A Practical Introduction to Polar Codes

(A very simple tutorial for beginners)

Harish Vangala, Yi Hong, and Emanuele Viterbo

Monash University, Australia

23 February, 2016

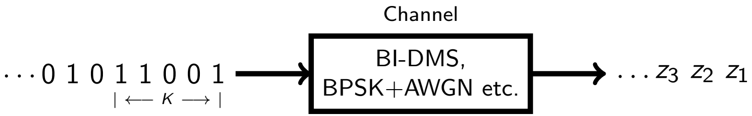
This presentation, and other useful resources such as MATLAB modules can be found here:

is.gd/polarcodes

(Or, http://www.ecse.monash.edu.au/staff/eviterbo/polarcodes.html)

Recall: The Coding Problem

Recall: The Coding Problem



Recall: The Coding Problem

Recall: The Coding Problem

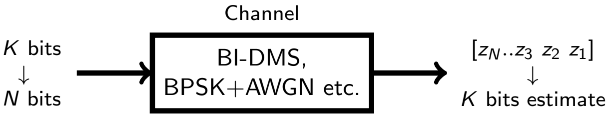
The Uncoded System

Recall: The Coding Problem

Channel

$[1\ 1\ 0\ 0\ 1]$
$\downarrow$
$[1\ 1\ 0\ 0\ 1\ 1\ 0\ 1]$ $\longrightarrow$ $\boxed{\text{BI-DMS, BPSK+AWGN etc.}}$ $\longrightarrow$ $[z_N..z_3\ z_2\ z_1]$
$\downarrow$
$[1\ 1\ 0\ 0\ 1]$

Recall: The Coding Problem

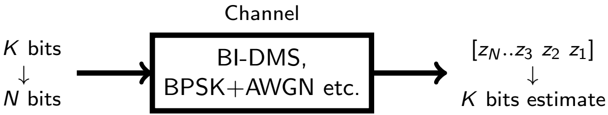


Recall: The Coding Problem

Channel

K bits $\downarrow$ N bits $\longrightarrow$ $\boxed{\text{BI-DMS, BPSK+AWGN etc.}}$ $\longrightarrow$ $[z_N..z_3\ z_2\ z_1]$ $\downarrow$ K bits estimate

The Encoding



The Encoding

The Decoding

Recall: The Coding Problem

Channel

K bits $\downarrow$ N bits $\longrightarrow$ $\boxed{\text{BI-DMS, BPSK+AWGN etc.}}$ $\longrightarrow$ $[z_N..z_3\ z_2\ z_1]$ $\downarrow$ K bits estimate

The Encoding **The Decoding**

The Coding System to achieve **Shannon capacity**

Recall: The Coding Problem

Channel

K bits $\downarrow$ N bits $\longrightarrow$ $\boxed{\text{BI-DMS, BPSK+AWGN etc.}}$ $\longrightarrow$ $[z_N..z_3\ z_2\ z_1]$ $\downarrow$ K bits estimate

The Encoding **The Decoding**

The Coding System to achieve **Shannon capacity**

- The Polar Coding System (originally for BI-DMS)

Recall: The Coding Problem

Channel

K bits
$\downarrow$
N bits

BI-DMS,
BPSK+AWGN etc.

$[z_N..z_3\ z_2\ z_1]$
$\downarrow$
K bits estimate

The Encoding

The Decoding

The Coding System to achieve **Shannon capacity**

- The Polar Coding System (originally for BI-DMS)
  1. Encoding

Recall: The Coding Problem

Channel

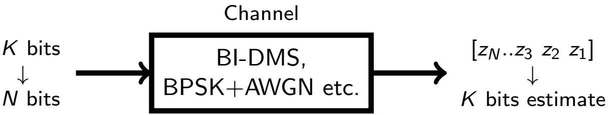
K bits $\downarrow$ N bits → BI-DMS, BPSK+AWGN etc. → $[z_N .. z_3 \ z_2 \ z_1]$ $\downarrow$ K bits estimate

The Encoding **The Decoding**

The Coding System to achieve **Shannon capacity**

- The Polar Coding System (originally for BI-DMS)
  1. Encoding
  2. Decoding

Recall: The Coding Problem



The Encoding **The Decoding**

The Coding System to achieve **Shannon capacity**

- The Polar Coding System (originally for BI-DMS)
  1. Encoding
  2. Decoding
  3. Code-construction

Polar Codes: A Brief Background

- First ever *provably* capacity achieving codes [1]
- Invented by Erdal Arıkan[2], eventually in 2009, using:

Channel Polarization

Let a BI-DMS channel with capacity $0 \leq C \leq 1$. When a codeword is Tx in N channel-uses, the channel polarization converts,

$$\left.\begin{array}{lll} \textcircled{1} & C \text{ fraction of the } N \text{ bit-channels as } \textit{noiseless} & \text{(i.e. their capacity}\approx 1) \\ \textcircled{2} & (1-C) \text{ remaining as } \textit{extremely-noisy} & \text{(i.e., their capacity}\approx 0) \end{array}\right\} \text{asymptotically, as } N \to \infty$$

- Attractive features:
  1. Fixed, low, and deterministic $\mathcal{O}(N \log_2 N)$ encoding and decoding
  2. Explicit construction
  3. Easy to implement

[1] On "Symmetric, Binary Input, and Discrete Memoryless Channels"(BI-DMS) and later extended to many other channels.

[2] Erdal Arıkan, "Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels", IEEE Trans. IT, 2009.

A simplified list of pros and cons

Advantages	Challenges
Simple encoding & decoding algo.	High $\mathcal{O}(N)$ latency
Explicit construction	Poorer performance under SCD compared to LDPC codes, at finite N
Easy to implement and high h/w efficiency	Solutions are costlier for improving performance, comparable to LDPC, at finite N
Has the best available performance under advanced decoders	
No error floors in BSC/BEC	

A Practical Introduction to Polar Codes

(A very simple tutorial for beginners)

Harish Vangala, Yi Hong, and Emanuele Viterbo

Monash University, Australia

23 February, 2016

This presentation, and other useful resources such as MATLAB modules can be found here:

is.gd/polarcodes

(Or, http://www.ecse.monash.edu.au/staff/eviterbo/polarcodes.html)

1. Code Construction
2. Encoding
3. Decoding

1.1 Construction of Polar Codes

- Simply the selection of K out of N indices — $\{0, \dots, N-1\}$, $N = 2^n$

- Many algorithms exist, the simplest is to use the recursion:
  $z \rightarrow \{2z - z^2, z^2\}$ (its use is justified in [1] and illustrated next)

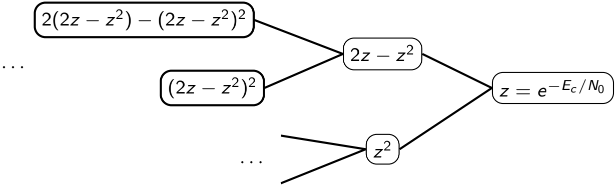
- The channel:

  > *Additive White Gaussian Channel* (AWGN)
  > with zero-mean and variance $\frac{N_0}{2}$.
  > (Used for the purposes of illustrations here throughout)

- With modest changes, the following discussion holds for any commonly used channel such as BEC, BSC etc.

[1] H. Vangala, Y. Hong, and E. Viterbo, "A Comparitive Study of Polar Code Constructions for the AWGN channel", arXiv:1501.02473, 2015

1.1 Construction of Polar Codes [contd.



1. STOP when the tree has N leaves, indexed from top $0, \ldots, N-1$
2. Find the leaves holding K least values, let their indices be $\mathcal{J}$,
3. Output $\mathcal{J}$

Notes:

1. The initial z is the Bhattacharyya parameter of the AWGN. Under the BPSK modulation of $\{\pm\sqrt{E_c}\}$, and noise-variance $N_0/2$,
$$z = \exp\left(-E_c/N_0\right)$$

1.2 The code varies with SNR and diff. constructions!

1.
- A very important characteristic of polar codes — *The non-universality*
- Code can change significantly with different choices of *design-SNR*s
- The choice of a good design-SNR is very important [1]

2.
- More accurate construction algorithms exist in many
- The best achievable performance is approx. same for any construction algorithm for at least until $N \leq 64K$ [1]

[1] H. Vangala, Y. Hong, and E. Viterbo, "A Comparitive Study of Polar Code Constructions for the AWGN channel", arXiv:1501.02473, 2015

- Using the provided matlab code,[1] one can perform the construction of polar codes in matlab, simply as follows.

```
>> N=128; K=64; Ec=1; N0=2;
```

% Blocklength, message-length, BPSK energy, and AWGN noise ($\sigma^2 = \frac{N_0}{2}$)

```
>> initPC(N,K,Ec,N0);
```

% A global structure of parameters is formed and made implicitly available for encoding/decoding

[1] http://www.ecse.monash.edu.au/staff/eviterbo/polarcodes.html

A Practical Introduction to Polar Codes

(A very simple tutorial for beginners)

Harish Vangala, Yi Hong, and Emanuele Viterbo

Monash University, Australia

23 February, 2016

This presentation, and other useful resources such as MATLAB modules can be found here:

is.gd/polarcodes

(Or, http://www.ecse.monash.edu.au/staff/eviterbo/polarcodes.html)

1. Code Construction
2. Encoding
3. Decoding

2.1 The Parameters

- An $(N, K, \mathcal{I})$ polar code is desired, where

  1. $N = 2^n$ — Code length in bits
  2. K — Information length in bits
  3. $\mathcal{I} = \texttt{bitreversed}(\mathcal{J})$ — a set of K indices, $\mathcal{I} \subset \{0, 1, \ldots, N-1\}$ (*information bit indices*)
  4. The complementary set $\mathcal{I}^c$ is called *frozen bit indices*

- The *kernel*: $\mathbf{F}^{\otimes n} \triangleq \mathbf{F} \otimes \mathbf{F} \otimes \ldots$ (n times)

$$\mathbf{F} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

$$\texttt{bitreversed}(b_1 b_2 \ldots b_n) \triangleq b_n \ldots b_2 b_1$$

where "$b_1 b_2 \ldots b_n$" is the n-bit binary form of a given index.

2.1 The Parameters

- An $(N, K, \mathcal{I})$ polar code is desired, where

  1. $N = 2^n$ — Code length in bits
  2. K — Information length in bits
  3. $\mathcal{I} = \texttt{bitreversed}(\mathcal{J})$ — a set of K indices, $\mathcal{I} \subset \{0, 1, \ldots, N-1\}$ (*information bit indices*)
  4. The complementary set $\mathcal{I}^c$ is called *frozen bit indices*

- The *kernel*: $\mathbf{F}^{\otimes n} \triangleq \mathbf{F} \otimes \mathbf{F} \otimes \ldots$ (n times)

$$\mathbf{F}^{\otimes 2} = \begin{pmatrix} \mathbf{F} & \mathbf{F} \\ 0 & \mathbf{F} \end{pmatrix}$$

$$\texttt{bitreversed}(b_1 b_2 \ldots b_n) \triangleq b_n \ldots b_2 b_1$$

where "$b_1 b_2 \ldots b_n$" is the n-bit binary form of a given index.

2.1 The Parameters

- An $(N, K, \mathcal{I})$ polar code is desired, where
  1. $N = 2^n$ — Code length in bits
  2. K — Information length in bits
  3. $\mathcal{I} = \texttt{bitreversed}(\mathcal{J})$ — a set of K indices, $\mathcal{I} \subset \{0, 1, \ldots, N-1\}$ (*information bit indices*)
  4. The complementary set $\mathcal{I}^c$ is called *frozen bit indices*
- The *kernel*: $\mathbf{F}^{\otimes n} \triangleq \mathbf{F} \otimes \mathbf{F} \otimes \ldots$ (n times)

$$\mathbf{F}^{\otimes 3} = \begin{pmatrix} \mathbf{F}^{\otimes 2} & \mathbf{F}^{\otimes 2} \\ 0 & \mathbf{F}^{\otimes 2} \end{pmatrix}$$

$$\texttt{bitreversed}(b_1 b_2 \ldots b_n) \triangleq b_n \ldots b_2 b_1$$

where "$b_1 b_2 \ldots b_n$" is the n-bit binary form of a given index.

Encoding Eq. (Kbits $\rightarrow$ Nbits):

$$\mathbf{x} = \mathbf{F}^{\otimes n}\mathbf{d} \qquad (\sim \mathbf{x} = \mathbf{G}\mathbf{u})$$

$$\text{where, } \begin{cases} \mathbf{d}_{\mathcal{I}^c} = 0, \text{ and} \\ \mathbf{d}_{\mathcal{I}} = \mathbf{u} \text{ — the message} \end{cases}$$

2.2 The Polar Encoding

Example: $N = 8, \quad K = 5, \quad \mathcal{I} = \{1, 3, 5, 6, 7\}$

$$\boxed{\mathbf{x} = \mathbf{F}^{\otimes 3}\mathbf{d}}$$

Encoding Eq. (Kbits $\rightarrow$ Nbits):

$$\mathbf{x} = \mathbf{F}^{\otimes n}\mathbf{d} \qquad (\sim \mathbf{x} = \mathbf{G}\mathbf{u})$$

$$\text{where, } \begin{cases} \mathbf{d}_{\mathcal{I}^c} = 0, \text{ and} \\ \mathbf{d}_{\mathcal{I}} = \mathbf{u} \text{ — the message} \end{cases}$$

2.2 The Polar Encoding

Example: $N = 8, \quad K = 5, \quad \mathcal{I} = \{1, 3, 5, 6, 7\}$

Encoding Eq. (Kbits $\rightarrow$ Nbits):

$$\mathbf{x} = \mathbf{F}^{\otimes n}\mathbf{d} \qquad (\sim \mathbf{x} = \mathbf{G}\mathbf{u})$$

where, $\begin{cases} \mathbf{d}_{\mathcal{I}^c} = 0, \text{ and} \\ \mathbf{d}_{\mathcal{I}} = \mathbf{u} \text{ — the message} \end{cases}$

$$\boxed{\mathbf{x} = \mathbf{F}^{\otimes 3}\mathbf{d}}$$

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{pmatrix}$$

Example: $N = 8, \quad K = 5, \quad \mathcal{I} = \{1, 3, 5, 6, 7\}$

Encoding Eq. (Kbits $\to$ Nbits):

$$\mathbf{x} = \mathbf{F}^{\otimes n}\mathbf{d} \qquad (\sim \mathbf{x} = \mathbf{G}\mathbf{u})$$

where, $\begin{cases} \mathbf{d}_{\mathcal{I}^c} = 0, \text{ and} \\ \mathbf{d}_{\mathcal{I}} = \mathbf{u} \text{ — the message} \end{cases}$

$$\boxed{\mathbf{x} = \mathbf{F}^{\otimes 3}\mathbf{d}}$$

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \boxed{d_0 = 0} \\ d_1 \\ \boxed{d_2 = 0} \\ d_3 \\ \boxed{d_4 = 0} \\ d_5 \\ d_6 \\ d_7 \end{pmatrix}$$

2.2 The Polar Encoding

Example: $N = 8, \quad K = 5, \quad \mathcal{I} = \{1, 3, 5, 6, 7\}$

Encoding Eq. (Kbits $\rightarrow$ Nbits):

$$\mathbf{x} = \mathbf{F}^{\otimes n}\mathbf{d} \qquad (\sim \mathbf{x} = \mathbf{G}\mathbf{u})$$

where, $\begin{cases} \mathbf{d}_{\mathcal{I}^c} = 0, \text{ and} \\ \mathbf{d}_{\mathcal{I}} = \mathbf{u} \text{ — the message} \end{cases}$

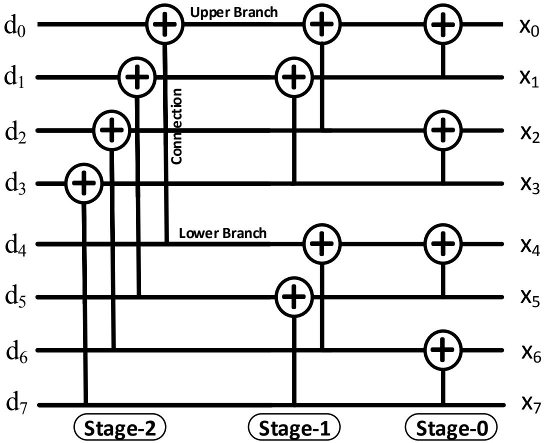
$$\boxed{\mathbf{x} = \mathbf{F}^{\otimes 3}\mathbf{d}}$$

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \boxed{d_0 = 0} \\ d_1 \\ \boxed{d_2 = 0} \\ d_3 \\ \boxed{d_4 = 0} \\ d_5 \\ d_6 \\ d_7 \end{pmatrix}$$

A very efficient $\mathcal{O}(N \log N)$ implementation is available

2.3 Efficient $\mathcal{O}(N \log_2 N)$ implementation

$$\mathbf{x} = \mathbf{F}^{\otimes n}\mathbf{d}, \text{ in just } \left(\frac{N}{2}\log_2 N\right) \text{ XORs}$$



2.4 Matlab session for encoding

- Again, using the provided matlab code,[1] one can perform the encoding of polar codes in matlab, simply as follows (assume initialization with initPC())

```
>> u=(rand(K,1)>0.5);   % K-bit random message
>> x=pencode(u);   % The efficient polar encoding
```

[1] http://www.ecse.monash.edu.au/staff/eviterbo/polarcodes.html

[2] H. Vangala, E. Viterbo, and Yi Hong, "Efficient systematic polar encoding", IEEE Communication Letters, 2016.

2.4 Matlab session for encoding

- Again, using the provided matlab code,[1] one can perform the encoding of polar codes in matlab, simply as follows (assume initialization with initPC())

```
>> u=(rand(K,1)>0.5);    % K-bit random message
>> x=pencode(u);    % The efficient polar encoding
```

- Even systematic encoding is also available

```
>> x_systematic = systematic_pencode(u);
```

---

[1] http://www.ecse.monash.edu.au/staff/eviterbo/polarcodes.html

[2] H. Vangala, E. Viterbo, and Yi Hong, “Efficient systematic polar encoding”, IEEE Communication Letters, 2016.

A Practical Introduction to Polar Codes

(A very simple tutorial for beginners)

Harish Vangala, Yi Hong, and Emanuele Viterbo

Monash University, Australia

23 February, 2016

This presentation, and other useful resources such as MATLAB modules can be found here:

is.gd/polarcodes

(Or, http://www.ecse.monash.edu.au/staff/eviterbo/polarcodes.html)

1. Code Construction
2. Encoding
3. Decoding

1. Code Construction
2. Encoding
3. **Decoding**

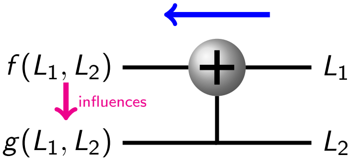
3.1 The elements of the decoding algorithm

- The basic decoder: Successive Cancellation Decoding (*aka* SCD)
- Is also *fundamental* to more advanced & efficient decoders
- Uses another two-way recursive algorithm, on N received likelihoods
- Obtains N new likelihoods in N iterations

N likelihoods of the elements in "**x**"

$\downarrow$

N likelihoods of the bits in "**d**", *sequential*



- The two likelihood operations in use:

$$\begin{pmatrix} L_1 \\ L_2 \end{pmatrix} \longrightarrow \begin{pmatrix} f(L_1, L_2) \\ g(L_1, L_2) \end{pmatrix} = \begin{pmatrix} \frac{L_1 L_2 + 1}{L_1 + L_2} \\ L_2 \cdot L_1 \text{ or } L_2 / L_1 \end{pmatrix}$$

- The second operation depends on the intermediate bit decisions from the upper branch

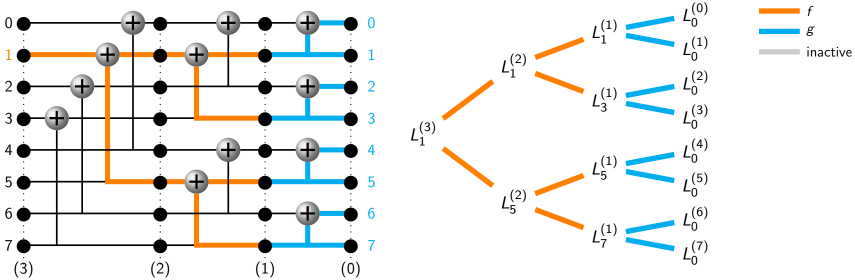
3.2 A numerical issue

- Numerical underflows are natural with using LRs
- Use of LLRs is suggested instead
- The new formulae become:

$$\begin{pmatrix} l_1 \\ l_2 \end{pmatrix} \longrightarrow \begin{pmatrix} \ln f(e^{l_1}, e^{l_2}) \\ \ln g(e^{l_1}, e^{l_2}) \end{pmatrix} = \begin{pmatrix} \ln\left(\frac{1+exp(l_1+l_2)}{\exp(l_1)+\exp(l_2)}\right) \\ l_2 + l_1 \text{ or } l_2 - l_1 \end{pmatrix}$$

$$\approx \begin{pmatrix} \text{sign}(l_1)\,\text{sign}(l_2)\,\min\{|l_1|, |l_2|\} \\ l_2 + (-1)^u l_1 \end{pmatrix}$$

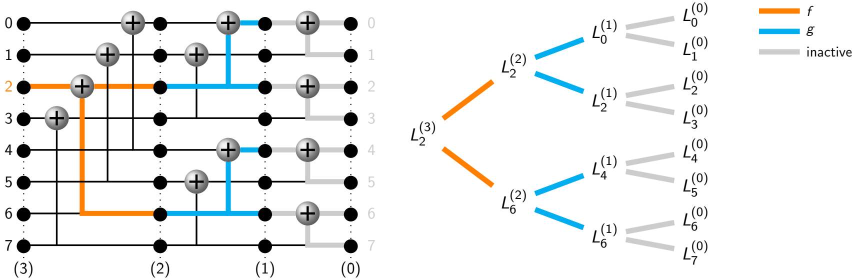
- $\rightarrow$ N input likelihoods
  $\rightarrow$ N iterations & N computational trees (different *active* depths)
  $\rightarrow$ N output likelihoods



A computational-tree: example-1

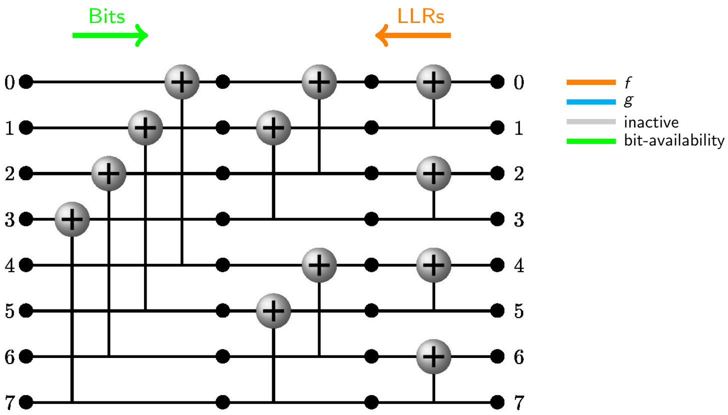
- These N trees are naturally embedded in an $N \times (n+1)$ array (shown next)
- The g formula is used only at the (entire) *last active-level*

- → N input likelihoods
  → N iterations & N computational trees (different *active* depths)
  → N output likelihoods

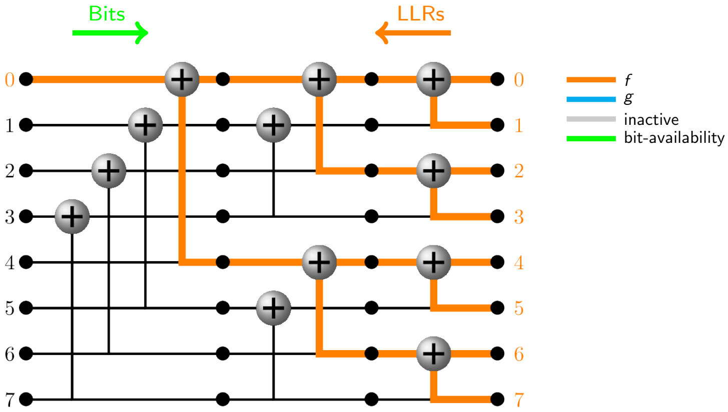


A computational-tree: example-2

- These N trees are naturally embedded in an $N \times (n+1)$ array (shown next)
- The g formula is used only at the (entire) *last active-level*



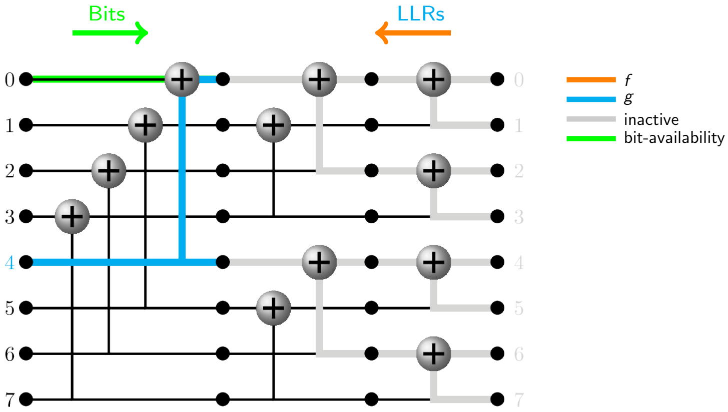
1. RHS indices (root nodes) follow bit-reversed order
   In above case of $N = 8$, it is: 0,4,2,6,1,5,3,7
2. Depth of the *active-tree* rooted at i (within the markings of f and g)
   $= 1 +$ (#consecutive-zeros in binary-i starting from MSB)
   [truncated to n]
3. The ML bit-decisions are made (only) at the root, and are broadcasted back (shown in green)



1. RHS indices (root nodes) follow bit-reversed order
   In above case of $N = 8$, it is: 0,4,2,6,1,5,3,7

2. Depth of the *active-tree* rooted at i (within the markings of f and g)
   $= 1 +$ (#consecutive-zeros in binary-i starting from MSB)
   [truncated to n]

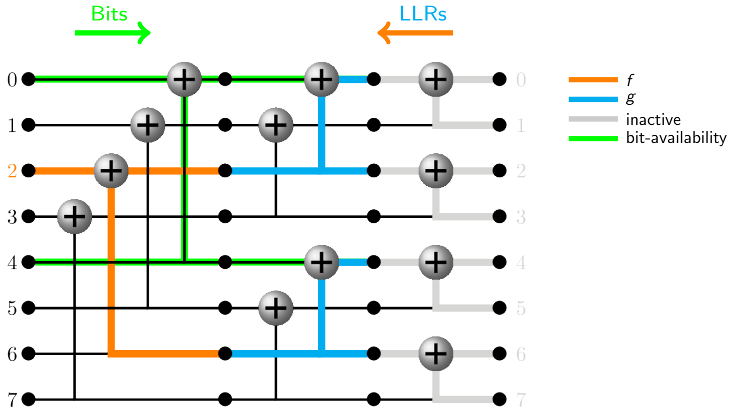
3. The ML bit-decisions are made (only) at the root, and are broadcasted back (shown in green)



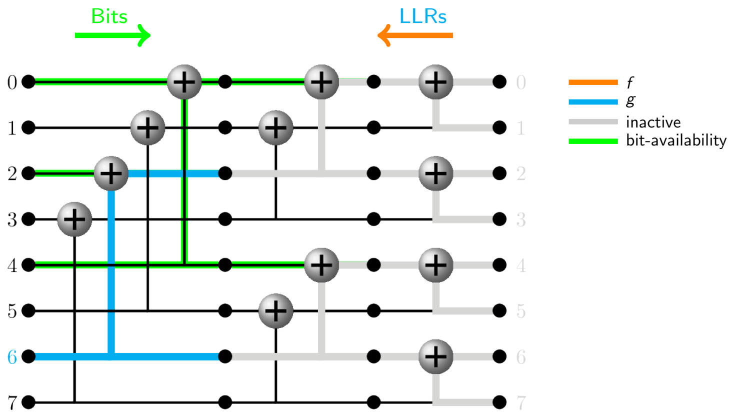
1. RHS indices (root nodes) follow bit-reversed order
   In above case of $N = 8$, it is: 0,4,2,6,1,5,3,7

2. Depth of the *active-tree* rooted at i (within the markings of f and g)
   $= 1 +$ (#consecutive-zeros in binary-i starting from MSB) [truncated to n]

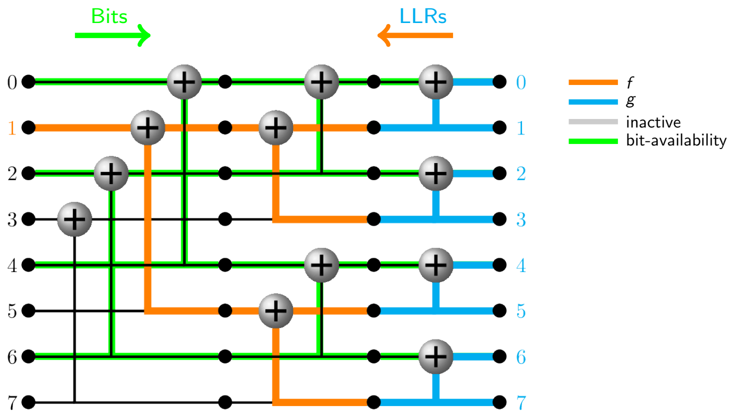
3. The ML bit-decisions are made (only) at the root, and are broadcasted back (shown in green)



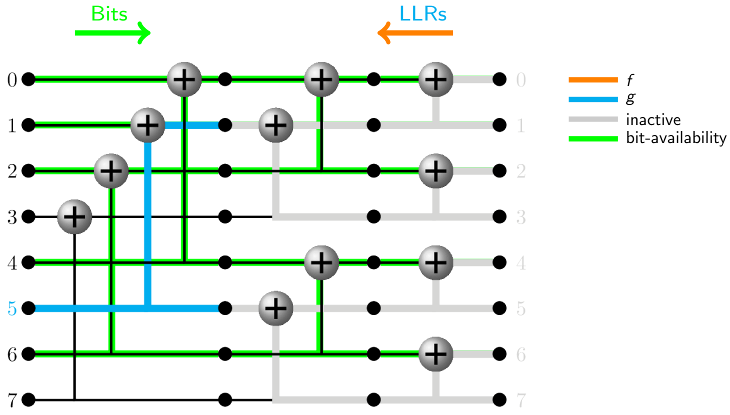
1. RHS indices (root nodes) follow bit-reversed order
   In above case of $N = 8$, it is: 0,4,2,6,1,5,3,7
2. Depth of the *active-tree* rooted at i (within the markings of f and g)
   $= 1 +$ (#consecutive-zeros in binary-i starting from MSB) [truncated to n]
3. The ML bit-decisions are made (only) at the root, and are broadcasted back (shown in green)



1. RHS indices (root nodes) follow bit-reversed order
   In above case of $N = 8$, it is: 0,4,2,6,1,5,3,7
2. Depth of the *active-tree* rooted at i (within the markings of f and g)
   $= 1 +$ (#consecutive-zeros in binary-i starting from MSB) [truncated to n]
3. The ML bit-decisions are made (only) at the root, and are broadcasted back (shown in green)



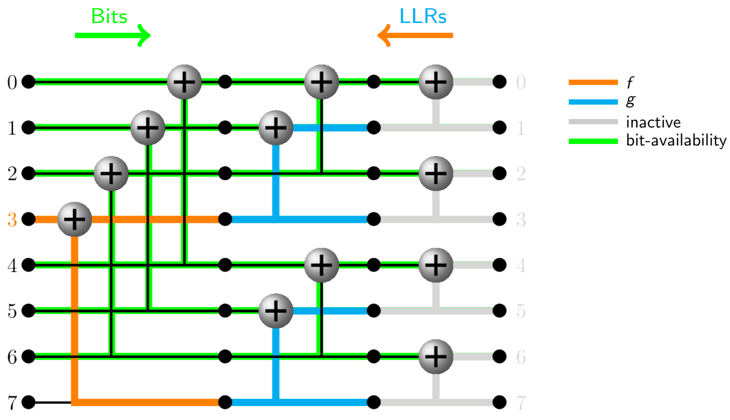
1. RHS indices (root nodes) follow bit-reversed order
   In above case of $N = 8$, it is: 0,4,2,6,1,5,3,7
2. Depth of the *active-tree* rooted at i (within the markings of f and g)
   $= 1 + (\#\text{consecutive-zeros in binary-}i \text{ starting from MSB})$
   [truncated to n]
3. The ML bit-decisions are made (only) at the root, and are broadcasted back (shown in green)



1. RHS indices (root nodes) follow bit-reversed order
   In above case of $N = 8$, it is: 0,4,2,6,1,5,3,7

2. Depth of the *active-tree* rooted at i (within the markings of f and g)
   $= 1 +$ (#consecutive-zeros in binary-i starting from MSB)
   [truncated to n]

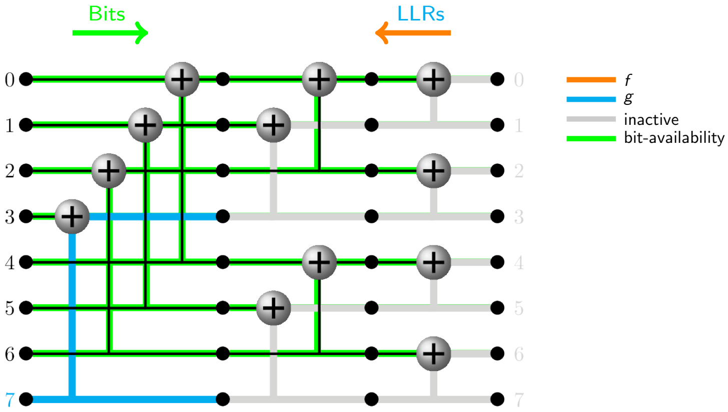
3. The ML bit-decisions are made (only) at the root, and are broadcasted back (shown in green)



1. RHS indices (root nodes) follow bit-reversed order
   In above case of $N = 8$, it is: 0,4,2,6,1,5,3,7

2. Depth of the *active-tree* rooted at i (within the markings of f and g)
   $= 1 +$ (#consecutive-zeros in binary-i starting from MSB)
   [truncated to n]

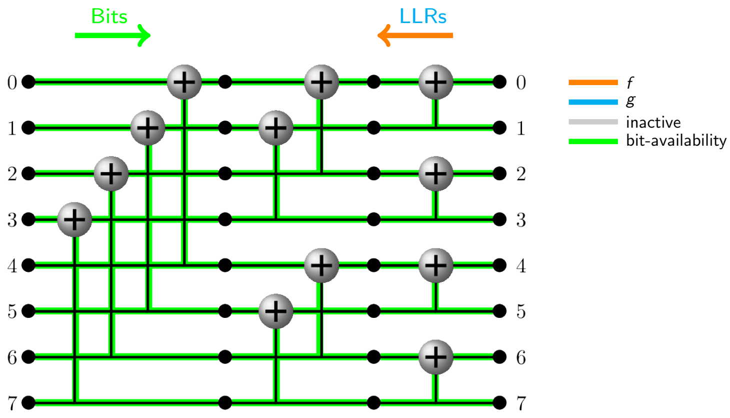
3. The ML bit-decisions are made (only) at the root, and are broadcasted back (shown in green)



1. RHS indices (root nodes) follow bit-reversed order
   In above case of $N = 8$, it is: 0,4,2,6,1,5,3,7

2. Depth of the *active-tree* rooted at i (within the markings of f and g)
   $= 1 +$ (#consecutive-zeros in binary-i starting from MSB) [truncated to n]

3. The ML bit-decisions are made (only) at the root, and are broadcasted back (shown in green)



1. RHS indices (root nodes) follow bit-reversed order
   In above case of $N = 8$, it is: 0,4,2,6,1,5,3,7
2. Depth of the *active-tree* rooted at i (within the markings of f and g)
   $= 1 +$ (#consecutive-zeros in binary-i starting from MSB) [truncated to n]
3. The ML bit-decisions are made (only) at the root, and are broadcasted back (shown in green)

3.4 MATLAB session for decoding

Using the openly available matlab code[1]: (assume initialization with initPC())

```
>> u= (rand(K,1)>0.5); % Message
>> x= pencode(u);      % Polar encoding
>> y= (2*x-1)*sqrt(Ec) + sqrt(N0/2)*randn(N,1); % AWGN

>> u_decoded= pdecode(y);
% The Successive Cancellation Decoding

>> logical(sum(u==u_decoded)) % Check if properly decoded
ans =
1
```

[1] http://www.ecse.monash.edu.au/staff/eviterbo/polarcodes.html

3.4 MATLAB session for decoding

Using the openly available matlab code[1]: (assume initialization with initPC())

```
>> u= (rand(K,1)>0.5); % Message
>> x= pencode(u);     % Polar encoding
>> y= (2*x-1)*sqrt(Ec) + sqrt(N0/2)*randn(N,1); % AWGN
```

```
>> u_decoded= pdecode(y);
% The Successive Cancellation Decoding
```

```
>> logical(sum(u==u_decoded)) % Check if properly decoded
ans =
1
```

[1] http://www.ecse.monash.edu.au/staff/eviterbo/polarcodes.html

4. Performance plots

Using the openly available matlab code[1]:

```
>> N=128; K=64; EbN0range=0:0.4:2; designSNRdB=0;
>> plotPC(N,K,EbN0range,designSNRdB,0); %last argument avoids being verbose
```

[1] http://www.ecse.monash.edu.au/staff/eviterbo/polarcodes.html

4. Performance plots

Using the openly available matlab code[1]:

```
>> N=128; K=64; EbN0range=0:0.4:2; designSNRdB=0;

>> plotPC(N,K,EbN0range,designSNRdB,0); %last argument avoids being verbose

Completed SNR points (out of 6):
0.00 dB (time taken:27.69 sec)
0.40 dB (time taken:27.18 sec)
0.80 dB (time taken:27.18 sec)
1.20 dB (time taken:27.16 sec)
1.60 dB (time taken:27.10 sec)
2.00 dB (time taken:40.55 sec)

 Eb/N0 range (dB) :   0        0.4000   0.8000   1.2000   1.6000   2.0000

 Frame Error Rates:   0.7080   0.5930   0.4790   0.3730   0.2400   0.1342

 Bit Error Rates :    0.2311   0.1776   0.1393   0.1012   0.0646   0.0317
```

[1] http://www.ecse.monash.edu.au/staff/eviterbo/polarcodes.html

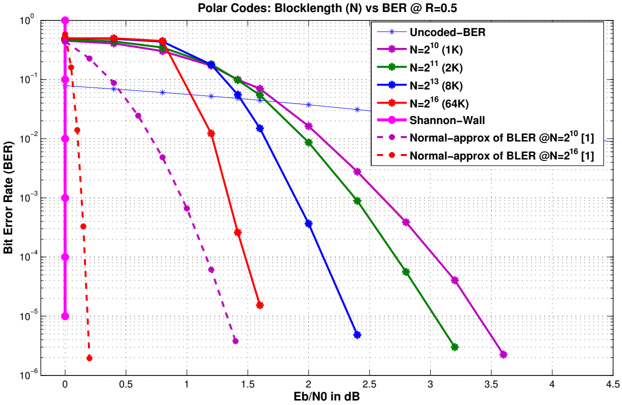


Figure: Sample performance of polar codes under basic SCD, when $R = 0.5$

[1] Eq. (296) of Y. Polyanskiy, H. V. Poor, and S. Verdu, "*Channel Coding Rate in the Finite Blocklength Regime*", IEEE Transactions on Information Theory, 2010, 56, 2307–2359.

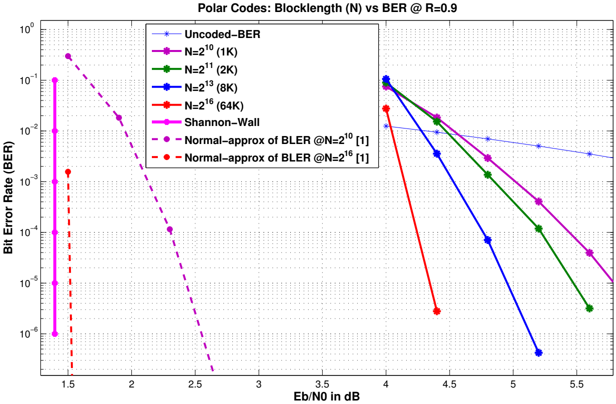


Figure: Sample performance of polar codes under basic SCD, when $R = 0.9$

[1] Eq. (296) of Y. Polyanskiy, H. V. Poor, and S. Verdu, "*Channel Coding Rate in the Finite Blocklength Regime*", IEEE Transactions on Information Theory, 2010, 56, 2307–2359.

References

1. Polar coding algorithms in MATLAB, http://www.ecse.monash.edu.au/staff/eviterbo/polarcodes.html
2. Erdal Arıkan, *Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels*, IEEE Trans. IT, 2009

Some of our works:

3. H. Vangala, E. Viterbo, and Yi Hong, *A Comparative Study of Polar Code Constructions for the AWGN Channel*, arXiv:1501.02473, 2015
4. H. Vangala, E. Viterbo, and Yi Hong, *Efficient systematic polar encoding*, IEEE Communication Letters, 2015
5. H. Vangala, E. Viterbo, and Yi Hong, *Permuted successive cancellation decoder for polar codes*, International Symposium on Information Theory and its Applications, ISITA 2014, Melbourne , Oct. 2014
6. H. Vangala, E. Viterbo, and Yi Hong, *A new multiple folded successive cancellation decoder for polar codes*, Information Theory Workshop, ITW 2014, Hobart, Tasmania , Nov. 2014