$

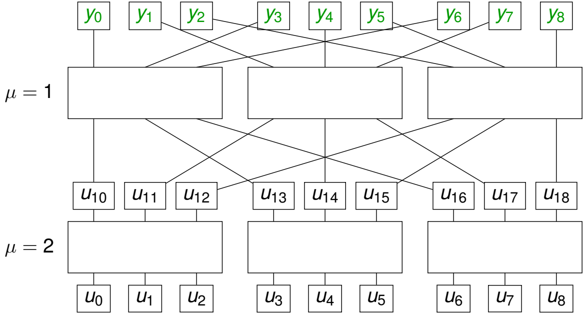


Example: $K_3 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$

$$W_{\mu}^{(3i)}(u_0^{3i}|y_0^{n-1}) = \sum_{u_{2i+1},u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_{\mu}^{(3i+1)}(u_0^{3i+1}|y_0^{n-1}) = \sum_{u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_{\mu}^{(3i+2)}(u_0^{3i+2}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$



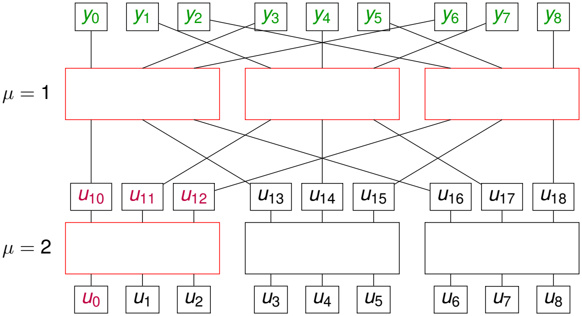
- At layer $\mu \ge 1$ calculations are performed in batches of size $l^{m-\mu}$
- Reuse of probabilities and processor state
- Partial sums are supplied to the processors

Example: $K_3 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$

$$W_{\mu}^{(3i)}(u_0^{3i}|y_0^{n-1}) = \sum_{u_{2i+1},u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0\le t\le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0\le t\le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0\le t\le i|y_{2,3}^{n-1})$$

$$W_{\mu}^{(3i+1)}(u_0^{3i+1}|y_0^{n-1}) = \sum_{u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0\le t\le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0\le t\le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0\le t\le i|y_{2,3}^{n-1})$$

$$W_{\mu}^{(3i+2)}(u_0^{3i+2}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0\le t\le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0\le t\le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0\le t\le i|y_{2,3}^{n-1})$$



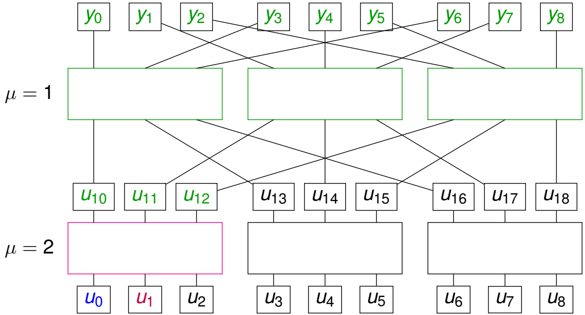
- At layer $\mu \ge 1$ calculations are performed in batches of size $l^{m-\mu}$
- Reuse of probabilities and processor state
- Partial sums are supplied to the processors

Example: $K_3 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$

$$W_{\mu}^{(3i)}(u_0^{3i}|y_0^{n-1}) = \sum_{u_{2i+1},u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0\le t\le i|y_{0,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+1}, 0\le t\le i|y_{1,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+2}, 0\le t\le i|y_{2,3}^{n-1})$$

$$W_{\mu}^{(3i+1)}(u_0^{3i+1}|y_0^{n-1}) = \sum_{u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0\le t\le i|y_{0,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+1}, 0\le t\le i|y_{1,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+2}, 0\le t\le i|y_{2,3}^{n-1})$$

$$W_{\mu}^{(3i+2)}(u_0^{3i+2}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0\le t\le i|y_{0,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+1}, 0\le t\le i|y_{1,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+2}, 0\le t\le i|y_{2,3}^{n-1})$$



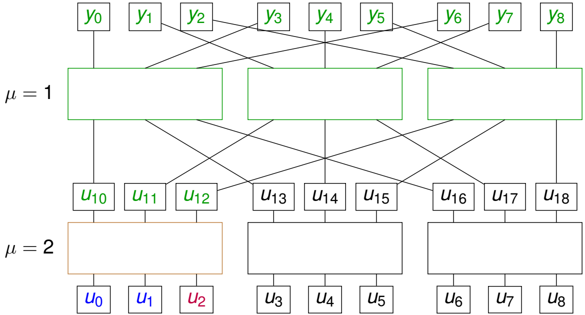
- At layer $\mu \geq 1$ calculations are performed in batches of size $l^{m-\mu}$
- Reuse of probabilities and processor state
- Partial sums are supplied to the processors

Example: $K_3 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$

$$W_{\mu}^{(3i)}(u_0^{3i}|y_0^{n-1}) = \sum_{u_{2i+1},u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_{\mu}^{(3i+1)}(u_0^{3i+1}|y_0^{n-1}) = \sum_{u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_{\mu}^{(3i+2)}(u_0^{3i+2}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$



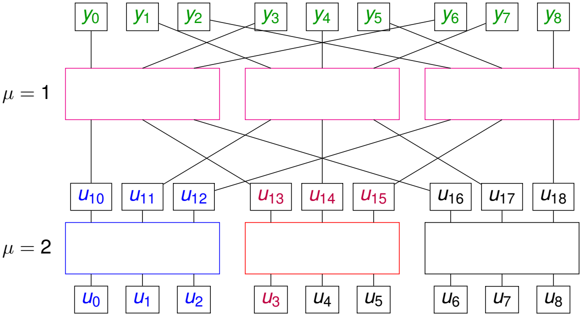
- At layer $\mu \ge 1$ calculations are performed in batches of size $l^{m-\mu}$
- Reuse of probabilities and processor state
- Partial sums are supplied to the processors

Example: $K_3 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$

$$W_\mu^{(3i)}(u_0^{3i}|y_0^{n-1}) = \sum_{u_{2i+1},u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_\mu^{(3i+1)}(u_0^{3i+1}|y_0^{n-1}) = \sum_{u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_\mu^{(3i+2)}(u_0^{3i+2}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$



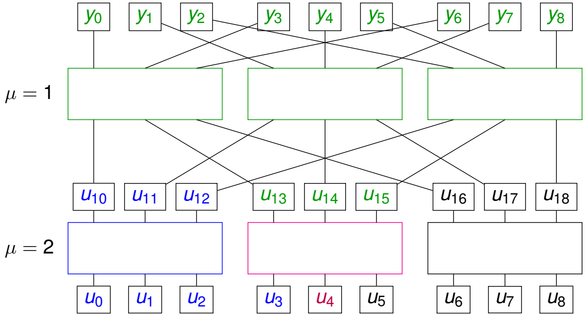
- At layer $\mu \geq 1$ calculations are performed in batches of size $l^{m-\mu}$
- Reuse of probabilities and processor state
- Partial sums are supplied to the processors

Example: $K_3 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$

$$W_{\mu}^{(3i)}(u_0^{3i}|y_0^{n-1}) = \sum_{u_{2i+1},u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_{\mu}^{(3i+1)}(u_0^{3i+1}|y_0^{n-1}) = \sum_{u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_{\mu}^{(3i+2)}(u_0^{3i+2}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$



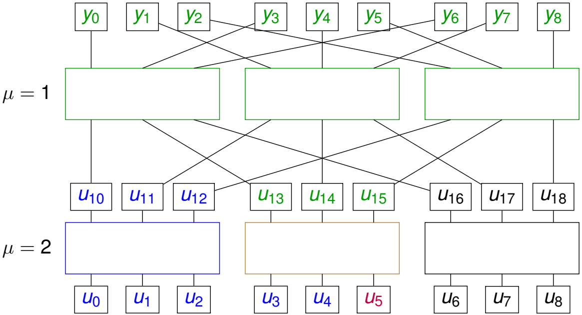
- At layer $\mu \ge 1$ calculations are performed in batches of size $l^{m-\mu}$
- Reuse of probabilities and processor state
- Partial sums are supplied to the processors

Example: $K_3 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$

$$W_\mu^{(3i)}(u_0^{3i}|y_0^{n-1}) = \sum_{u_{2i+1},u_{2i+2}} W^{(i)}_{\mu-1}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W^{(i)}_{\mu-1}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W^{(i)}_{\mu-1}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_\mu^{(3i+1)}(u_0^{3i+1}|y_0^{n-1}) = \sum_{u_{2i+2}} W^{(i)}_{\mu-1}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W^{(i)}_{\mu-1}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W^{(i)}_{\mu-1}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_\mu^{(3i+2)}(u_0^{3i+2}|y_0^{n-1}) = W^{(i)}_{\mu-1}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W^{(i)}_{\mu-1}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W^{(i)}_{\mu-1}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$



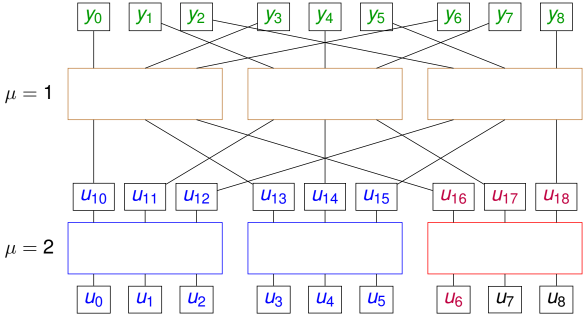
- At layer $\mu \ge 1$ calculations are performed in batches of size $l^{m-\mu}$
- Reuse of probabilities and processor state
- Partial sums are supplied to the processors

Example: $K_3 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$

$$W_\mu^{(3i)}(u_0^{3i}|y_0^{n-1}) = \sum_{u_{2i+1},u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_\mu^{(3i+1)}(u_0^{3i+1}|y_0^{n-1}) = \sum_{u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_\mu^{(3i+2)}(u_0^{3i+2}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$



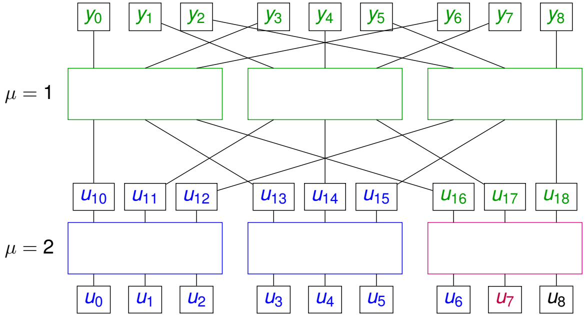
- At layer $\mu \ge 1$ calculations are performed in batches of size $l^{m-\mu}$
- Reuse of probabilities and processor state
- Partial sums are supplied to the processors

Example: $K_3 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$

$$W_{\mu}^{(3i)}(u_0^{3i}|y_0^{n-1}) = \sum_{u_{2i+1},u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_{\mu}^{(3i+1)}(u_0^{3i+1}|y_0^{n-1}) = \sum_{u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_{\mu}^{(3i+2)}(u_0^{3i+2}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1})W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$



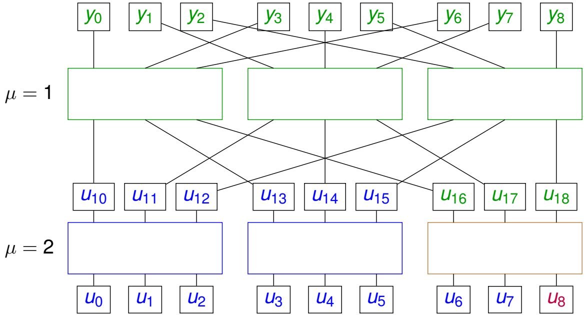
- At layer $\mu \ge 1$ calculations are performed in batches of size $l^{m-\mu}$
- Reuse of probabilities and processor state
- Partial sums are supplied to the processors

Example: $K_3 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$

$$W_\mu^{(3i)}(u_0^{3i}|y_0^{n-1}) = \sum_{u_{2i+1}, u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_\mu^{(3i+1)}(u_0^{3i+1}|y_0^{n-1}) = \sum_{u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_\mu^{(3i+2)}(u_0^{3i+2}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$



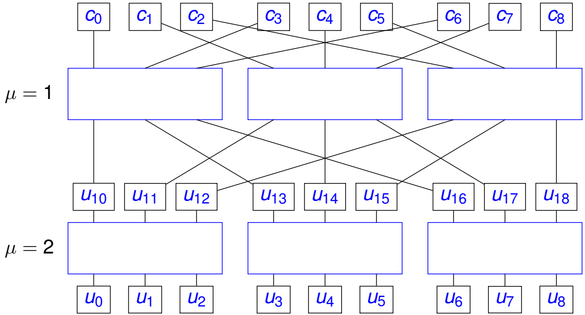
- At layer $\mu \geq 1$ calculations are performed in batches of size $l^{m-\mu}$
- Reuse of probabilities and processor state
- Partial sums are supplied to the processors

Example: $K_3 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$

$$W_\mu^{(3i)}(u_0^{3i}|y_0^{n-1}) = \sum_{u_{2i+1},u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_\mu^{(3i+1)}(u_0^{3i+1}|y_0^{n-1}) = \sum_{u_{2i+2}} W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$

$$W_\mu^{(3i+2)}(u_0^{3i+2}|y_0^{n-1}) = W_{\mu-1}^{(i)}(u_{3t}+u_{3t+1}+u_{3t+2}, 0 \le t \le i|y_{0,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+1}, 0 \le t \le i|y_{1,3}^{n-1}) W_{\mu-1}^{(i)}(u_{3t+2}, 0 \le t \le i|y_{2,3}^{n-1})$$



- At layer $\mu \ge 1$ calculations are performed in batches of size $l^{m-\mu}$
- Reuse of probabilities and processor state
- Partial sums are supplied to the processors

Improved Decoding Methods

- The successive cancellation decoding algorithm is highly suboptimal
- List SC decoding: an immediate extension of the Tal-Vardy algorithm[7]
  - Shared memory data structures for probabilities/LLRs, partial sums, *and processor state*
- Sequential/stack decoding[8]
  - Relies on Tal-Vardy data structures
  - The score function given in [9] provides substantial reduction of average complexity
  - Negligible performance degradation with respect to SCL decoding

[7] I. Tal and A. Vardy, "List decoding of polar codes," IEEE Transactions On Information Theory, vol. 61, no. 5, May 2015.

[8] V. Miloslavskaya and P. Trifonov, "Sequential decoding of polar codes with arbitrary binary kernel," in Proceedings of IEEE Information Theory Workshop. Hobart, Australia: IEEE, 2014

[9] P. Trifonov, "A score function for sequential decoding of polar codes," in Proceedings of IEEE ISIT, Vail, USA, 2018.

Kernel Processing: An Approximation

$$W_m^{(i)}(u_0^i|y_0^{n-1}) = \sum_{u_{i+1}^{n-1}} \prod_{j=0}^{n-1} W\left((u_0^{n-1}K_l^{\otimes m})_j|y_j\right) \approx \mathcal{W}_m^{(i)}(u_0^i|y_0^{n-1}) = \max_{u_{i+1}^{n-1}} \prod_{j=0}^{n-1} W\left((u_0^{n-1}K_l^{\otimes m})_j|y_j\right)$$

Performance loss is negligible

$$\mathcal{W}_m^{(li+s)}(u_0^{li+s}|y_0^{n-1}) = \max_{u_{li+s+1}^{li+l-1}} \prod_{j=0}^{l-1} \mathcal{W}_{m-1}^{(i)}\left((u_{lt}^{l(t+1)-1}K_l)_j, 0 \le t \le i|y_{j,l}^{n-1}\right)$$

$\mathcal{W}_m^{(i)}(u_0^i|y_0^{n-1})$ is the probability of the most likely continuation of vector u_0^i, not taking into account any freezing constraints on $u_j, j < i$

Kernel Processing Problem

- Assume $m=1$ for the sake of simplicity. Assume that $W(y|c)$ is a symmetric channel

$$\mathcal{W}_1^{(s)}(u_0^s|y_0^{l-1}) = \max_{u_{s+1}^{l-1}} \prod_{j=0}^{l-1} W((u_0^{l-1}K)_j|y_j) = \max_{u_{s+1}^{l-1}} \prod_{j=0}^{l-1} W\left((u_{s+1}^{l-1}K_{s+1..l-1})_j|y_j(-1)^{(u_0^s K_{0..s})_j}\right)$$

- $K_{a..b}$ is the submatrix of K given by rows $a,\ldots,b$
- If all u_i are equiprobable, computing $\mathcal{W}_1^{(s)}(u_0^s|y_0^{l-1})$ is equivalent to ML decoding of y_0^{l-1} in the coset given by $u_0^s K_{0..s}$ of the code generated by $K_{s+1..l-1}$

LLR-domain Kernel Processing

- The log-likelihood ratio $S_{\mu}^{(i)}(u_0^{i-1}, y_0^{l^{\mu}-1}) = \ln \frac{\mathcal{W}_{\mu}^{(i)}(u_0^{i-1},0|y_0^{l^{\mu}-1})}{\mathcal{W}_{\mu}^{(i)}(u_0^{i-1},1|y_0^{l^{\mu}-1})}, \mu \geq 0$
- Let $\hat{c}_i$ be the hard decision value corresponding to y_i

$$
\begin{aligned}
S_1^{(i)}(u_0^{i-1}, y_0^{l-1}) &= \log \frac{\max_{u_{s+1}^{l-1}} \prod_{j=0}^{l-1} W((u_0^{i-1}, 0, u_{i+1}^{l-1})K)_j | y_j)}{\max_{u_{s+1}^{l-1}} \prod_{j=0}^{l-1} W((u_0^{i-1}, 1, u_{i+1}^{l-1})K)_j | y_j)} \\
&= \max_{u_{s+1}^{l-1}} \left(\sum_{j=0}^{l-1} \left(\log W((u_0^{i-1}, 0, u_{i+1}^{l-1})K)_j | y_j) \right) - \log W(\hat{c}_j | y_j) \right) \\
&\quad - \max_{u_{s+1}^{l-1}} \left(\sum_{j=0}^{l-1} \left(\log W((u_0^{i-1}, 1, u_{i+1}^{l-1})K)_j | y_j) \right) - \log W(\hat{c}_j | y_j) \right) \\
&= \max_{u_{s+1}^{l-1}} M((u_0^{i-1}, 0, u_{i+1}^{l-1})K, S_0^{l-1}) - \max_{u_{s+1}^{l-1}} M((u_0^{i-1}, 1, u_{i+1}^{l-1})K, S_0^{l-1})
\end{aligned}
$$

LLR-domain Kernel Processing

- Hard decision $\hat{c}_j = \arg\max_{c\in\{0,1\}} W(c|y_j)$
- $\log W(c_j|y_j) - \log W(\hat{c}_j|y_j) = \begin{cases} 0, & \hat{c}_j = c_j \\ -\left|\log \frac{W(0|y_j)}{W(1|y_j)}\right|, & \hat{c}_j \neq c_j \end{cases}$
- Input LLRs $S_j = \log \frac{W(0|y_j)}{W(1|y_j)}$
- Correlation discrepancy[10]

$$M(c_0^{l-1}, S_0^{l-1}) = -\sum_{j:(-1)^{c_j} S_j < 0} |S_j|$$

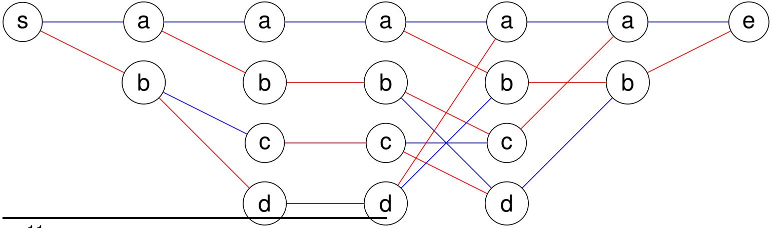
- Kernel input LLRs

$$S_1^{(i)}(u_0^{i-1}, y_0^{l-1}) = \max_{u_{s+1}^{l-1}} M((u_0^{i-1}, 0, u_{i+1}^{l-1})K, S_0^{l-1}) - \max_{u_{s+1}^{l-1}} M((u_0^{i-1}, 1, u_{i+1}^{l-1})K, S_0^{l-1})$$

[10] Sometimes it is defined without the $-$ sign

Trellis Representation of Linear Codes

- Any binary linear code of length l can be represented by a minimal trellis[11]
- Codewods correspond to distinct paths in a trellis from the start to the end nodes
- If two codewords y_0^{l-1}, z_0^{l-1} satisfy $y_0^i = z_0^i$, then they pass through the same nodes in the minimal trellis up to symbol i
- If two codewords y_0^{l-1}, z_0^{l-1} satisfy $a_i^{l-1} = b_i^{l-1}$, then they pass through the same nodes in the minimal trellis starting from symbol i
- Viterbi algorithm can be used to implement ML decoding

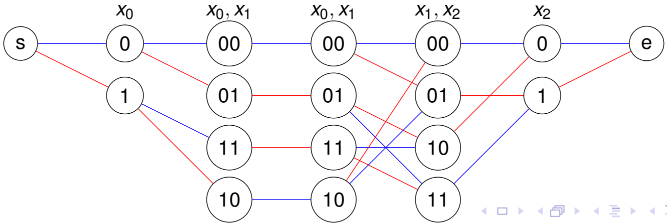


[11] A. Vardy, "Trellis structure of codes," in Handbook of Coding Theory, Elsevier Science, 1998

Minimum Span Form of the Generator Matrix

- A vector c_0^{l-1} starts in position $b = b(c_0^{l-1})$ if $c_b \neq 0, c_i = 0, 0 \leq i < b$
- A vector c_0^{l-1} ends in position $e = e(c_0^{l-1})$ if $c_e \neq 0, c_i = 0, e < i < l$
- One can transform the generator matrix of the code so that its rows start and end in distinct columns (minimum span form)
- Vector c_0^{l-1} is active from position $b(c_0^{l-1})$ till position $e(c_0^{l-1}) - 1$
- Trellis nodes at level i are labeled by the values of information symbols x_j corresponding to generator matrix rows active in position i
- Edges are labeled with $(xG)_i$

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$



Extended Kernel Codes

- t-th extended[12] kernel code $\overline{C_K}^{(t)}$ is generated by
$$\overline{K}^{(t)} = \begin{pmatrix} K[t] & 1 \\ K[t+1] & 0 \\ \vdots & \vdots \\ K[l-1] & 0 \end{pmatrix}$$
- Assume that y_l is erased. Given y_0^l, find most probable codewords $(c_0, \dots, c_{l-1}, u_t), u_t \in \mathbb{F}_2$ of code $\overline{C}_K^{(t)}$ generated by $\overline{K}^{(t)}$
- Arikan kernel $K = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$: $\overline{K}^{(0)} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$, $\overline{K}^{(1)} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$

[12] H. Griesser, V. R. Sidorenko, "A posteriory probability decoding of nonsystematically encoded block codes," Problems of Information Transmission, 38(3), 2002

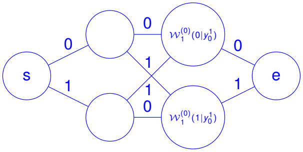
Extended Kernel Codes

- t-th extended[12] kernel code $\overline{C}_K{}^{(t)}$ is generated by

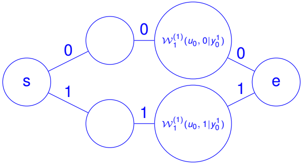
$$\overline{K}^{(t)} = \begin{pmatrix} K[t] & 1 \\ K[t+1] & 0 \\ \vdots & \vdots \\ K[l-1] & 0 \end{pmatrix}$$

- Assume that y_l is erased. Given y_0^l, find most probable codewords $(c_0, \dots, c_{l-1}, u_t)$, $u_t \in \mathbb{F}_2$ of code $\overline{C}_K^{(t)}$ generated by $\overline{K}^{(t)}$
- Arikan kernel $K = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$: $\overline{K}^{(0)} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$,

$$\overline{K}^{(1)} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$$



Extended trellis for $\overline{\mathcal{C}}_K^{(0)}$

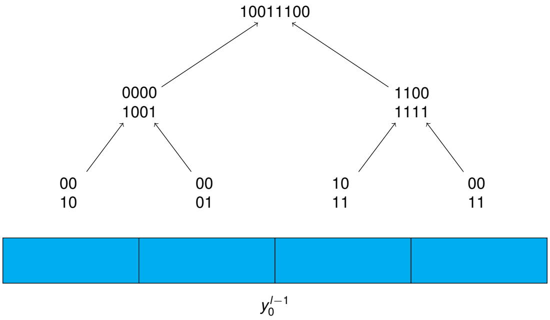


Extended trellis for $\overline{\mathcal{C}}_K^{(1)}$

[12] H. Griesser, V. R. Sidorenko, "A posteriory probability decoding of nonsystematically encoded block codes," Problems of Information Transmission, 38(3), 2002

Recursive Maximum Likelihood Decoding of Linear Codes: the Idea

- Viterbi algorithm is not optimal in terms of complexity
- Partition the noisy vector y_0^{l-1} into a number of sections
- Find the most probable codeword subvectors for each section
- Combine short codeword subvectors into longer subvectors
- Do this recursively[13]



[13] T. Fujiwara, H. Yamamoto, T. Kasami, and S. Lin, "A trellis-based recursive maximum-likelihood decoding algorithm for binary linear block codes," IEEE Transactions On Information Theory, vol. 44, no. 2, March 1998.

Sectionalized Trellis of a Linear Block Code

- Given a linear code C, let $C_{a,b}$ be its subcode, such that all its codewords have non-zero symbols only in positions $a \leq i < b$
- Let $p_{a,b}(C)$ be a linear code obtained by puncturing all symbols, except those in positions $a \leq i < b$, from codewords of C
- Let $s_{a,b}(C) = p_{a,b}(C_{a,b})$, i.e. a code obtained from C by shortening it on all symbols except those with indices $a \leq i < b$.
- Trellis paths from time a to time b correspond to cosets in $p_{a,b}(C)/s_{a,b}(C)$. The same coset may occur several times in a trellis

$$C : G = \begin{pmatrix} 1&1&1&1&1&1&1&1 \\ 1&1&1&1&0&0&0&0 \\ 1&1&0&0&1&1&0&0 \\ 1&0&1&0&1&0&1&0 \end{pmatrix}$$

Sectionalized Trellis of a Linear Block Code

- Given a linear code C, let $C_{a,b}$ be its subcode, such that all its codewords have non-zero symbols only in positions $a \leq i < b$
- Let $p_{a,b}(C)$ be a linear code obtained by puncturing all symbols, except those in positions $a \leq i < b$, from codewords of C
- Let $s_{a,b}(C) = p_{a,b}(C_{a,b})$, i.e. a code obtained from C by shortening it on all symbols except those with indices $a \leq i < b$.
- Trellis paths from time a to time b correspond to cosets in $p_{a,b}(C)/s_{a,b}(C)$. The same coset may occur several times in a trellis

$$C : G = \begin{pmatrix} 1&1&1&1&1&1&1&1 \\ 1&1&1&1&0&0&0&0 \\ 1&1&0&0&1&1&0&0 \\ 1&0&1&0&1&0&1&0 \end{pmatrix}$$

$$p_{0,4}(C) : G^{(p)}_{0,4} = \begin{pmatrix} 1&1&1&1 \\ \hline 1&1&0&0 \\ 1&0&1&0 \end{pmatrix}$$

Sectionalized Trellis of a Linear Block Code

- Given a linear code C, let $C_{a,b}$ be its subcode, such that all its codewords have non-zero symbols only in positions $a \leq i < b$
- Let $p_{a,b}(C)$ be a linear code obtained by puncturing all symbols, except those in positions $a \leq i < b$, from codewords of C
- Let $s_{a,b}(C) = p_{a,b}(C_{a,b})$, i.e. a code obtained from C by shortening it on all symbols except those with indices $a \leq i < b$.
- Trellis paths from time a to time b correspond to cosets in $p_{a,b}(C)/s_{a,b}(C)$. The same coset may occur several times in a trellis

$$C : G = \begin{pmatrix} 1&1&1&1&1&1&1&1 \\ 1&1&1&1&0&0&0&0 \\ 1&1&0&0&1&1&0&0 \\ 1&0&1&0&1&0&1&0 \end{pmatrix}$$

$$p_{0,4}(C) : G^{(p)}_{0,4} = \begin{pmatrix} 1&1&1&1 \\ \hline 1&1&0&0 \\ 1&0&1&0 \end{pmatrix}$$

$$s_{0,4}(C) : \overline{G}^{(s)}_{0,4} = \begin{pmatrix} 1&1&1&1 \end{pmatrix}$$

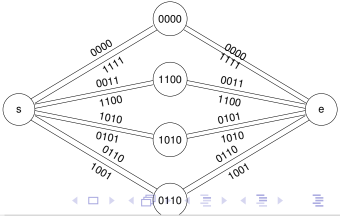
Sectionalized Trellis of a Linear Block Code

- Given a linear code C, let $C_{a,b}$ be its subcode, such that all its codewords have non-zero symbols only in positions $a \leq i < b$
- Let $p_{a,b}(C)$ be a linear code obtained by puncturing all symbols, except those in positions $a \leq i < b$, from codewords of C
- Let $s_{a,b}(C) = p_{a,b}(C_{a,b})$, i.e. a code obtained from C by shortening it on all symbols except those with indices $a \leq i < b$.
- Trellis paths from time a to time b correspond to cosets in $p_{a,b}(C)/s_{a,b}(C)$. The same coset may occur several times in a trellis

$$C : G = \begin{pmatrix} 1&1&1&1&1&1&1&1 \\ 1&1&1&1&0&0&0&0 \\ 1&1&0&0&1&1&0&0 \\ 1&0&1&0&1&0&1&0 \end{pmatrix}$$

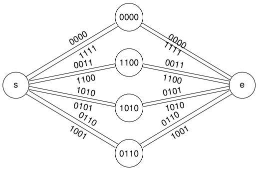
$$p_{0,4}(C) : G^{(p)}_{0,4} = \left(\begin{array}{cccc} 1&1&1&1 \\ \hline 1&1&0&0 \\ 1&0&1&0 \end{array}\right)$$

$$s_{0,4}(C) : \overline{G}^{(s)}_{0,4} = \begin{pmatrix} 1&1&1&1 \end{pmatrix}$$



Recursive Trellis Decoding of Linear Block Codes

- For each coset $D \in p_{a,b}(C)/s_{a,b}(C)$ find the most probable element $I(D)$, and $m(D) = M(I(D), y_a^{b-1})$
- *Composite branch table* $T_{a,b}$ stores $(I(D), m(D))$
- ML decoding of (n, k) code C: $p_{0,n}(C)/s_{0,n}(C)$ contains a single element, so $T_{0,n}$ contains the solution of the ML decoding problem
- Construction[14] of $T_{a,b}$: $b - a \geq 2$:
  - Let $z : a < z < b - 1$ be a subsection boundary
  - Consider all combinations of cosets $D' \in p_{a,z}(C)/s_{a,z}(C)$, $D'' \in p_{z,b}(C)/s_{z,b}(C)$, such that $D' \bullet D'' = D \in p_{a,b}(C)/s_{a,b}(C)$, i.e. concatenation of any their representatives is in $D \in p_{a,b}(C)/s_{a,b}(C)$
  - $m(D) = \max_{D',D''}(m(D') + m(D'')), I(D) = I(D') \bullet I(D'')$



[14] T. Fujiwara, H. Yamamoto, T. Kasami, and S. Lin, "A trellis-based recursive maximum-likelihood decoding algorithm for binary linear block codes," IEEE Transactions On Information Theory, vol. 44, no. 2, March 1998

Generator Matrices of Section Codes

- Generator matrix of $p_{a,b}(C)$ is $G^{(p)}_{a,b} = \begin{pmatrix} G^{(s)}_{a,z} & 0 \\ 0 & G^{(s)}_{z,b} \\ G^{(00)}_{a,b} & G^{(01)}_{a,b} \\ \hline G^{(10)}_{a,b} & G^{(11)}_{a,b} \end{pmatrix}$
- Generator matrix of $s_{a,b}(C)$ is $G^{(s)}_{a,b} = \begin{pmatrix} G^{(s)}_{a,z} & 0 \\ 0 & G^{(s)}_{z,b} \\ G^{(00)}_{a,b} & G^{(01)}_{a,b} \end{pmatrix}$
- $G^{(00)}_{a,b}, G^{(01)}_{a,b}$ are some $k''_{a,b} \times (z-a)$ and $k''_{a,b} \times (b-z)$ matrices
- $G^{(10)}_{a,b}, G^{(11)}_{a,b}$ are some $k'_{a,b} \times (z-a)$ and $k'_{a,b} \times (b-z)$ matrices
- One-to-one correspondence between $vG'_{a,b}$, where $G'_{a,b} = \begin{pmatrix} G^{(10)}_{a,b} & G^{(11)}_{a,b} \end{pmatrix}$, and cosets $D \in p_{a,b}(C)/s_{a,b}(C)$. CBT entries are indexed by $v \in \mathbb{F}_2^{k'_{a,b}}$

Merging the Composite Branch Tables

- $CBT_{a,b}[v].m = \max_{w \in \mathbb{F}_2^{k''_{a,b}}} \left(CBT_{a,z}[A].m + CBT_{z,b}[B].m \right), v \in \mathbb{F}_2^{k'_{a,b}}$
  where A and B are indices of the cosets $D' \in p_{a,z}(C)/s_{a,z}(C)$ and $D'' \in p_{z,b}(C)/s_{z,b}(C)$, such that $\begin{pmatrix} w & v \end{pmatrix} \begin{pmatrix} G_{a,b}^{(00)} \\ G_{a,b}^{(10)} \end{pmatrix} \in D'$ and $\begin{pmatrix} w & v \end{pmatrix} \begin{pmatrix} G_{a,b}^{(01)} \\ G_{a,b}^{(11)} \end{pmatrix} \in D''$
- Such values A, B can be identified from
  $$\begin{pmatrix} A' & A \end{pmatrix} \begin{pmatrix} G_{a,z}^{(s)} \\ G'_{a,z} \end{pmatrix} = \begin{pmatrix} w & v \end{pmatrix} \begin{pmatrix} G_{a,b}^{(00)} \\ G_{a,b}^{(10)} \end{pmatrix} \qquad \begin{pmatrix} B' & B \end{pmatrix} \begin{pmatrix} G_{z,b}^{(s)} \\ G'_{z,b} \end{pmatrix} = \begin{pmatrix} w & v \end{pmatrix} \begin{pmatrix} G_{a,b}^{(01)} \\ G_{a,b}^{(11)} \end{pmatrix},$$
  where A', B' are the vectors not used elsewhere
- The solutions are $A = \begin{pmatrix} w & v \end{pmatrix} \widehat{G}_{a,b}$ and $B = \begin{pmatrix} w & v \end{pmatrix} \widetilde{G}_{a,b}$ for some $\widehat{G}_{a,b}$ and $\widetilde{G}_{a,b}$
- Complexity is $O(2^{k'_{a,b}+k''_{a,b}})$

Recursive Trellis Processing of Polarization Kernels

- Let S_0^{l-1} be the input LLR vector, i.e. $S_j = \log \frac{W(0|y_j)}{W(1|y_j)}$
- SC decoding requires computing

$$S_1^{(i)}(u_0^{i-1}, y_0^{l-1}) = \max_{u_{i+1}^{l-1}} M((u_0^{i-1}, 0, u_{i+1}^{l-1})K, S_0^{l-1}) - \max_{u_{i+1}^{l-1}} M((u_0^{i-1}, 1, u_{i+1}^{l-1})K, S_0^{l-1}), i = 0, 1, \ldots, l-1$$

- Successive decoding in the cosets of extended kernel codes K
- The recursive trellises for cosets of a linear code have the same structure as the recursive trellis of the code itself
- Section codes $p_{a,b}(\overline{\mathcal{C}}^{(t)}), s_{a,b}(\overline{\mathcal{C}}^{(t)})$ may be identical for several phases t

Recursive Binary Kernel Processing

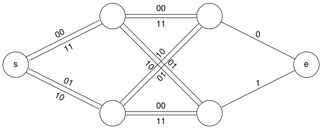
- t-th extended kernel code $\overline{\mathcal{C}}^{(t)}$ is generated by $\overline{K}^{(t)} = \begin{pmatrix} K[t] & 1 \\ K[t+1] & 0 \\ \vdots & \vdots \\ K[l-1] & 0 \end{pmatrix}$
- Due to invertibility of kernel K, one has $|p_{0,l}(\overline{\mathcal{C}}^{(t)})/s_{0,l}(\overline{\mathcal{C}}^{(t)})| = 2$
- Composite branch table for section $[0, l)$, denoted $T_{0,l}^{(t)}$, contains $M((u_0^{i-1}, u_i, u_{i+1}^{l-1})K, S_0^{l-1}), u_i \in \mathbb{F}_2$
- No need to store $I(D)$, only correlation discrepancies $m(D)$ should be computed

Reusing CBTs Across Phases

- Cosets $p_{a,b}(\overline{\mathcal{C}}^{(t)})/s_{a,b}(\overline{\mathcal{C}}^{(t)})$ may be identical for different phases t. Re-use the CBTs from prior phases to obtain further complexity savings
- Even if section codes are different, the CBT at phase t may be a subvector of the CBT at phase $t-1$

Example: 2-iterated Arikan Kernel $K_4 = B_{2,2}F_2^{\otimes 2} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$ I

1. $\overline{K}^{(0)} = \left(\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{array}\right)$



$$M(i, S_j) = w_{ij} = \begin{cases} 0, & i = \hat{c}_j \\ -|S_j|, & i \neq \hat{c}_j \end{cases}$$

$$S_j = \ln \frac{W(0|y_j)}{W(1|y_j)}$$

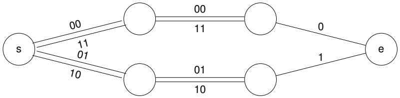
$p_{0,4}(\overline{\mathcal{C}}^{(0)})$ is a $(4,4,1)$ code, $s_{0,4}(\overline{\mathcal{C}}^{(0)})$ is a $(4,3,2)$ code
Coset representatives of $p_{0,4}(\overline{\mathcal{C}}^{(0)})/s_{0,4}(\overline{\mathcal{C}}^{(0)})$: $(0,0,0,0)$ and $(1,0,0,0)$

$p_{0,2}(\overline{\mathcal{C}}^{(0)})$, $p_{2,4}(\overline{\mathcal{C}}^{(0)})$ are $(2,2,1)$ codes, $s_{0,2}(\overline{\mathcal{C}}^{(0)})$ and $s_{2,4}(\overline{\mathcal{C}}^{(0)})$, are $(2,1,2)$ codes. Hence,

$$T_{0,2}^{(0)} = [\max(w_{00} + w_{01}, w_{10} + w_{11}), \max(w_{10} + w_{01}, w_{00} + w_{11})]$$
$$T_{2,4}^{(0)} = [\max(w_{02} + w_{03}, w_{12} + w_{13}), \max(w_{12} + w_{03}, w_{02} + w_{13})]$$
$$T_{0,4}^{(0)} = [\max(T_{0,2}^{(0)}[0] + T_{2,4}^{(0)}[0], T_{0,2}^{(0)}[1] + T_{2,4}^{(0)}[1]), \max(T_{0,2}^{(0)}[0] + T_{2,4}^{(0)}[1], T_{0,2}^{(0)}[1] + T_{2,4}^{(0)}[0])]$$

Example: 2-iterated Arikan Kernel $K_4 = B_{2,2}F_2^{\otimes 2} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$ II

2. $\overline{K}^{(0)} = \left(\begin{array}{cccc|c} 1 & 0 & 1 & 0 & 1 \\ \hline 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{array}\right)$



$p_{x,x+2}(\overline{\mathcal{C}}^{(1)}) = p_{x,x+2}(\overline{\mathcal{C}}^{(0)})$ and $s_{x,x+2}(\overline{\mathcal{C}}^{(1)}) = s_{x,x+2}(\overline{\mathcal{C}}^{(0)})$ for $x \in \{0, 2\}$

$p_{0,4}(\overline{\mathcal{C}}^{(1)})$ is a $(4, 3, 2)$ code, while $s_{0,4}(\overline{\mathcal{C}}^{(1)})$ is a $(4, 2, 2)$ code generated by two last rows of K_4

Consider the cosets of the latter code given by vectors $(0, 0, 0, 0)$ and $(1, 0, 1, 0)$. Assuming $u_0 = 0$, one obtains

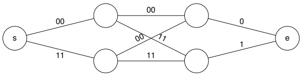
$$T_{0,4}^{(1)} = [T_{0,2}^{(0)}[0] + T_{2,4}^{(0)}[0], T_{0,2}^{(0)}[1] + T_{2,4}^{(0)}[1]].$$

Assuming $u_0 = 1$, one obtains

$$T_{0,4}^{(1)} = [T_{0,2}^{(0)}[1] + T_{2,4}^{(0)}[0], T_{0,2}^{(0)}[0] + T_{2,4}^{(0)}[1]].$$

Example: 2-iterated Arikan Kernel $K_4 = B_{2,2}F_2^{\otimes 2} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$ III

3. $\overline{K}^{(0)} = \left(\begin{array}{cccc|c} 1 & 1 & 0 & 0 & 1 \\ \hline 1 & 1 & 1 & 1 & 0 \end{array}\right)$



$$w_{ij} = \begin{cases} 0, & i = \hat{c}_j \\ -|S_j|, & i \neq \hat{c}_j \end{cases}$$

$$S_j = \ln \frac{W(0|y_j)}{W(1|y_j)}$$

$p_{0,2}(\overline{\mathcal{C}}^{(2)})$ and $p_{2,4}(\overline{\mathcal{C}}^{(2)})$ are $(2,1,2)$ codes, while $s_{0,2}(\overline{\mathcal{C}}^{(2)})$ and $s_{2,4}(\overline{\mathcal{C}}^{(2)})$ are $(2,0,\infty)$ codes. $p_{0,4}(\overline{\mathcal{C}}^{(2)})$ is a $(4,2,2)$ code, $s_{0,4}(\overline{\mathcal{C}}^{(2)})$ is a $(4,1,4)$ code. Cosets of $s_{0,4}(\overline{\mathcal{C}}^{(2)})$ are given by $(0,0,0,0)$ and $(1,1,0,0)$. For $u_0 = u_1 = 0$ one has

$$T_{0,2}^{(2)} = [w_{00} + w_{01}, w_{10} + w_{11}];\ T_{2,4}^{(2)} = [w_{02} + w_{03}, w_{12} + w_{13}]$$

$$T_{0,4}^{(2)} = [\max(T_{0,2}^{(2)}[0] + T_{2,4}^{(2)}[0], T_{0,2}^{(2)} + T_{2,4}^{(2)}[1]), \max(T_{0,2}^{(2)}[0] + T_{2,4}^{(2)}[1], T_{0,2}^{(2)}[1] + T_{2,4}^{(2)}[0])].$$

Example: 2-iterated Arikan Kernel $K_4 = B_{2,2}F_2^{\otimes 2} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$ IV

4. $p_{0,2}(\overline{\mathcal{C}}^{(3)}) = p_{0,2}(\overline{\mathcal{C}}^{(2)}) = p_{2,4}(\overline{\mathcal{C}}^{(2)}) = p_{2,4}(\overline{\mathcal{C}}^{(3)})$ are $(2,1,2)$ codes, while $s_{0,2}(\overline{\mathcal{C}}^{(2)})$ and $s_{2,4}(\overline{\mathcal{C}}^{(2)})$ are $(2,0,\infty)$ codes. $p_{0,4}(\overline{\mathcal{C}}^{(3)})$ is a $(4,1,4)$ code, $s_{0,4}(\overline{\mathcal{C}}^{(4)})$ is a $(4,0,\infty)$ code.

$$T_{0,4}^{(3)} = [T_{0,2}^{(2)}[0] + T_{2,4}^{(2)}[0], T_{0,2}^{(2)}[1] + T_{2,4}^{(2)}[1]].$$

A Simplification

- Consider computing $T=[\max(w_{00}+w_{01}, w_{10}+w_{11}), \max(w_{10}+w_{01}, w_{00}+w_{11})]$,

$$w_{ij}=\begin{cases}0, & i=\hat{c}_j\\ -|S_j|, & i\neq \hat{c}_j,\end{cases}$$

  $\hat{c}_j$ is the hard decision corresponding to LLR $S_j=\ln\frac{W(0|y_j)}{W(1|y_j)}$
- The final result of kernel processing does not change if the same value is subtracted from all CBT entries at any section

$$\begin{aligned}\widetilde{T}=&[\max(w_{00}+w_{01}, w_{10}+w_{11})-w_{10}-w_{11}, \max(w_{10}+w_{01}, w_{00}+w_{11})-w_{10}-w_{11}]\\ =&[\max(w_{00}-w_{10}+w_{01}-w_{11}, 0), \max(w_{01}-w_{11}, w_{00}-w_{10})]\\ =&[\max(S_0+S_1, 0), \max(S_1, S_0)]\\ \widehat{T}=&[\max(S_0+S_1, 0)-\max(S_1, S_0), 0]=[\operatorname{sgn}(S_0)\operatorname{sgn}(S_1)\min(|S_0|, |S_1|), 0]\end{aligned}$$

- Complexity: 6 operations $\rightarrow$ 1 operation

Yet Another Simplification

$$T_{0,4}^{(1)} = \begin{cases} [T_{0,2}^{(0)}[0] + T_{2,4}^{(0)}[0], T_{0,2}^{(0)}[1] + T_{2,4}^{(0)}[1]], & u_0 = 0 \\ [T_{0,2}^{(0)}[1] + T_{2,4}^{(0)}[0], T_{0,2}^{(0)}[0] + T_{2,4}^{(0)}[1]], & u_0 = 1 \end{cases}$$

The final result of kernel processing does not change if the same value is subtracted from all CBT entries at any section

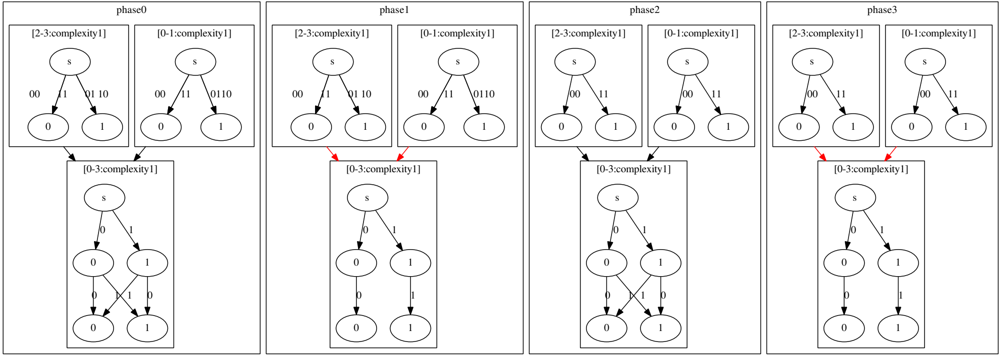
$$\widehat{T}_{0,4}^{(1)} = \begin{cases} [T_{0,2}^{(0)}[0] - T_{0,2}^{(0)}[1] + T_{2,4}^{(0)}[0] - T_{2,4}^{(0)}[1], 0], & u_0 = 0 \\ [T_{0,2}^{(0)}[1] - T_{0,2}^{(0)}[0] + T_{2,4}^{(0)}[0] - T_{2,4}^{(0)}[1], 0], & u_0 = 1 \end{cases}$$

$$\widehat{T}_{0,4}^{(1)} = [(-1)^{u_1}(T_{0,2}^{(0)}[0] - T_{0,2}^{(0)}[1]) + (T_{2,4}^{(0)}[0] - T_{2,4}^{(0)}[1]), 0]$$

If we know that $T_{a,b}^{(0)}[1] = 0$, this simplifies to

$$\widehat{T}_{0,4}^{(1)} = [(-1)^{u_1} T_{0,2}^{(0)}[0] + T_{2,4}^{(0)}[0], 0]$$

Example: 2-iterated Arikan Kernel $K_4 = B_{2,2}F_2^{\otimes 2} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$



- For the Arikan matrix $B_{2,m}F_2^{\otimes m}$, changes of section codes follow the same pattern as in the successive cancellation algorithm
- Min-Sum SC algorithm for an Arikan polar code of length 2^m is a special case of the recursive trellis processing algorithm for kernel $B_{2,m}F_2^{\otimes m}$

Successive Maximization

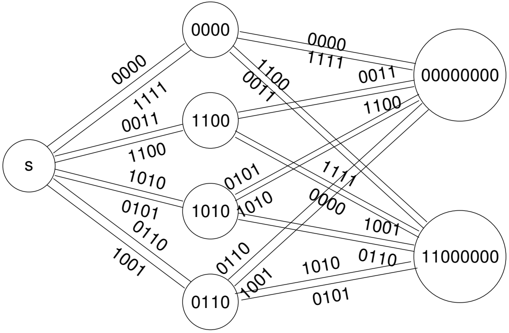
$$
\begin{aligned}
T_{a,b}[v].m &= \max_{w \in \mathbb{F}_2^{k''_{i,a,b}}} \left(T_{x,z}[A].m + T_{z,y}[B].m \right) \\
&= \max_{w_{k''_{i,a,b}-1}} \dots \max_{w_1} \max_{w_0} \left(T_{x,z}[A].m + T_{z,y}[B].m \right),
\end{aligned}
$$

where $A = (w \quad v)\, \widehat{G}_{a,b}$ and $B = (w \quad v)\, \widetilde{G}_{a,b}$

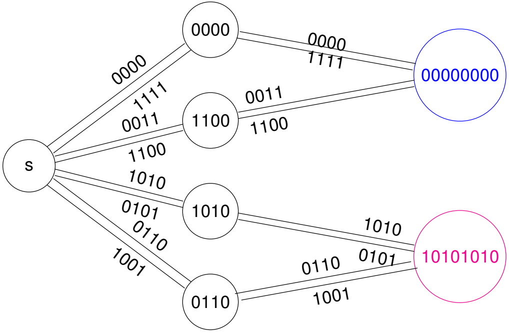
- Keep intermediate results of maximization
- Re-use saved intermediate results at later phases

Sorted Arikan Kernel: Maximization Forest

$$K_8 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$ has scaling exponent 3.577



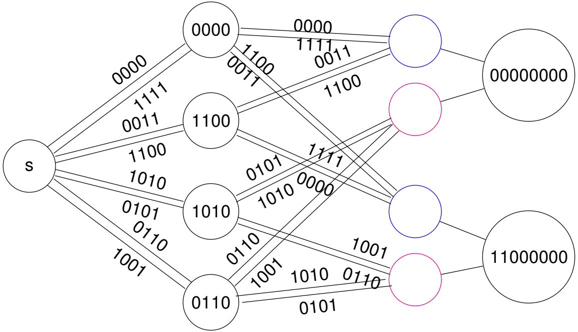
Trellis for $\overline{K}^{(3)}$



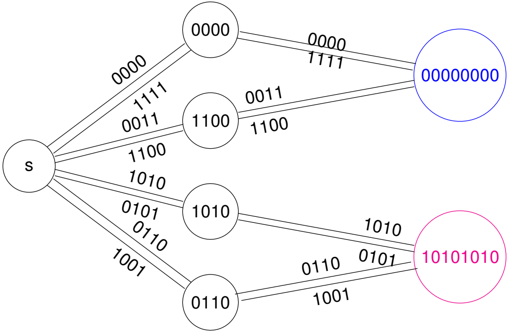
Trellis for $\overline{K}^{(4)}$

Sorted Arikan Kernel: Maximization Forest

$$K_8 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$ has scaling exponent 3.577

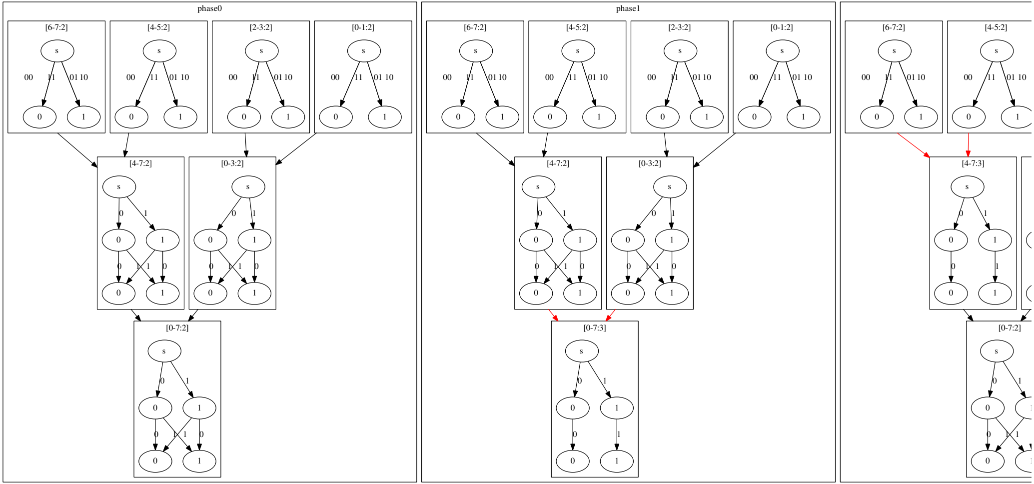


Trellis for $\overline{K}^{(3)}$

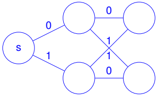
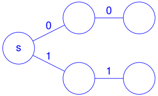
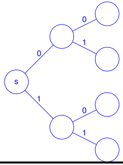


Trellis for $\overline{K}^{(4)}$

Sorted Arikan Kernel K_8: Recursive Trellises for Section Codes



Special Trellises

Type	Trellis	Simplified expression	Complexity M_j
1		$T_{a,b}[0] = \mathrm{sgn}(T_{a,z}[0])\,\mathrm{sgn}(T_{z,b}[0])\min(\lvert T_{a,z}[0]\rvert, \lvert T_{z,b}[0]\rvert)$ $T_{a,b}[1] = 0$	1
2		$T_{a,b}[0] = (-1)^{c'} T_{a,z}[0] + (-1)^{c''} T_{z,b}[0]$ $T_{a,b}[1] = 0$	1
3		Let $\hat{c}_0, \hat{c}_1$ be the hard decisions for $T_{a,z}[0], T_{z,b}[1]$ $T_{a,b}[\hat{c}_0, \hat{c}_1] = 0;\ T_{a,b}[1 \oplus \hat{c}_0, 1 \oplus \hat{c}_1] = -\lvert T_{a,z}[0]\rvert - \lvert T_{z,b}[0]\rvert$ $T_{a,b}[1 \oplus \hat{c}_0, \hat{c}_1] = -\lvert T_{a,z}[0]\rvert;\ T_{a,b}[\hat{c}_0, 1 \oplus \hat{c}_1] = -\lvert T_{z,b}[0]\rvert$	1

Complexity

Total complexity of kernel processing $\mathbf{C} = \sum_{i=0}^{l-1}(\delta_i + c_{i,0,l})$

- $\delta_i \in \{0, 1\}$ is the complexity of computing the final LLR from the obtained CBT
- Complexity of construction of the CBT for section $[a, b)$ at phase i

$$c_{i,a,b} = \begin{cases} m_{iab}, & \text{if CBTs for subsections can be reused} \\ m_{iab} + c_{i,a,z} + c_{i,z,b}, & \text{otherwise,} \end{cases}$$

- Complexity of computations at section $[a, b)$ on phase i

$$m_{ixy} = \begin{cases} 0, & \text{if forest reuse is possible} \\ M_j, & \text{if type-}j\text{ special trellis is encountered,} \\ 2^{k'_{iab}+k''_{iab}-f_{iab}} + 2^{k'_{iab}}(2^{k''_{iab}-f_{iab}} - 1), & \text{otherwise,} \end{cases}$$

$f_{iab} \in \{0, 1\}$ shows if an extra simplification is possible

Optimizing Sectionalization

- The complexity strongly depends on sectionalization, i.e. selection of $z : a < z < b$
- A dynamic programming algorithm with complexity $O(l^4)$ for finding an optimal sectionalization
- Example: 24×24 kernel
  - Uniform sectionalization ($z = (a + b)/2$): 715 summations, 449 comparisons
  - Optimized: 250 summations and 124 comparisons
    $[0, 24)$ splits into $[0, 16)$ and $[16, 24)$ with further uniform sectionalization

Complexity of Kernel Processing

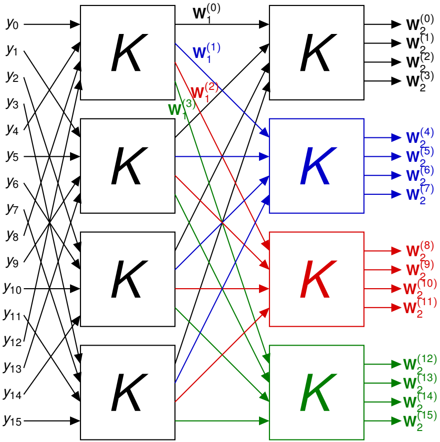
Kernel K_l	$E(K_l)$	$\mu(K_l)$	State of the art			Recursive trellis	
			Method	Add	Comp.	Add	Comp.
$K_{16}B_4$	0.51828	3.45	window[15]	95	86	131	105
$K_{32}B_5$	0.521936	3.417	window	297	274	406	262
K_{32}^r	0.52194	3.42111	Viterbi	4536	9072	355	191
K_{32}^{bch}	0.53656	3.1221	Viterbi	99745	199490	31079	28337
$K_{32}^{enbch'}$	0.53656	3.1221	window	2864420		32183	29873
K_{20}^*	0.506169	3.43827	Viterbi	7524	15054	2893	2001
K_{20}	0.49943	3.64931	Viterbi	1866	3756	289	189
K_{24}^*	0.51577	3.3113	Viterbi	9922	19860	1621	1207
K_{24}	0.502911	3.61903	Viterbi	2102	4218	241	124

- Pipelined implementation
- Latency $O(\log_2 l)$

[15] G.Trofimiuk, P. Trifonov. Window Processing of Binary Polarization Kernels. IEEE Transactions on Communications, 2021

Kernel Processing

- Successive cancellation of $(n = l^m, k)$ polar code with polarizing transform $K^{\otimes m}$ requires computing the probabilities $W_m^{(i)}(u_0^i|y_0^{n-1})$ for each $i \in [n]$
    - For non-frozen symbol u_i one should compute $W_m^{(i)}(\widehat{u}_0^{i-1}.0|y_0^{n-1})$ and $W_m^{(i)}(\widehat{u}_0^{i-1}.1|y_0^{n-1})$
    - $\widehat{u}_0^{i-1}$ are already determined by SC decoder
- The vector u_0^i is referred to as a *path*
- $W_m^{(i)}(u_0^i|y_0^{n-1})$ is computed recursively
- We consider computing the probabilities $W_1^{(i)}(u_0^i|y_0^{l-1})$ of only *one layer* of polarizing transform K



The Idea of Window Processing Algorithm

We consider processing of $l \times l, l = 2^t$, kernel K

- Let $\widetilde{W}_t^{(i)}(v_0^i|y_0^{l-1})$ be a probability of input symbols v_0^i for Arikan kernel $F_2^{\otimes t}$

The main idea of window processing is to compute $W_1^{(\phi)}(u_0^\phi|y_0^{l-1})$ using several values $\widetilde{W}_t^{(i)}(v_0^i|y_0^{l-1})$ for some $i \geq \phi$

- Motivation: the probabilities $\widetilde{W}_t^{(i)}(v_0^i|y_0^{l-1})$ are very easy to calculate

$$\begin{aligned}
\widetilde{W}_\lambda^{(2\psi)}(u_0^{2\psi}|y_0^{2^\lambda-1}) &= \sum_{u_{2\psi+1}\in\mathbb{F}_2} \widetilde{W}_{\lambda-1}^{(\psi)}(u_{0,even}^{2\psi+1}+u_{0,odd}^{2\psi+1}|y_{0,even}^{2^\lambda-1})\widetilde{W}_{\lambda-1}^{(\psi)}(u_{0,odd}^{2\psi+1}|y_{0,odd}^{2^\lambda-1}) \\
\widetilde{W}_\lambda^{(2\psi+1)}(u_0^{2\psi+1}|y_0^{2^\lambda-1}) &= \widetilde{W}_{\lambda-1}^{(\psi)}(u_{0,even}^{2\psi+1}+u_{0,odd}^{2\psi+1}|y_{0,even}^{2^\lambda-1})\widetilde{W}_{\lambda-1}^{(\psi)}(u_{0,odd}^{2\psi+1}|y_{0,odd}^{2^\lambda-1})
\end{aligned}$$

Transition Matrix

We need to establish the relation between the input vectors u and v of polarizing transforms K and $F_2^{\otimes t}$ respectively

- We can write $TK = F_2^{\otimes t}$, where T is referred to as the *transition matrix*.
- $c_0^{l-1} = v_0^{l-1} F_2^{\otimes t} = u_0^{l-1} K$

This implies that $u_0^{l-1} = v_0^{l-1} T$, or

$$u_\phi = \sum_{s=0}^{l-1} v_s T[s, \phi] = \sum_{\substack{s=0 \\ T_{[s,\phi]}=1}}^{\tau_\phi} v_s$$

- τ_ϕ denotes the position of the last non-zero symbol in the ϕ-th column of T
- We will also need $h_\phi = \max_{0 \le \phi' \le \phi} \tau_{\phi'}$

Transition Matrix

We need to establish the relation between the input vectors u and v of polarizing transforms K and $F_2^{\otimes t}$ respectively

- We can write $TK = F_2^{\otimes t}$, where T is referred to as the *transition matrix*.
- $c_0^{l-1} = v_0^{l-1} F_2^{\otimes t} = u_0^{l-1} K$

This implies that $u_0^{l-1} = v_0^{l-1} T$, or

$$u_\phi = \sum_{s=0}^{l-1} v_s T[s,\phi] = \sum_{\substack{s=0 \\ T_{[s,\phi]}=1}}^{\tau_\phi} v_s$$

- τ_ϕ denotes the position of the last non-zero symbol in the ϕ-th column of T
- We will also need $h_\phi = \max_{0 \le \phi' \le \phi} \tau_{\phi'}$

Relations Between Input vectors

We also need the expression for each component of v_0^l,

$$v_{\tau_\phi} = u_\phi + \sum_{\substack{s=0 \\ T_{[s,\phi]}=1}}^{\tau_\phi - 1} v_s \quad (2)$$

Some τ_ϕ might be equal, so, we transform (2)

1. $\Theta' = (\mathbb{T} \quad I)$, $\mathbb{T}$ is obtained by transposing T^{-1} and reversing the order of columns
2. Transform Θ' into a minimum-span form Θ, ϕ-th row starts in ϕ and ends in z_ϕ column
3. Compute

$$u_\phi = \sum_{s=0}^{\phi-1} u_s \Theta_{l-1-\phi, l-1-s} + \sum_{j=0}^{\omega_\phi} v_j \Theta_{l-1-\phi, l+j}$$

where $\omega_\phi = z_{l-1-\phi} - l$

$$v_{\omega_\phi} = \sum_{s=0}^{\phi} u_s \Theta_{l-1-\phi, l-1-s} + \sum_{j=0}^{\omega_\phi - 1} v_j \Theta_{l-1-\phi, l+j}.$$

Example of Transition Matrix

$$K'_{16} = \begin{pmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0\\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
0&1&1&0&1&1&0&0&1&0&1&0&0&0&0&0\\
1&1&0&0&1&0&1&0&0&1&1&0&0&0&0&0\\
1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0\\
1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0\\
1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix}, \quad F_2^{\otimes 4} = \begin{pmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0\\
1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0\\
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0\\
1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix}$$

Example of Transition Matrix

$$
T = \begin{array}{r}0\\1\\2\\3\\4\\5\\6\\7\\8\\9\\10\\11\\12\\13\\14\\15\end{array}
\begin{pmatrix}
\mathbf{1}&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&\mathbf{1}&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&\mathbf{1}&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&\mathbf{1}&0&0&0&0&0&0&0&0\\
0&0&0&\mathbf{1}&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&\mathbf{1}&0&0&0&0&0&0\\
0&0&0&0&0&1&1&0&0&0&\mathbf{1}&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&\mathbf{1}&0&0&0&0\\
0&0&0&0&\mathbf{1}&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&\mathbf{1}&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&\mathbf{1}&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&\mathbf{1}&0&0&0\\
0&0&0&0&0&0&0&0&\mathbf{1}&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&\mathbf{1}&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&\mathbf{1}&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&\mathbf{1}
\end{pmatrix}
$$

$$
\begin{array}{r}
\tau = \\ h =
\end{array}
\begin{array}{cccccccccccccccc}
0&1&2&4&8&9&10&3&12&5&6&7&11&13&14&15\\
0&1&2&4&8&9&10&10&12&12&12&12&12&13&14&15
\end{array}
$$

$$
\begin{aligned}
u_0 &= v_0\\
u_1 &= v_1\\
u_2 &= v_2\\
u_3 &= v_4\\
u_4 &= v_8\\
u_5 &= v_6 \oplus v_9\\
u_6 &= v_5 \oplus v_6 \oplus v_{10}\\
u_7 &= v_3\\
u_8 &= v_{12}\\
u_9 &= v_5\\
u_{10} &= v_6\\
u_{11} &= v_7\\
u_{12} &= v_{11}\\
u_{13} &= v_{13}\\
u_{14} &= v_{14}\\
u_{15} &= v_{15}
\end{aligned}
$$

Decoding Window

- We are able to reconstruct u_0^ϕ from $v_0^{h_\phi}$
- If $h_\phi > \phi$, then some values of $v_0^{h_\phi}$ are independent from u_0^ϕ and, therefore, unknown
- By *decoding window* we denote the set

$$D_\phi = [h_\phi + 1] \setminus \{\omega_0, \omega_1, \ldots, \omega_\phi\}$$

of indices of independent (from u_0^ϕ) components of $v_0^{h_\phi}$

$$W_1^{(\phi)}(u_0^\phi | y_0^{l-1}) = \sum_{v_0^{h_\phi} \in \mathcal{Z}_\phi^{(u_\phi)}} \widetilde{W}_t^{(h_\phi)}(v_0^{h_\phi} | y_0^{l-1})$$

- $\mathcal{Z}_\phi^{(b)}$ is the set of vectors $v_0^{h_\phi}$, such as $v_s \in \mathbb{F}_2, s \in D_\phi$, the values of $v_t, t \in [h_\phi + 1] \setminus D_\phi$, are obtained according to the transition matrix T and $u_\phi = b$
- $|D_\phi| = h_\phi - \phi$

Decoding Window

- We are able to reconstruct u_0^ϕ from $v_0^{h_\phi}$
- If $h_\phi > \phi$, then some values of $v_0^{h_\phi}$ are independent from u_0^ϕ and, therefore, unknown
- By *decoding window* we denote the set

$$D_\phi = [h_\phi + 1] \setminus \{\omega_0, \omega_1, \dots, \omega_\phi\}$$

of indices of independent (from u_0^ϕ) components of $v_0^{h_\phi}$

$$W_1^{(\phi)}(u_0^\phi | y_0^{l-1}) = \sum_{v_0^{h_\phi} \in \mathcal{Z}_\phi^{(u_\phi)}} \widetilde{W}_t^{(h_\phi)}(v_0^{h_\phi} | y_0^{l-1})$$

- $\mathcal{Z}_\phi^{(b)}$ is the set of vectors $v_0^{h_\phi}$, such as $v_s \in \mathbb{F}_2, s \in D_\phi$, the values of $v_t, t \in [h_\phi + 1] \setminus D_\phi$, are obtained according to the transition matrix T and $u_\phi = b$
- $|D_\phi| = h_\phi - \phi$

Decoding Window. Example

ϕ	K'_{16}	
	u_ϕ	D_ϕ
0	v_0	$\{\}$
1	v_1	$\{\}$
2	v_2	$\{\}$
3	v_4	$\{3\}$
4	v_8	$\{3,5,6,7\}$
5	$v_6 \oplus v_9$	$\{3,5,6,7\}$
6	$v_5 \oplus v_6 \oplus v_{10}$	$\{3,5,6,7\}$
7	v_3	$\{5,6,7\}$
8	v_{12}	$\{5,6,7,11\}$
9	v_5	$\{6,7,11\}$
10	v_6	$\{7,11\}$
11	v_7	$\{11\}$
12	v_{11}	$\{\}$
13	v_{13}	$\{\}$
14	v_{14}	$\{\}$
15	v_{15}	$\{\}$

Probability Computation Example

Consider window processing of $\phi = 6$ of K'_{16} kernel

$$W_1^{(6)}(u_0^6|y_0^{15}) = \sum_{u_7^{15}\in\mathbb{F}_2^9} W_1^{(15)}(u_0^{15}|y_0^{15}) = \sum_{u_7^{15}\in\mathbb{F}_2^9} \prod_{i=0}^{n-1} W((u_0^{15}K'_{16})_i|y_0^{15})$$

- We have the constraint $u_6 = v_5 \oplus v_6 \oplus v_{10} \Rightarrow \tau_6 = h_6 = 10 \Rightarrow$ we should consider paths v_0^{10} and their probabilities $\widetilde{W}_4^{(10)}(v_0^{10}|y_0^{15})$
- How to construct paths v_0^{10} from already estimated by SC decoding symbols $\widehat{u}_0^5$?
  - $v_0 = \widehat{u}_0$, $v_1 = \widehat{u}_1$, $v_2 = \widehat{u}_2$, $v_4 = \widehat{u}_3$, $v_8 = \widehat{u}_4$
  - v_3, v_5, v_6, v_7 are not defined by $\widehat{u}_0^5 \Rightarrow$ decoding window $D_6 = \{3, 5, 6, 7\}$
  - We should consider all $(v_3, v_5, v_6, v_7) \in \mathbb{F}_2^4$
  - For given values v_5, v_6 and u_6, we have $v_9 = \widehat{u}_5 \oplus v_6$ and $v_{10} = u_6 \oplus v_5 \oplus v_6$

Probability Computation Example

Set of considered Arikan SC paths v_0^{10}:

$$\mathcal{Z}_6^{(u_6)} = \begin{matrix}
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 0, & \widehat{u}_3, & 0, & 0, & 0, & \widehat{u}_4, & \widehat{u}_5, & u_6 &],\\
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 0, & \widehat{u}_3, & 0, & 0, & 1, & \widehat{u}_4, & \widehat{u}_5, & u_6 &],\\
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 0, & \widehat{u}_3, & 0, & 1, & 0, & \widehat{u}_4, & \widehat{u}_5, & u_6 \oplus 1 &],\\
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 0, & \widehat{u}_3, & 0, & 1, & 1, & \widehat{u}_4, & \widehat{u}_5, & u_6 \oplus 1 &],\\
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 0, & \widehat{u}_3, & 1, & 0, & 0, & \widehat{u}_4, & \widehat{u}_5 \oplus 1, & u_6 \oplus 1 &],\\
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 0, & \widehat{u}_3, & 1, & 0, & 1, & \widehat{u}_4, & \widehat{u}_5 \oplus 1, & u_6 \oplus 1 &],\\
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 0, & \widehat{u}_3, & 1, & 1, & 0, & \widehat{u}_4, & \widehat{u}_5 \oplus 1, & u_6 &],\\
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 0, & \widehat{u}_3, & 1, & 1, & 1, & \widehat{u}_4, & \widehat{u}_5 \oplus 1, & u_6 &],\\
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 1, & \widehat{u}_3, & 0, & 0, & 0, & \widehat{u}_4, & \widehat{u}_5, & u_6 &],\\
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 1, & \widehat{u}_3, & 0, & 0, & 1, & \widehat{u}_4, & \widehat{u}_5, & u_6 &],\\
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 1, & \widehat{u}_3, & 0, & 1, & 0, & \widehat{u}_4, & \widehat{u}_5, & u_6 \oplus 1 &],\\
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 1, & \widehat{u}_3, & 0, & 1, & 1, & \widehat{u}_4, & \widehat{u}_5, & u_6 \oplus 1 &],\\
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 1, & \widehat{u}_3, & 1, & 0, & 0, & \widehat{u}_4, & \widehat{u}_5 \oplus 1, & u_6 \oplus 1 &],\\
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 1, & \widehat{u}_3, & 1, & 0, & 1, & \widehat{u}_4, & \widehat{u}_5 \oplus 1, & u_6 \oplus 1 &],\\
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 1, & \widehat{u}_3, & 1, & 1, & 0, & \widehat{u}_4, & \widehat{u}_5 \oplus 1, & u_6 &],\\
[\widehat{u}_0, & \widehat{u}_1, & \widehat{u}_2, & 1, & \widehat{u}_3, & 1, & 1, & 1, & \widehat{u}_4, & \widehat{u}_5 \oplus 1, & u_6 &]
\end{matrix}$$

Subchannel probability:

$$W_1^{(6)}(u_0^6|y_0^{15}) = \sum_{v_0^{10} \in \mathcal{Z}_6^{(u_6)}} \widetilde{W}_4^{(10)}(v_0^{10}|y_0^{15})$$

For **one** K'_{16} probability we need to calculate **16** probabilities for Arikan matrix $F_2^{\otimes 4}$

Log-Likelihood Ratios for Arikan Matrix

- Approximate bit subchannel probabilities

$$\widetilde{\mathcal{W}}_t^{(i)}(v_0^i|y_0^{l-1}) = \max_{v_{i+1}^{l-1}\in\mathbb{F}_2^{l-i-1}} \widetilde{W}_t^{(l-1)}(v_0^{l-1}|y_0^{l-1})$$

- Modified log-likelihood ratios

$$\widetilde{S}_t^{(i)}(v_0^{i-1}, y_0^{l-1}) = \log \frac{\widetilde{\mathcal{W}}_t^{(i)}(v_0^{i-1}.0|y_0^{l-1})}{\widetilde{\mathcal{W}}_t^{(i)}(v_0^{i-1}.1|y_0^{l-1})}$$

- Recursive computation

$$\widetilde{S}_\lambda^{(2i)}(v_0^{2i-1}, y_0^{N-1}) = Q(a,b) = \operatorname{sgn}(a)\operatorname{sgn}(b)\min(|a|,|b|)$$
$$\widetilde{S}_\lambda^{(2i+1)}(v_0^{2i}, y_0^{N-1}) = P(a,b,v_{2i}) = (-1)^{v_{2i}}a+b,$$
$$a = \widetilde{S}_{\lambda-1}^{(i)}(v_{0,even}^{2i-1}\oplus v_{0,odd}^{2i-1}, y_{0,even}^{N-1}), b = \widetilde{S}_{\lambda-1}^{(i)}(v_{0,odd}^{2i-1}, y_{0,odd}^{N-1}),$$

Path Score

- The log-likelihood of a path v_0^i can be obtained

$$R_y(v_0^i) = R(v_0^i|y_0^{l-1}) = \log \widetilde{\mathcal{W}}_t^{(i)}(v_0^i|y_0^{l-1}) = R_y(v_0^{i-1}) + \tau\left(\widetilde{S}_t^{(i)}(v_0^{i-1}, y_0^{l-1}), v_i\right),$$

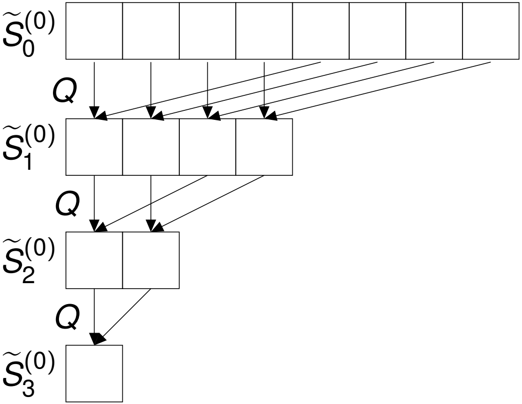
where $R_y(\epsilon)$ can be set to 0, ϵ is an empty sequence, and

$$\tau(S, v) = \begin{cases} 0, & \operatorname{sgn}(S) = (-1)^v \\ -|S|, & \text{otherwise.} \end{cases}$$

- The intermediate LLRs $\widetilde{S}_\lambda^{(i)} = \widetilde{S}_\lambda^{(i)}(v_0^{i-1}|y_0^{N-1})$ can be reused during the computation of $\widetilde{S}_t^{(i)}$ for different i

Illustration of LLR Computation

Phase 0



Phase 1

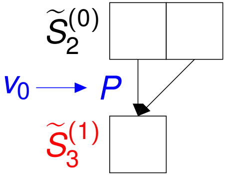
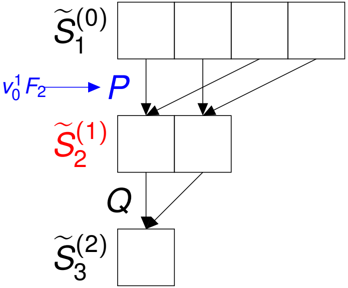
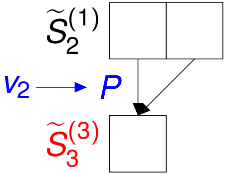


Illustration of LLR Computation

Phase 2



Phase 3



Window Processing in LLR domain

Approximate probabilities

$$\mathcal{W}_1^{(\phi)}(u_0^{\phi}|y_0^{l-1}) = \max_{v_0^{h_\phi} \in \mathcal{Z}_\phi^{(u_\phi)}} \widetilde{\mathcal{W}}_t^{(h_\phi)}(v_0^{h_\phi}|y_0^{l-1})$$

Log likelihood ratios

$$S_1^{(\phi)}(u_0^{\phi-1}, y_0^{l-1}) = \max_{v_0^{h_\phi} \in \mathcal{Z}_\phi^{(0)}} R_y(v_0^{h_\phi}) - \max_{v_0^{h_\phi} \in \mathcal{Z}_\phi^{(1)}} R_y(v_0^{h_\phi})$$

Summary of window processing:

- Compute the transition matrix T offline
- For each phase: compute $|\mathcal{Z}_\phi^{(0)}| + |\mathcal{Z}_\phi^{(1)}| = 2^{|D_\phi|+1}$ paths scores $R_y(v_0^{h_\phi})$, two maximums and obtain $S_1^{(\phi)}(u_0^{\phi-1}, y_0^{l-1})$

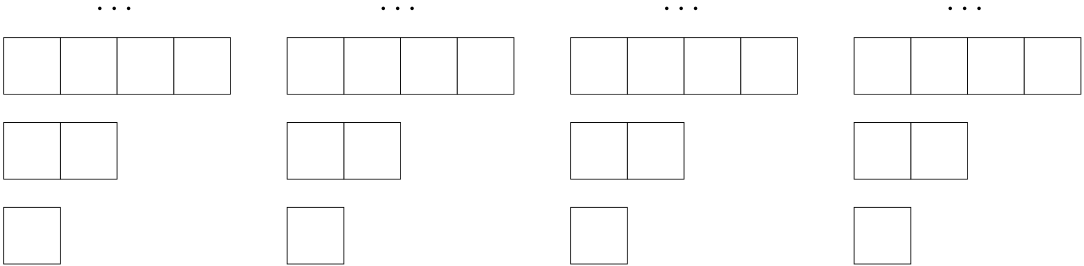
Simplifications of Window Processing[16]

- In many cases we can reuse intermediate LLRs $\widetilde{S}_{\lambda}^{(i)}$ arising during the for of $R_y(v_0^{h_\phi})$ corresponding to all $v_0^{h_\phi}$ generated by the decoding window
  - For instance, consider $h_{\phi-1} = 8, h_\phi = 9$, to compute $\widetilde{S}_t^{(9)}(v_0^8, y_0^{l-1})$ at phase $h_\phi = 9$ we can reuse intermediate LLRs $S_{t-1}^{(4)}$ obtained at the phase $h_{\phi-1} = 8$
- Joint computation of path scores $R_y(v_0^{h_\phi})$
- It is possible to reuse results of path scores maximization
  - In case of $h_\phi = h_{\phi+1} = \cdots = h_{\phi+q}$ paths scores $R_y(v_0^{h_\phi})$ remains the same. One can use maximization forest to obtain $S_1^{(\phi)}(u_0^{\phi-1}, y_0^{l-1})$, $S_1^{(\phi+1)}(u_0^{\phi}, y_0^{l-1}), \cdots,$ $S_1^{(\phi+q-)}(u_0^{\phi+q-1}, y_0^{l-1})$
  - In case of $h_{\phi+1} = h_\phi + 1$ (which implies that $D_{\phi+1} = D_\phi$) one half of computed $R_y(v_0^{h_\phi})$ remains the same

[16] G. Trofimiuk and P. Trifonov, "Window Processing of Binary Polarization Kernels," in IEEE Transactions on Communications, July 2021

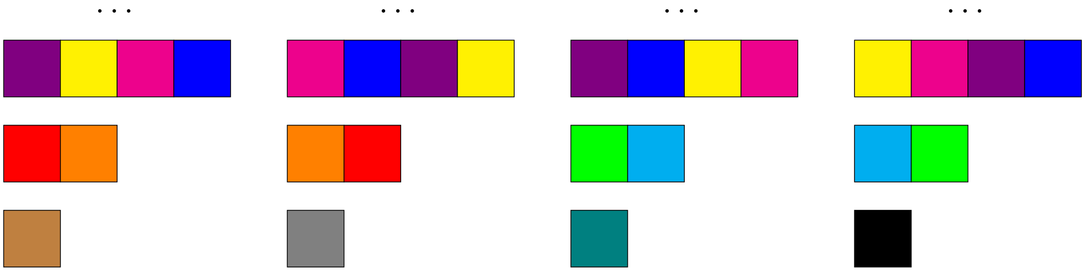
The Idea of Identification of Common Subexpressions

In window processing we need to compute several $F_2^{\otimes t}$ LLRs



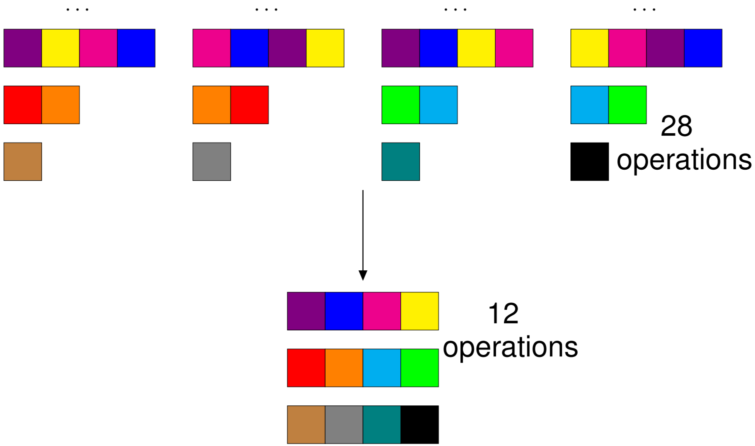
The Idea of Identification of Common Subexpressions

Some of intermediate LLRs $\widetilde{S}_{\lambda}^{(i)}$ can be the same for different paths



The Idea of Identification of Common Subexpressions

We can compute only **unique** intermediate LLRs $\widetilde{S}_{\lambda}^{(i)}$



Arithmetic Complexity of Window Processing[17]

ϕ	$K'_{16}, E=0.51828, \mu=3.346$			$K_{16}, E=0.51828, \mu=3.45$		
	u_ϕ	D_ϕ	Cost	u_ϕ	D_ϕ	Cost
0	v_0	$\{\}$	15	v_0	$\{\}$	15
1	v_1	$\{\}$	1	v_1	$\{\}$	1
2	v_2	$\{\}$	3	v_2	$\{\}$	3
3	v_4	$\{3\}$	21	v_3	$\{\}$	1
4	v_8	$\{3,5,6,7\}$	127	v_4	$\{\}$	7
5	$v_6 \oplus v_9$	$\{3,5,6,7\}$	48	v_8	$\{5,6,7\}$	67
6	$v_5 \oplus v_6 \oplus v_{10}$	$\{3,5,6,7\}$	95	$v_6 \oplus v_9$	$\{5,6,7\}$	24
7	v_3	$\{5,6,7\}$	1	$v_5 \oplus v_6 \oplus v_{10}$	$\{5,6,7\}$	47
8	v_{12}	$\{5,6,7,11\}$	127	v_5	$\{6,7\}$	1
9	v_5	$\{6,7,11\}$	1	v_6	$\{7\}$	1
10	v_6	$\{7,11\}$	1	v_7	$\{\}$	1
11	v_7	$\{11\}$	1	v_{11}	$\{\}$	1
12	v_{11}	$\{\}$	1	v_{12}	$\{\}$	7
13	v_{13}	$\{\}$	1	v_{13}	$\{\}$	1
14	v_{14}	$\{\}$	3	v_{14}	$\{\}$	3
15	v_{15}	$\{\}$	1	v_{15}	$\{\}$	1

The source code is available at `https://github.com/gtrofimiuk/SCLKernelDecoder`

[17] G. Trofimiuk and P. Trifonov, "Window Processing of Binary Polarization Kernels," in IEEE Transactions on Communications, July 2021

Arithmetic Complexity of Window Processing[18]

K_{32} kernel (constructed by algorithm[17]) with $E = 0.521936$, $\mu = 3.417$

ϕ	u_ϕ	D_ϕ	Cost	ϕ	u_ϕ	D_ϕ	Cost
0	v_0	$\{\}$	31	16	v_{18}	$\{14,15\}$	16
1	v_1	$\{\}$	1	17	$v_{14} \oplus v_{19}$	$\{14,15\}$	15
2	v_2	$\{\}$	3	18	v_{14}	$\{15\}$	1
3	v_3	$\{\}$	1	19	v_{15}	$\{\}$	1
4	v_4	$\{\}$	7	20	v_{20}	$\{\}$	7
5	v_8	$\{5,6,7\}$	67	21	v_{24}	$\{21,22,23\}$	67
6	$v_5 \oplus v_6 \oplus v_9$	$\{5,6,7\}$	24	22	$v_{21} \oplus v_{22} \oplus v_{25}$	$\{21,22,23\}$	24
7	$v_5 \oplus v_{10}$	$\{5,6,7\}$	47	23	$v_{21} \oplus v_{26}$	$\{21,22,23\}$	47
8	v_5	$\{6,7\}$	1	24	v_{21}	$\{22,23\}$	1
9	v_6	$\{7\}$	1	25	v_{22}	$\{23\}$	1
10	v_7	$\{\}$	1	26	v_{23}	$\{\}$	1
11	v_{11}	$\{\}$	1	27	v_{27}	$\{\}$	1
12	v_{16}	$\{12,13,14,15\}$	127	28	v_{28}	$\{\}$	7
13	$v_{12} \oplus v_{17}$	$\{12,13,14,15\}$	63	29	v_{29}	$\{\}$	1
14	v_{12}	$\{13,14,15\}$	1	30	v_{30}	$\{\}$	3
15	v_{13}	$\{14,15\}$	1	31	v_{31}	$\{\}$	1

The source code is available at `https://github.com/gtrofimiuk/SCLKernelDecoder`

[17] G. Trofimiuk and P. Trifonov, "Construction of binary polarization kernels for low complexity window processing," 2019 IEEE Information Theory Workshop 2019

[18] G. Trofimiuk and P. Trifonov, "Window Processing of Binary Polarization Kernels," in IEEE Transactions on Communications, July 2021

1. Motivation
   - Arikan polar codes and their limitations
   - What is possible with large kernels?
2. Decoding polar codes with large kernels
   - Successive cancellation decoding
   - Kernel processing (marginalization)
   - Trellis representation of linear codes
   - Recursive trellis processing
   - Window processing
3. Design of polar codes
   - Finding good polarization kernels
   - Code design for the BEC
   - Code design for the AWGN channel
   - Codes with Improved Distance Properties
4. Conclusions

Polarization Properties

Arikan kernel K_2

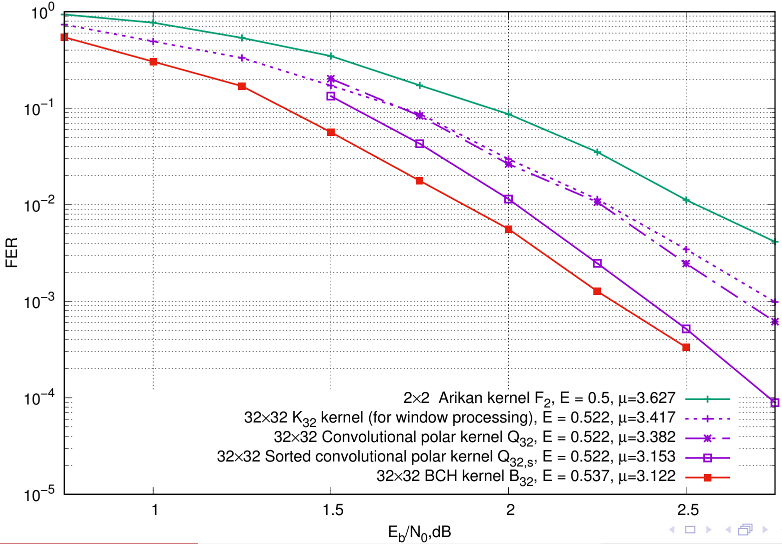
- Rate of polarization: $E(K_2) = 0.5$
- Scaling exponent: $\mu(BEC, K_2) = 3.627$

Asymptotic results

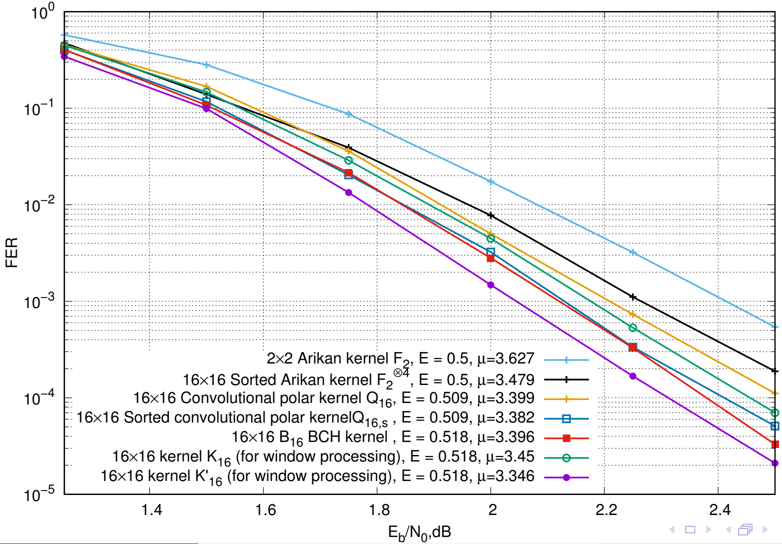
- There exist $l \times l$ kernels K_l with rate of polarization $E(K_l) \xrightarrow{l\to\infty} 1$
- There exist kernels with scaling exponent $\mu(BEC, K_l) \xrightarrow{l\to\infty} 2$

Our goal is to obtain kernels of different lengths with good polarization properties

Performance of (1024, 512) Polar Codes with Different Kernels



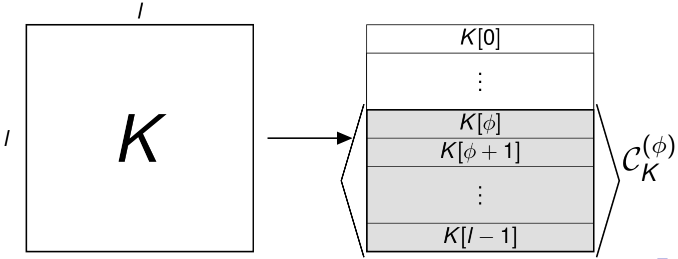
Performance of $(4096, 2048)$ Polar Codes with Different Kernels



Kernel Codes

Consider $l \times l$ kernel K

- $\langle g_1, g_2, \dots, g_k \rangle$ is a linear block code generated by the vectors $g_1, g_2, \dots, g_k$
- $K[i]$ is an i-th row of a matrix K
- $[l]$ denotes the set of n integers $\{0, 1, \dots, l-1\}$
- Let $\mathcal{C}_K^{(\phi)} = \langle K[\phi], \dots, K[l-1] \rangle$, $\phi \in [l]$, be an $(l, l-\phi, d_K^{(\phi)})$ *kernel code*
- $\mathcal{C}_K^{(l)}$ contains only zero codeword

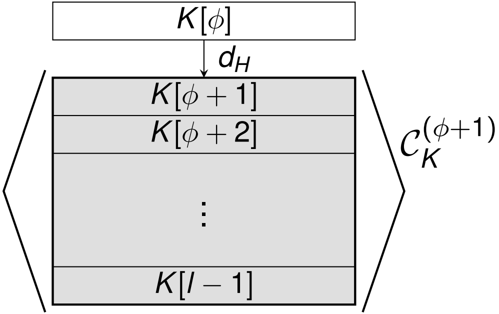


Computing the Rate of Polarization

- $d_H(a,b)$ is a Hamming distance between the vectors a and b
- $d_H(b,\mathcal{C}) = \min_{c\in\mathcal{C}} d_H(b,c)$ is a minimal distance between vector b and code $\mathcal{C}$

Partial Distances (PD)

$$\mathcal{D}_i = d_H(K[\phi], \mathcal{C}_K^{(\phi+1)}), \phi \in [l-1],$$
$$\mathcal{D}_{l-1} = d_H(K[l-1], 0)$$



Rate of Polarization:

$$E(K) = \frac{1}{l}\sum_{i=0}^{l-1}\log_l \mathcal{D}_i$$

The vector $\mathcal{D}$ is referred to as a *partial distance profile* (PDP)

Some Methods of Kernel Construction

l	Exhaustive search[19]		Shortened BCH kernels[20]	Code decomposition[21]
	E	μ	E	E
17	0.49361	3.573	0.49175	
18	0.50052	3.528	0.48968	0.49521
19	0.50054	3.444	0.48742	0.49045
20	0.50617	3.439	0.49659	
21	0.50868	3.374	0.48705	0.49604
22	0.51181	3.353	0.49445	0.50118
23	0.51213	3.372	0.50071	0.50705
24	0.51577	3.3113	0.50445	0.51577
25	0.51683	3.281	0.50040	0.50608
26	0.51921	3.256	0.50470	
27	0.51935	3.278	0.50836	

[19] G. Trofimiuk, "A Search Method for Large Polarization Kernels," 2021 IEEE International Symposium on Information Theory (ISIT), 2021

[20] S. B. Korada, E. Sasoglu, and R. Urbanke, "Polar codes: Characterization of exponent, bounds, and constructions," IEEE Trans. on Inf. Th., 2010

[21] N. Presman, O. Shapira, S. Litsyn, T. Etzion, A. Vardy, Binary polarization kernels from code decompositions, IEEE Trans. On Inf. Th., vol. 61, no. 5, 2015

Motivation of the Exhaustive Search Algorith

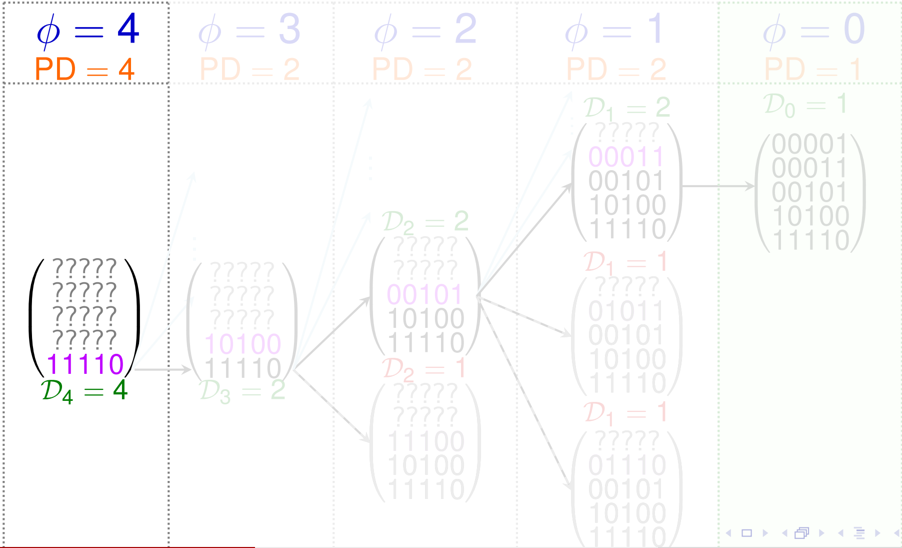
How to obtain kernels of size l with the best polarization properties?

1. Polarization properties
   - Rate of polarization E
   - Scaling exponent μ
2. Rate of polarization
   - Independent of channel
   - Can be *explicitly* computed

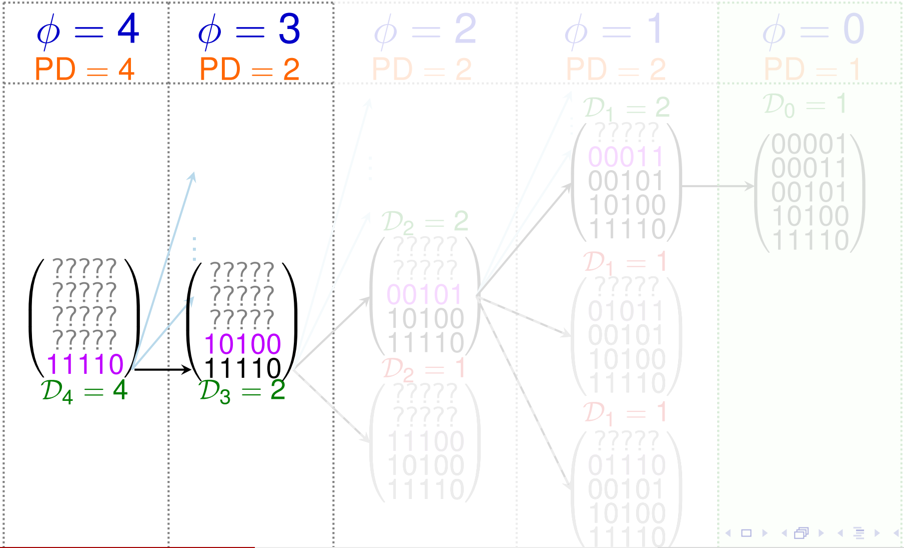
E depends on partial distances only

- We can search for $l \times l$ kernels with given partial distance profile

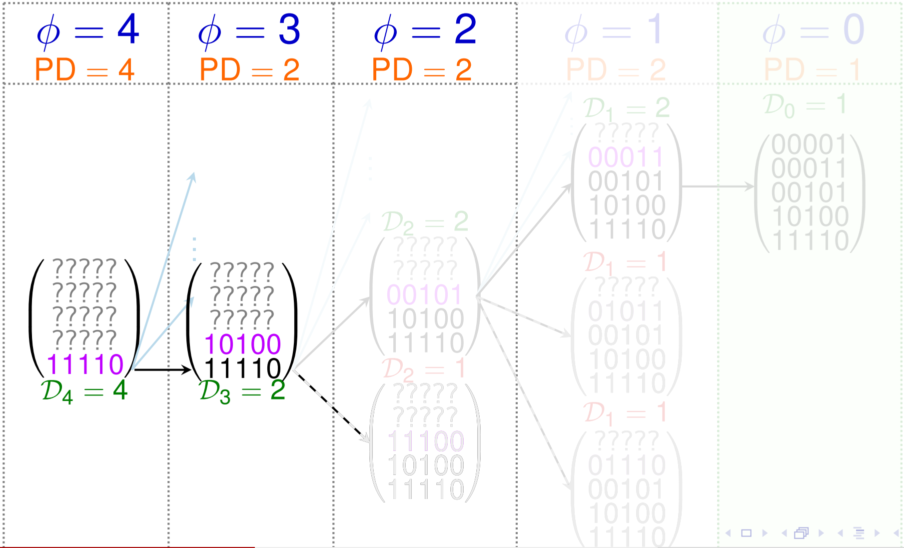
Example: 5×5 Kernel with Partial Distances $[1, 2, 2, 2, 4]$



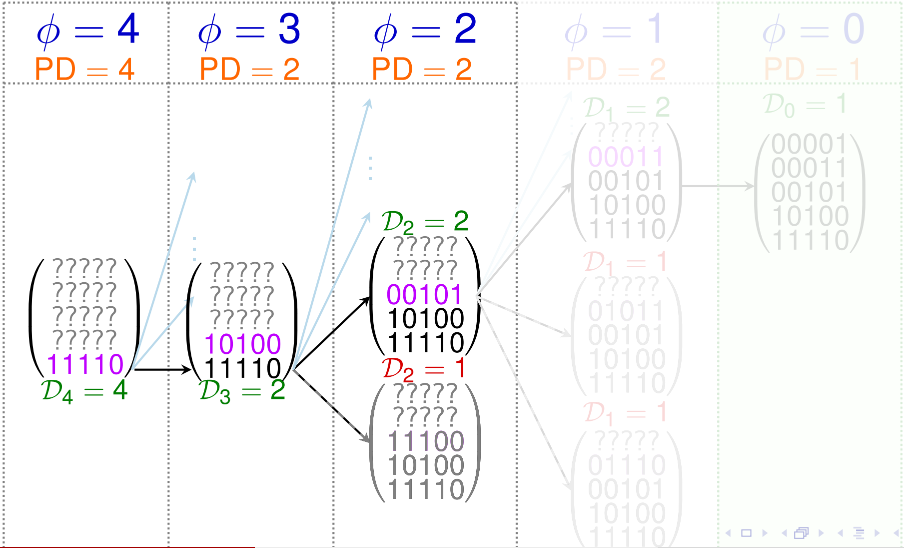
Example: 5×5 Kernel with Partial Distances $[1, 2, 2, 2, 4]$



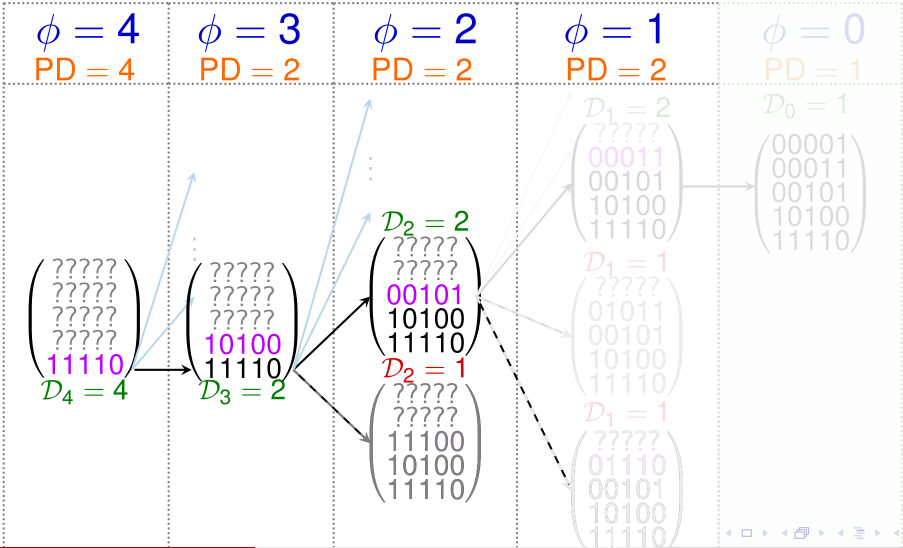
Example: 5×5 Kernel with Partial Distances $[1, 2, 2, 2, 4]$



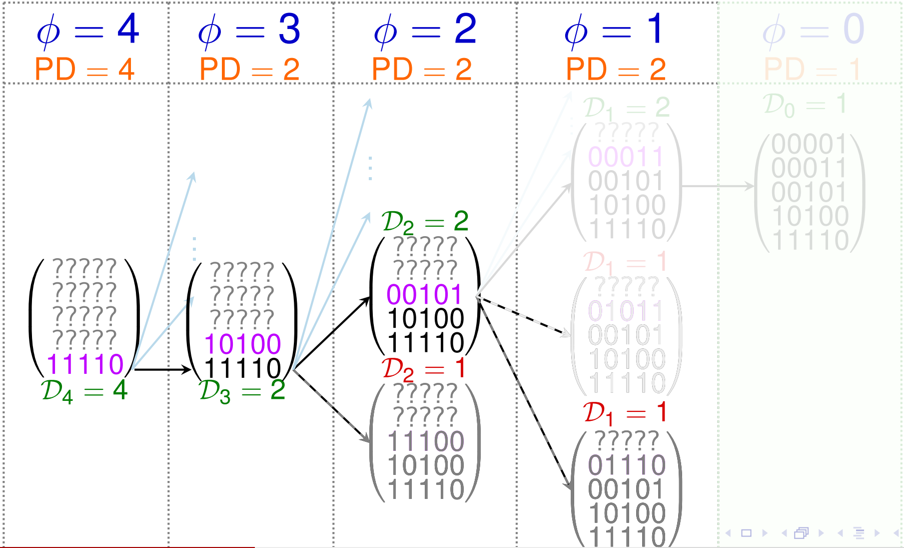
Example: 5×5 Kernel with Partial Distances $[1, 2, 2, 2, 4]$



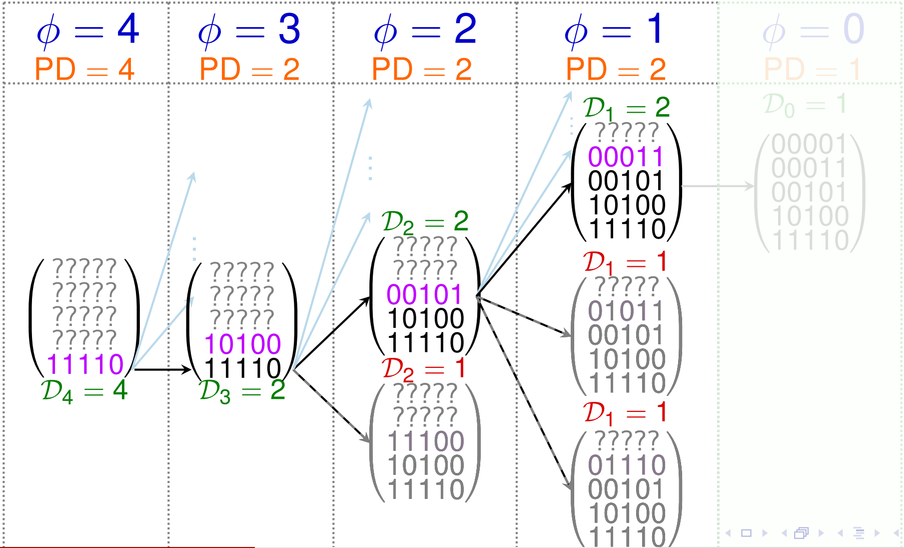
Example: 5×5 Kernel with Partial Distances $[1, 2, 2, 2, 4]$



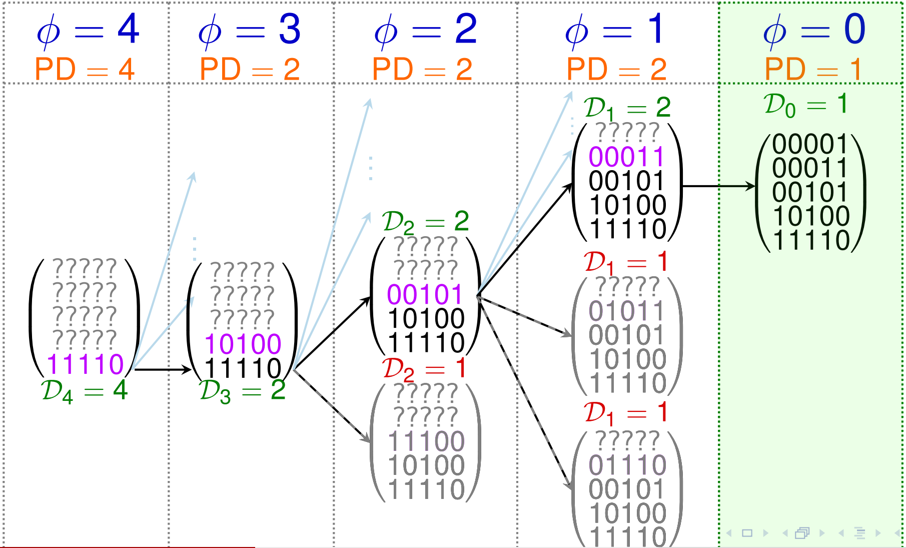
Example: 5×5 Kernel with Partial Distances $[1, 2, 2, 2, 4]$



Example: 5×5 Kernel with Partial Distances $[1, 2, 2, 2, 4]$



Example: 5×5 Kernel with Partial Distances $[1, 2, 2, 2, 4]$



The Basic Algorithm

Algorithm 1: $\texttt{BasicKernelSearch}(K, \phi, \mathbb{M}, \mathcal{D})$

if $\phi = -1$ **then**
  return K;
for ***each*** $v \in \mathbb{M}_\phi$ **do**
  $d \leftarrow d_H(v, \mathcal{C}_K^{(\phi+1)})$;
  if $d = \mathcal{D}_\phi$ **then**
    $K[\phi] \leftarrow v$;
    $\widehat{K} \leftarrow \texttt{BasicKernelSearch}(K, \phi - 1, \mathbb{M}, \mathcal{D})$;
    if $\widehat{K} \neq \mathbf{0}^{l \times l}$ **then**
      return $\widehat{K}$;
return $\mathbf{0}^{l \times l}$;

- $\mathbb{M}_\phi$ is a set of candidate rows
- For each returned K we have $K[\phi] \in \mathbb{M}_\phi$

Algorithm 1 is a **depth-first search** over candidate rows

- Default candidate rows $\mathbb{M}_\phi^{(def)} = \left\{ v_0^{l-1} | v_0^{l-1} \in \mathbb{F}_2^l, \text{wt}(v) \geq \mathcal{D}_\phi \right\}$

Can be restricted to

$$\mathbb{M}_\phi^{(r)} = \left\{ v_0^{l-1} | v_0^{l-1} \in \mathbb{F}_2^l, \text{wt}(v) = \mathcal{D}_\phi \right\}$$

Bounds on Partial Distances

- The minimum distance $d_K^{(\phi)}$ of kernel codes $\mathcal{C}_K^{(\phi)}$ is given by $\min_{\phi \le i < l} \mathcal{D}_i$

Consider $l \times l$ kernel K with nondecreasing PDP $\mathcal{D}$, such as $\mathcal{D}_\phi \le \mathcal{D}_{\phi+1}$ for $\phi \in [l-1]$

- Let $d[n,k]$ is a best known minimum distance of (n,k) code. Thus, $\mathcal{D}_\phi \le d[l, l-\phi]$

Let $\mathcal{D}$ be a nondecreasing PDP

- If $\mathcal{D}_1 = 2$, then[22] $\mathcal{D}_i$ is even for all $i \ge 1$;
- For $0 \le i < l$, we have[22]

$$\sum_{i'=i}^{l} 2^{l-i'} \mathcal{D}_{i'} \le 2^{l-i} l$$

[22] H.P. Lin, S. Lin, and K. A. Abdel-Ghaffar, Linear and nonlinear binary kernels of polar codes of small dimensions with maximum exponents, IEEE Transactions On Information Theory, vol. 61, no. 10, 2015

Construction of Good Polarization Kernels

1. Generate various non-decreasing PDPs (satisfying the above bounds)
2. For each PDP run the exhaustive search algorithm
   - Unfortunately, it is hard to determine sufficient running time
   - Typically, if kernels with a given PDP exists, then depth-first search finds a corresponding kernel quickly
3. Pick a kernel with the best polarization properties

Kernels with Good Polarization Properties

The processing complexity is measured as a number of addition and comparison operations

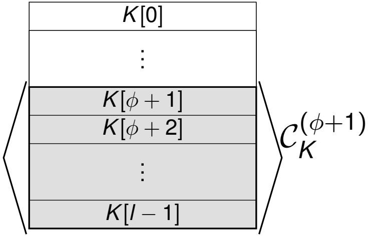
l	E	μ	Partial distances	Recursive trellis processing complexity
17	0.49361	3.573	1, 1, 2, 2, 2, 3, 4, 4, 4, 5, 6, 7, 8, 8, 8, 8, 16	1250
18	0.50052	3.528	1, 2, 2, 2, 2, 2, 4, 4, 4, 6, 6, 6, 6, 8, 8, 10, 10, 12	2946
19	0.50054	3.444	1, 2, 2, 2, 2, 2, 4, 4, 4, 4, 6, 6, 6, 8, 8, 8, 10, 10, 16	5048
20	0.50617	3.439	1, 2, 2, 2, 2, 2, 4, 4, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 12, 16	4894
21	0.50868	3.374	1, 2, 2, 2, 2, 2, 4, 4, 4, 4, 6, 6, 6, 6, 8, 8, 10, 10, 10, 14, 14	10978
22	0.51181	3.353	1, 2, 2, 2, 2, 2, 4, 4, 4, 4, 6, 6, 6, 6, 8, 8, 8, 10, 10, 10, 12, 20	18120
23	0.51213	3.372	1, 2, 2, 2, 2, 2, 4, 4, 4, 4, 6, 6, 6, 6, 6, 8, 8, 10, 10, 10, 12, 14, 16	17786
24	0.51577	3.3113	1, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8, 12, 12, 12, 16, 16	2828
25	0.51683	3.281	1, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 6, 6, 6, 8, 8, 8, 8, 8, 10, 12, 12, 12, 16, 18	40566
26	0.51921	3.256	1, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 6, 6, 6, 6, 8, 8, 8, 10, 10, 10, 12, 12, 12, 14, 24	54848
27	0.51935	3.278	1, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 6, 6, 6, 6, 8, 8, 8, 8, 10, 10, 10, 12, 12, 14, 14, 24	93764

All obtained kernels are available at `arxiv.org/abs/2101.10269`

- The processing complexity is too high, how to reduce it?

Kernel Processing

Computing the bit subchannel probability $\mathcal{W}_1^{(\phi)}(u_0^\phi|y_0^{l-1})$ for one layer of $l \times l$ transform K is equal to **ML decoding** of the kernel code $\mathcal{C}_K^{(\phi+1)}$ and its coset $\mathcal{C}_K^{(\phi+1)} \oplus K[\phi]$



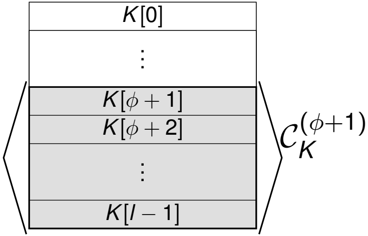
- ML decoding can be done by Viterbi algorithm or recursive trellis decoding
- The minimum distance $d_K^{(\phi)}$ of kernel codes $\mathcal{C}_K^{(\phi)}$ is given by $\min_{\phi \le i < l} \mathcal{D}_i$
- Trellis state complexity is lower bounded by the minimum distance of the code

To obtain polarization kernels which admit low processing complexity, we can reduce the minimum distance of the kernel codes

- As a consequence, we obtain kernels with the degraded polarization properties

Kernel Processing

Computing the bit subchannel probability $\mathcal{W}_1^{(\phi)}(u_0^{\phi}|y_0^{l-1})$ for one layer of $l \times l$ transform K is equal to **ML decoding** of the kernel code $\mathcal{C}_K^{(\phi+1)}$ and its coset $\mathcal{C}_K^{(\phi+1)} \oplus K[\phi]$



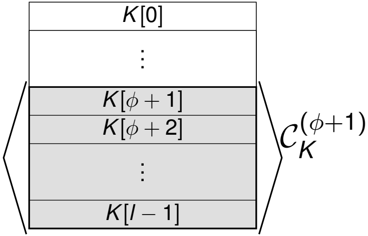
- ML decoding can be done by Viterbi algorithm or recursive trellis decoding
- The minimum distance $d_K^{(\phi)}$ of kernel codes $\mathcal{C}_K^{(\phi)}$ is given by $\min_{\phi \leq i < l} \mathcal{D}_i$
- Trellis state complexity is lower bounded by the minimum distance of the code

To obtain polarization kernels which admit low processing complexity, we can reduce the minimum distance of the kernel codes

- As a consequence, we obtain kernels with the degraded polarization properties

Kernel Processing

Computing the bit subchannel probability $\mathcal{W}_1^{(\phi)}(u_0^\phi|y_0^{l-1})$ for one layer of $l \times l$ transform K is equal to **ML decoding** of the kernel code $\mathcal{C}_K^{(\phi+1)}$ and its coset $\mathcal{C}_K^{(\phi+1)} \oplus K[\phi]$



- ML decoding can be done by Viterbi algorithm or recursive trellis decoding
- The minimum distance $d_K^{(\phi)}$ of kernel codes $\mathcal{C}_K^{(\phi)}$ is given by $\min\limits_{\phi \le i < l} \mathcal{D}_i$
- Trellis state complexity is lower bounded by the minimum distance of the code

To obtain polarization kernels which admit low processing complexity, we can reduce the minimum distance of the kernel codes

- As a consequence, we obtain kernels with the degraded polarization properties

Bounds on Trellis Complexity of Linear Codes

Let $\mathcal{C}$ be an (n, k, d) linear code over $\mathbb{F}_q$. Then the nonsectionalized state complexity satisfies [23]

$$s \geq \max_i (k - K(i, d) - K(n - i, d)), \tag{3}$$

where $K(i, d)$ is the maximum possible dimension of the code of length i and minimum distance d over $\mathbb{F}_q$

For (n, k, d) linear code $\mathcal{C}$ [24]

$$s \geq \left\lceil \frac{k(d-1)}{n} \right\rceil \tag{4}$$

[23] D. J. Muder, "Minimal trellises for block codes," in IEEE Transactions on Information Theory, vol. 34, no. 5, pp. 1049-1053, Sept. 1988

[24] A. Lafourcade and A. Vardy, "Asymptotically good codes have infinite trellis complexity," in IEEE Transactions on Information Theory, March 1995

Example with 16×16 Kernels

Kernel	E	μ	Distance properties																	Recursive trellis processing complexity
K'_{16}	0.51828	3.346	PDP	1	2	2	2	2	4	4	4	4	6	6	8	8	8	8	16	632
			$d_k^{(\phi)}$	1	2	2	2	2	4	4	4	4	6	6	8	8	8	8	16	
K_{16}	0.51828	3.45	PDP	1	2	2	4	2	2	4	4	6	6	8	8	4	8	8	16	236
			$d_k^{(\phi)}$	1	2	2	2	2	**2**	4	4	4	**4**	**4**	**4**	**4**	8	8	16	

Kernel	E	μ	Distance properties																	Recursive trellis processing complexity
$F_2^{\otimes 4}$	0.5	3.627	PDP	1	2	2	4	2	4	4	8	2	4	4	8	4	8	8	16	64
			$d_k^{(\phi)}$	1	2	2	2	2	2	2	2	2	4	4	4	4	8	8	16	
$F_2^{\otimes 4}$, sorted	0.5	3.479	PDP	1	2	2	2	2	4	4	4	4	4	4	8	8	8	8	16	328
			$d_k^{(\phi)}$	1	2	2	2	2	**4**	**4**	**4**	**4**	4	4	**8**	**8**	8	8	16	

Kernels which Admit Low Complexity Processing

1. Generate several PDPs corresponding to the *reduced rate of polarization* (compared to the best one)
2. *Permute* some PDP entries
3. Try to obtain the corresponding kernels by the exhaustive search algorithm

l	E	μ	Partial distances	Recursive trellis processing complexity
18	0.50052	3.528	1, 2, 2, 2, 2, 2, 4, 4, 4, 6, 6, 6, 6, 8, 8, 10, 10, 12	2946
18	0.49521	3.648	1, 2, 2, 4, 2, 2, 2, 4, 4, 6, 4, 6, 8, 8, 8, 8, 8, 16	1000
20	0.50617	3.439	1, 2, 2, 2, 2, 2, 4, 4, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 12, 16	4894
20	0.49943	3.649	1, 2, 2, 4, 2, 4, 2, 2, 4, 4, 6, 8, 8, 8, 4, 8, 12, 8, 8, 16	478
24	0.51577	3.3113	1, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8, 12, 12, 12, 16, 16	2828
24	0.50291	3.619	1, 2, 2, 4, 2, 4, 2, 4, 6, 2, 4, 4, 8, 8, 12, 4, 4, 8, 8, 12, 12, 8, 16, 16	365
27	0.51935	3.278	1, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 6, 6, 6, 6, 8, 8, 8, 8, 10, 10, 10, 12, 12, 14, 14, 24	93764
27	0.49720	3.766	1, 2, 2, 4, 2, 2, 4, 4, 6, 2, 4, 4, 6, 6, 8, 8, 10, 12, 4, 4, 8, 8, 12, 12, 8, 16, 16	998

There is a trade-off between processing complexity and polarization properties

Design of Polar Codes with Large Kernels

Method	Arikan kernel	Large kernels
Monte-Carlo	✓	✓
Binary erasure channel recursion	✓	✓
Gaussian approximation	✓	✓
Minimum distance	✓	✓
Density evolution	✓	✗
Degrading/upgrading approximation	✓	✗
Partial order	✓	✗
Polarization weight	✓	✗
Polar codes with CRC	✓	✓
Polar subcodes	✓	✓
Polar codes with distributed CRC	✓	✗
Shortening and puncturing	✓	✗ and ✓

✗: the method has not yet been developed

Polarization Behaviour

- If $W(y|c)$ is the binary erasure channel (BEC), then $W_m^{(i)}$ are also BEC
- Input erasure pattern $e \in \mathbb{F}_2^l$ at phase i is not correctable iff there exist $u_{i+1}^{l-1}, v_{i+1}^{l-1} : \forall u_0^{i-1}, \forall j : e_j = 0 : ((u_0^{i-1}, 0, u_{i+1}^{l-1})K)_j = ((u_0^{i-1}, 1, v_{i+1}^{l-1})K)_j$
- Let $E_{i,w}$ be the number of uncorrectable erasure patterns of weight w
- Let z be the input erasure probability. Erasure probability in $W_1^{(i)}$ is

$$f_i(z) = \sum_{w=0}^{l} E_{i,w} z^w (1-z)^{l-w}$$

- Polarization behaviour is the collection of functions $(f_0(z), \ldots, f_{l-1}(z))$

An Example

- Arikan kernel $K_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$
- Uncorrectable erasure patterns for phase 0:(10), (01), (11)

$$f_0(z) = 2z(1-z) + z^2 = 2z - z^2$$

- Uncorrectable erasure patterns for phase 1: (11)

$$f_1(z) = z^2$$

Finding the Polarization Behaviour

- If erasure pattern $e \in \{0,1\}^l$ is uncorrectable, then any $e' \in \{0,1\}^l$ which covers e, i.e $e'_i \geq e_i$, is also uncorrectable
- Let cover set $\Delta(S)$ be the set of vectors that cover at least one vector in set S
- Let $C_K^{(i)} = \langle K_{i..l-1} \rangle$ be the i-th kernel code, $C_K^{(l)} = \{0\}$.
- An erasure pattern e is uncorrectable at phase i iff $e \in \Delta(C_K^{(i)} \setminus C_K^{(i+1)}), 0 \leq i < l$
- The number of uncorrectable erasure patterns can be obtained from the trellis[25][26] of $C_K^{(i)}$

[25] V. Miloslavskaya, P. Trifonov. Design of binary polar codes with arbitrary kernel. In Proc of IEEE Information Theory Workshop , 2012

[26] H. Yao, A. Fazeli, A.Vardy. Explicit Polar Codes with Small Scaling Exponent. In Proc. of ISIT 2019.

Capacity of Bit Subchannels

- Capacity of a binary input symmetric channel

$$I_1^{(i)} = 1 - \int_{-\infty}^{\infty} p_i(\xi)\log_2(1+e^{-\xi})d\xi,$$

where $p_i(\xi|0)$ is the PDF[27] of the LLR $\ln\frac{W_1^{(i)}(y_0^{l-1},\mathbf{0}|0)}{W_1^{(i)}(y_0^{l-1},\mathbf{0}|1)}$, assuming that 0 is transmitted

- An approximation

$$\begin{aligned} I_1^{(i)} \approx& 1 - \int_{-\infty}^{\infty} f_i(\xi|0)\log_2\left(1+\frac{P\left\{u_i=1|S_1^{(i)}=\xi\right\}}{P\left\{u_i=0|S_1^{(i)}=\xi\right\}}\right)d\xi \\ =& 1 - \int_{-\infty}^{\infty} f_i(\xi|0)\log_2\left(1+\frac{f_i(-\xi|0)}{f_i(\xi|0)}\right)d\xi, \end{aligned}$$

where $f_i(\xi|0)$ is the PDF of $S_1^{(i)}(\mathbf{0}, y_0^{l-1})$, assuming that 0 is transmitted

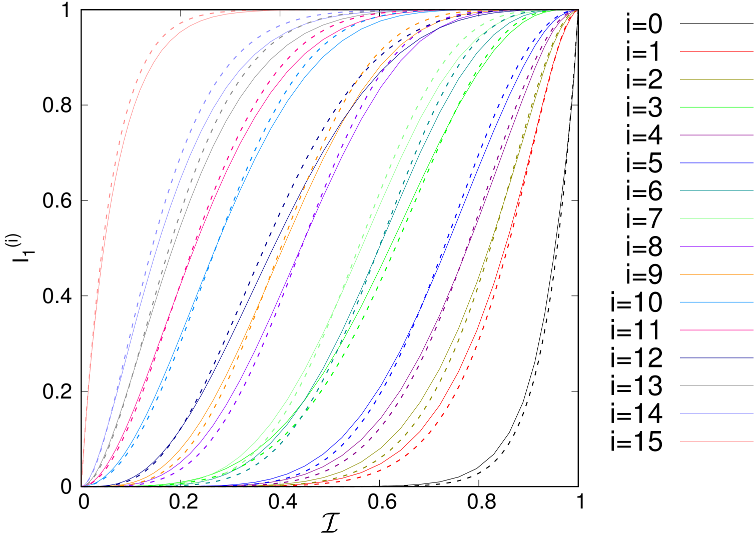
[27] T. Richardson, R. Urbanke. Modern coding theory. Cambridge University Press, 2008

Mutual Information Based Design

- Let $\mathcal{I} = I_0(X; Y)$ be the mutual information (symmetric capacity) of AWGN channel input X and output Y
- Simulations can be use to estimate PDF $f_i(\xi|0)$ and compute mutual information $I_1^{(i)}(\mathcal{I})$ of $W_1^{(i)}$ for any $\mathcal{I} : 0 < \mathcal{I} < 1$
- Assume that all bit subchannels are Gaussian, and they are completely characterized by their symmetric capacity

$$I_m^{(lj+s)}(\mathcal{I}) \approx \begin{cases} I_1^{(s)}(I_{m-1}^{(j)}(\mathcal{I})), & m > 0, 0 \le s < l, 0 \le j < l^{m-1} \\ \mathcal{I}, & m = 0, s = 0 \end{cases}$$

Subchannel Capacity Functions: $K_{16}, \mu = 3.45$



Capacity functions for BEC (dashed) are not identical to those for AWGN channel (solid)

Polar Codes with CRC

- Append CRC to the data before encoding it with a large kernel polar code
- Use Tal-Vardy list decoder
- Select a codeword with the valid CRC from the obtained list

Minimum Weight Codewords of Polar Codes

Partial distance D_j is the distance from the j-th row of kernel K to the code generated by rows $j+1,\ldots,l-1$

Theorem

Consider a (n,k,d) polar code $\mathcal{C}$ given by a polarizing transformation $A = K^{\otimes m}$ and a frozen set $\mathcal{K}$, where $n = l^m$, and partial distances D_i of kernel K satisfy

$$D_i = \mathrm{wt}(K[j]), 0 \le j < l_i, 0 \le i < m.$$

Then:

1. $d = \min_{i \notin \mathcal{F}} \mathrm{wt}(A_i)$, *where A_i is the i-th row of matrix A.*
2. *For any* $c_0^{n-1} = u_0^{n-1} A \in \mathcal{C} : \mathrm{wt}(c_0^{n-1}) = d\ \exists i : u_i = 1, \mathrm{wt}(A_i) = d$.

Example: for Arikan kernel one has $\mathrm{wt}(A_i) = 2^{\mathrm{wt}(i)}$

Dynamic Frozen Symbols

- Classical polar codes: frozen symbols $u_i = 0, i \in \mathcal{F}$
- A generalization: $u_i = \sum_{j=0}^{i-1} V_{s_i,j} u_j, i \in \mathcal{F}$

$$uV^T = 0,$$

  where s_i is the index of row of V having last 1 in position i
- The successive cancellation decoding algorithm:
  For $i = 0, 1, \ldots, 2^m - 1$:

$$\widehat{u}_i = \begin{cases} \sum_{j=0}^{i-1} V_{s_i,j} \widehat{u}_j, & i \in \mathcal{K} \\ \arg\max_{u_i} \mathcal{W}_m^{(i)}(y_0^{n-1}, \widehat{u}_0^{i-1} | u_i), & i \notin \mathcal{F} \end{cases}$$

- Decoding error probability $P_{SC} \leq \sum_{i \notin \mathcal{F}} P_{m,i}$
  - The same as for a classical polar code with frozen set $\mathcal{F}$
- Straightforward extension to list SC (Tal-Vardy) decoding
- How to select the constraint matrix V?

Reducing the Error Coefficient

- Consider a random linear k-dimensional subcode $\mathcal{C}$ of a *base* (n, k') polar code, $k < k'$. Let $(w_0, \dots, w_n)$ be its weight spectrum
- If all linear subspaces are equiprobable, then

$$\mathbf{E}[w_s] = w_s' \frac{2^k - 1}{2^{k'} - 1} \approx w_s' 2^{-(k'-k)}, s > 0,$$

- Select k' so that $\mathbf{E}[w_d]$ is sufficiently low
- Any codeword of $\mathcal{C}$ satisfies $c_0^{n-1} = u_0^{n-1} A_m$, where $u_0^{n-1} V^T = 0$
  - The constraint matrix $V = \begin{pmatrix} V' \\ \widetilde{V} \end{pmatrix}$
  - V' is an $(n - k') \times n$ matrix with distinct weight-1 rows, having 1's in positions $\mathcal{F}'$
  - $\widetilde{V}$ is a random $(k' - k) \times n$ full-rank matrix
- Polar-CRC codes: $\widetilde{V}$ is a check matrix of the CRC code

Reducing the Error Coefficient

- Consider a random linear k-dimensional subcode $\mathcal{C}$ of a *base* (n, k') polar code, $k < k'$. Let $(w_0, \dots, w_n)$ be its weight spectrum
- If all linear subspaces are equiprobable, then

$$\mathbf{E}[w_s] = w'_s \frac{2^k - 1}{2^{k'} - 1} \approx w'_s 2^{-(k'-k)}, s > 0,$$

- Select k' so that $\mathbf{E}[w_d]$ is sufficiently low
- Any codeword of $\mathcal{C}$ satisfies $c_0^{n-1} = u_0^{n-1} A_m$, where $u_0^{n-1} V^T = 0$
  - The constraint matrix $V = \begin{pmatrix} V' \\ \widetilde{V} \end{pmatrix}$
  - V' is an $(n - k') \times n$ matrix with distinct weight-1 rows, having 1's in positions $\mathcal{F}'$
  - $\widetilde{V}$ is a random $(k' - k) \times n$ full-rank matrix
- Polar-CRC codes: $\widetilde{V}$ is a check matrix of the CRC code

Type-A Dynamic Frozen Symbols

- For truly random subcodes i_j are too high $\Rightarrow$ the decoder may kill the correct path before it is able to exploit the dynamic freezing constraints
- $\widetilde{V}$ is responsible for elimination of low-weight codewords
- Construct $\widetilde{V}$, so that
  - the decoder can process dynamic freezing constraints as soon as possible
  - most of the low-weight codewords are still eliminated
- Let the indices of non-trivial dynamic frozen symbols be smallest possible, such that all $u_i : \mathrm{wt}(A_{m,i}) = d = \min_{i \notin \mathcal{F}} \mathrm{wt}(A_{m,i})$ are involved in at least one dynamic freezing constraint
- Set $V_{sj}, n-k' \leq s < n-k, 0 \leq j < i_s$, to independent equiprobable binary values

Type-A Dynamic Frozen Symbols

- For truly random subcodes i_j are too high $\Rightarrow$ the decoder may kill the correct path before it is able to exploit the dynamic freezing constraints
- $\widetilde{V}$ is responsible for elimination of low-weight codewords
- Construct $\widetilde{V}$, so that
  - the decoder can process dynamic freezing constraints as soon as possible
  - most of the low-weight codewords are still eliminated
- Let the indices of non-trivial dynamic frozen symbols be smallest possible, such that all $u_i : \mathrm{wt}(A_{m,i}) = d = \min_{i \notin \mathcal{F}} \mathrm{wt}(A_{m,i})$ are involved in at least one dynamic freezing constraint
- Set $V_{sj}, n - k' \leq s < n - k, 0 \leq j < i_s$, to independent equiprobable binary values

Penalizing Wrong Paths in the SCL Decoder: Type-B DFS

- Path score in the SCL algorithm: $R(\hat{u}_0^j) = R(\hat{u}_0^{j-1}) + \tau(S_m^{(j)}, \hat{u}_j)$, where

$$\tau(u, S) = \begin{cases} 0, & \text{if } (-1)^u = \operatorname{sgn} S \\ -|S|, & \text{otherwise} \end{cases}$$

- Incorrect $\hat{u}_i$ result in low-magnitude $S_m^{(j)}$ for many $j > i$, and many of $S_m^{(j)}$ still have correct signs $\Rightarrow$ wrong path in the SCL decoder are penalized slowly while processing constraints $u_i = 0, i \in \mathcal{F}'$
- Type-B dynamic frozen symbols
  - Speedup error propagation for wrong paths
  - Select q most reliable bit subchannels $\mathcal{W}_m^{(i)} : i \in \mathcal{F}'$
  - Replace $u_i = 0$ with $u_i = \sum_{j<i} R_{ij} u_j$, where R_{ij} are random binary values
  - If some $\hat{u}_j$ is incorrect, $\sum_{j<i} R_{ij} \hat{u}_j$ would disagree with the sign of $S_m^{(i)}$ with high probability $\Rightarrow$ the path is penalized

Penalizing Wrong Paths in the SCL Decoder: Type-B DFS

- Path score in the SCL algorithm: $R(\hat{u}_0^j) = R(\hat{u}_0^{j-1}) + \tau(S_m^{(j)}, \hat{u}_j)$, where

$$\tau(u, S) = \begin{cases} 0, & \text{if } (-1)^u = \operatorname{sgn} S \\ -|S|, & \text{otherwise} \end{cases}$$

- Incorrect $\hat{u}_i$ result in low-magnitude $S_m^{(j)}$ for many $j > i$, and many of $S_m^{(j)}$ still have correct signs $\Rightarrow$ wrong path in the SCL decoder are penalized slowly while processing constraints $u_i = 0, i \in \mathcal{F}'$
- Type-B dynamic frozen symbols
  - Speedup error propagation for wrong paths
  - Select q most reliable bit subchannels $\mathcal{W}_m^{(i)} : i \in \mathcal{F}'$
  - Replace $u_i = 0$ with $u_i = \sum_{j<i} R_{ij} u_j$, where R_{ij} are random binary values
  - If some $\hat{u}_j$ is incorrect, $\sum_{j<i} R_{ij} \hat{u}_j$ would disagree with the sign of $S_m^{(i)}$ with high probability $\Rightarrow$ the path is penalized

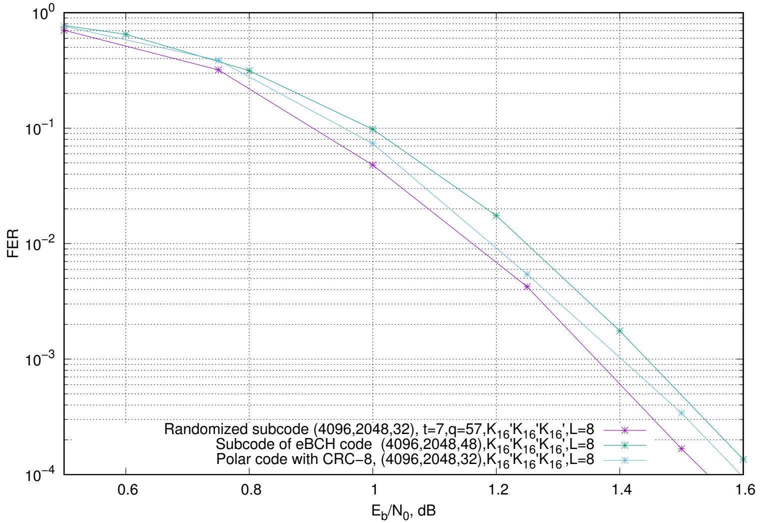
Polar Subcodes: an Algebraic Construction

Design an $(n = l^m, k, d)$ polar subcode

- Let H be a check matrix of $(n, \kappa > k, d)$ extended narrow-sense primitive BCH code
- Let $A_m = K_l^{\otimes m}$
- Let $V_0 = QHA_m^T$, where Q is an invertible matrix, such that last non-zero elements of V are located in distinct columns $j_i, 0 \leq i < n - \kappa$
- Let $\mathcal{F}_0 = \{j_0, \ldots, j_{n-\kappa-1}\}$
- Find indices $j_i \notin \mathcal{F}_0, n - \kappa \leq i < n - k$ of the least reliable bit subchannels
- Let $V = \begin{pmatrix} V_0 \\ V_1 \end{pmatrix}$, where V_1 contains distinct rows with 1's in positions $j_{n-\kappa}, \ldots, j_{n-k-1}$, and 0 elsewhere
- V is the constraint matrix of $(n, k, \geq d)$ polar subcode[28]

[28] P. Trifonov and V. Miloslavskaya, "Polar subcodes," IEEE Journal on Selected Areas in Communications, vol. 34, no. 2, February 2016

Performance of Polar Subcodes under SCL Decoding



$$\mu(K'_{16}) = 3.346$$

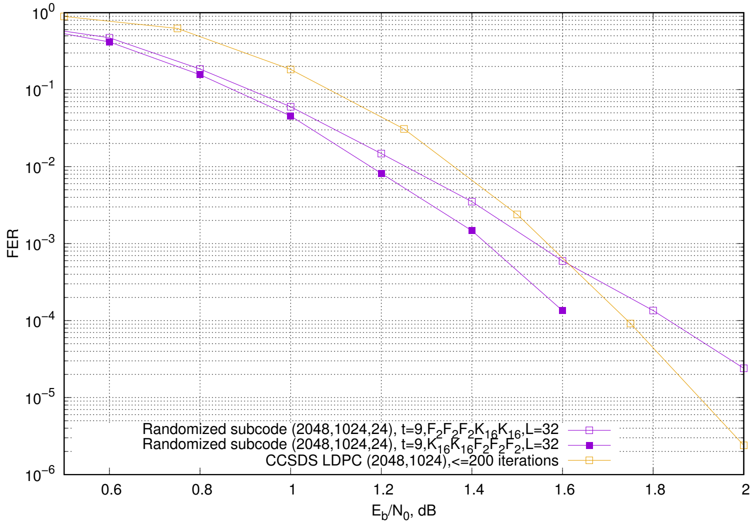
Mixed Kernel Codes

- Mixed kernel polarizing transformation $A = K_{l_1} \otimes K_{l_2} \otimes \cdots \otimes K_{l_m}$
- Rate of polarization for a Kronecker[29] product of matrices $E(A) = \sum_{l=1}^{m} \frac{E(K_{l_i})}{\log_{l_i} \prod_{i=1}^{m} l_i}$
- Kernels over different alphabets can be mixed[30]
- Changing the order of matrices in the Kronecker product does affect the performance

[29] M.-K. Lee, K. Yang. The exponent of a polarizing matrix constructed from the Kronecker product. Des. Codes Cryptogr. vol. 70, 2014

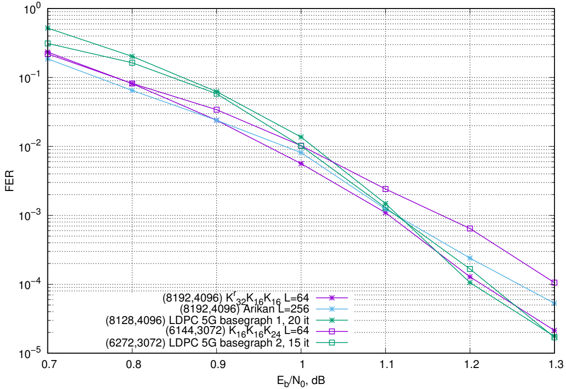
[30] N. Presman, O. Shapira, S. Litsyn. IEEE Journal On Selected Areas In Communications, Vol. 34, No. 2, February 2016

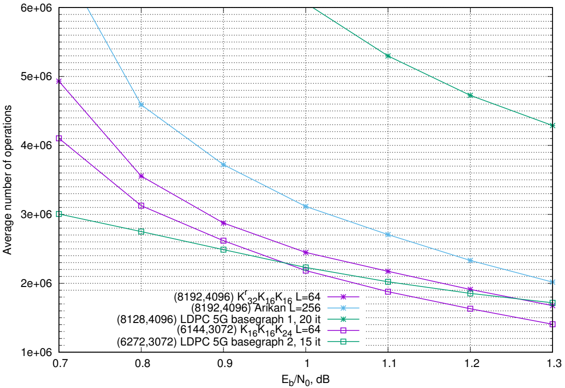
Performance of Mixed-Kernel Codes



Changing the order of kernels does affect the performance

Sequential Decoding of Large Kernel Codes vs BP Decoding of 5G LDPC Codes





- Kernel K^r_{32} has simple trellises $\Rightarrow$ decoding complexity much less than for the LDPC code
- Even K_{24} with complex trellises has some complexity advantage with respect to LDPC

Conclusions

- Polar codes with large kernels under SCL decoding can provide both performance and complexity improvement with respect to the codes based on the Arikan kernel
- The codes based on large kernels may have lower decoding complexity compared to LDPC codes with similar performance
- Efficient kernel processing is essential to obtain a practical implementation
  - Window processing
  - Recursive trellis processing
- Polarization kernels need to be carefully designed
  - Higher scaling exponent may provide simpler processing and better overall performance/complexity tradeoff
  - Length-compatible codes with common kernel processing
- Many of code design and decoding techniques developed for Arikan polar codes extend easily to the case of large kernels

Open Problems

- How to further reduce the complexity of kernel processing?
- How to explicitly construct kernels with a given rate of polarization and simple recursive trellis processing?
- How to find an optimal order of kernels in mixed kernel codes?
- How to compute scaling exponent for channels other than BEC?
- How to implement shortening and puncturing of large kernel codes?