List Decoding of Universal Polar Codes

Boaz Shuval and Ido Tal
Overview

- Universal polarization:
  - memoryless setting: Şaşoğlu & Wang, ISIT 2011
  - settings with memory: Shuval & Tal, ISIT 2019

- Error probability analysis based on successive cancellation (SC) decoding

- Potential for better results using successive cancellation list (SCL) decoding

This talk:
An efficient implementation of SCL for universal polarization
Setting

- Communication with uncertainty:
  - **Encoder**: Knows channel belongs to a set of channels
  - **Decoder**: Knows channel statistics (e.g., via estimation)

- Memory:
  - In channels
  - In input distribution

- Universal polar code achieves:
  - Vanishing error probability over set
  - Best rate (infimal information rate over set)
List decoding improves probability of error!

Capacity vs Word Error Rate for BSC with SC upper bound.
List decoding improves probability of error!

Word Error Rate

Capacity

BSC

SC upper bound
List decoding improves probability of error!

capacity word error rate

BSC

$L = 1$

SC upper bound
List decoding improves probability of error!

Capacity vs. Word Error Rate for BSC with different list sizes $L = 1, 2, 4, 8, 16$.

SC upper bound.
List decoding improves probability of error!

Capacity vs. Word Error Rate for different values of $L$ in the BEC channel.
List decoding improves probability of error!

Capacity

Word Error Rate

AWGN

- $L = 1$
- $L = 2$
- $L = 4$
- $L = 8$
- $L = 16$

SC upper bound
SCL in a nutshell

- **Parameter**: list size $L$
- Successively decode $u_0, u_1, \ldots, u_{N-1}$
- **Decoding path**: sequence of decoding decisions
  \[
  \hat{u}_{i-1} = [\hat{u}_0 \ \hat{u}_1 \ \ldots \ \hat{u}_{i-1}]
  \]
- If $u_i$ is not frozen, split each decoding path into two paths,
  \[
  \hat{u}_{i-1} \rightarrow [\hat{u}_{i-1} \ 0]
  \]
  \[
  \hat{u}_{i-1} \rightarrow [\hat{u}_{i-1} \ 1]
  \]
- For each $i$, keep $L$ best decoding paths
- **Final decoding**: choose best path
The Universal Construction

- Concatenation of two transform blocks

\[ U_0^{N-1} \rightarrow \text{fast} \rightarrow \text{slow} \rightarrow X_0^{N-1} \rightarrow \text{channel} \rightarrow Y_0^{N-1} \]

- Slow — for universality
- Fast (Arıkan) — for vanishing error probability
The Universal Construction

- Concatenation of two transform blocks

\[ U_0^{N-1} \rightarrow X_0^{N-1} \rightarrow \text{channel} \rightarrow Y_0^{N-1} \]

- Slow — for universality
- Fast (Arıkan) — for vanishing error probability

- Each block contains multiple copies of basic transforms
- **Polar**: certain indices of \( U_0^{N-1} \) are data bits, other indices “frozen” (Honda & Yamamoto 2013)
- **Universal**: data bit locations regardless of channel
Fast Transform vs. Slow Transform

Fast (Arıkan) transform

Slow transform
Fast Transform vs. Slow Transform

Fast (Arıkan) transform

Slow transform
Fast Transform vs. Slow Transform

Fast (Arıkan) transform

Slow transform
Fast Transform vs. Slow Transform

Fast (Arıkan) transform

Slow transform

lateral

medial

lateral
Fast Transform vs. Slow Transform

Fast (Arıkan) transform

Slow transform

lateral

medial

lateral
The Rules of Slow

Any medial on left is a child of medial − and medial + on right
The Rules of Slow

- Any medial on left is a child of medial − and medial + on right.
The Rules of Slow

- Any medial on left is a child of medial − and medial + on right

- Update of a medial + on left updates both medial − and medial + on right
The Rules of Slow

- Any medial on left is a child of medial − and medial + on right
- Update of a medial + on left updates both medial − and medial + on right
The Rules of Slow

- Any medial on left is a child of medial $-$ and medial $+$ on right
- Update of a medial $+$ on left updates both medial $-$ and medial $+$ on right
The Rules of Slow

- Any medial on left is a child of medial $-$ and medial $+$ on right
- Update of a medial $+$ on left updates both medial $-$ and medial $+$ on right
The Rules of Slow

- Any medial on left is a child of medial − and medial + on right
- Update of a medial + on left updates both medial − and medial + on right

Corollary

*Update of a medial + on left results in a cascade of updates, all the way to the channel on the far right*
List Decoding — Fast Transform vs. Slow Transform

**Fast Transform**

The layer at depth $\lambda$ gets updated once every $2^\lambda$ writes to the $U$ vector.

Deep layers get updated infrequently.

**Slow Transform**

The layer at depth $\lambda$ gets updated every other write to the $U$ vector.

All layers get updated frequently.
List Decoding — Fast Transform vs. Slow Transform

**Fast Transform**

The layer at depth $\lambda$ gets updated once every $2^\lambda$ writes to the $U$ vector

Deep layers get updated infrequently

Good news for sharing data between list decoding paths

**Slow Transform**

The layer at depth $\lambda$ gets updated every other write to the $U$ vector

All layers get updated frequently

Bad news for sharing data between list decoding paths
Is All Hope Lost?

Fast Transform
- Layer at depth $\lambda$ is updated once every $2^\lambda$ writes
- At layer of depth $\lambda$, $2^\lambda$ indices are updated at once

Slow Transform
- Layer at depth $\lambda$ is updated at every other write
- At layer of depth $\lambda$, at most two indices are updated
Is All Hope Lost?

Fast Transform

- Layer at depth $\lambda$ is updated \textit{once every $2^\lambda$ writes}
- At layer of depth $\lambda$, $2^\lambda$ indices are updated at once

Slow Transform

- Layer at depth $\lambda$ is updated at \textit{every other write}
- At layer of depth $\lambda$, \textit{at most two indices} are updated

Our Saving Grace

The updates are cyclic!
A Bespoke Data Structure

- **Aim:** mimic the fast transform’s update regime
  - Frequent *small* updates
  - Infrequent *large* updates

- **Bespoke data structure:** Cyclic Exponential Array

- **Main idea:** an array comprised of sub-arrays, growing exponentially in length
Example

entire array:

cyclic exponential array:

last written value
Example

entire array:

0


cyclic exponential array:

0

last written value
Example

entire array:

```
0 1
```

cyclic exponential array:

```
0
1
```

last written value
Example

entire array:

0 1 2


cyclic exponential array:

0 1

2 last written value
Example

entire array:

```
0 1 2 3
```

cyclic exponential array:

```
0 1 2 3
```

last written value

3
Example

entire array:

```
0 1 2 3 4
```

cyclic exponential array:

```

0 1 2 3
```

last written value

4
Example

entire array:

```
0 1 2 3 4 5
```

cyclic exponential array:

```
0 1 2 3
0 1 2 3
4
5
```

last written value
Example

entire array:

0 1 2 3 4 5 6

0 1 2 3 4 5 6

0 1 2 3

4 5

6

last written value
Example

entire array:

```
0 1 2 3 4 5 6 7
```

cyclic exponential array:

```
0 1 2 3
4 5
6
7
```

last written value: 7
Example

entire array:

```
0 1 2 3 4 5 6 7 8
```

cyclic exponential array:

```
0 1 2 3 4 5 6 7
```

last written value

8
Example

entire array:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

cyclic exponential array:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 12/15 |

last written value:
Example

entire array:

0 1 2 3 4 5 6 7 8 9 10

Cyclic exponential array:

0 1 2 3 4 5 6 7

8 9

10

Last written value
Example

entire array:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
</table>

Cyclic exponential array:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

last written value
Example

entire array:

```
0 1 2 3 4 5 6 7 8 9 10 11 12
```

cyclic exponential array:

```
0 1 2 3 4 5 6 7
8 9 10 11
```

12 — last written value
Example

entire array:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13
```

cyclic exponential array:

```
0 1 2 3 4 5 6 7
8 9 10 11
12
```

last written value

13
Example

entire array:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Cyclic exponential array:

0 1 2 3 4 5 6 7
8 9 10 11
12 13

14 last written value
Example

entire array:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Cyclic exponential array:

0 1 2 3 4 5 6 7
8 9 10 11
12 13
14
15

last written value
Example

entire array:

[cyclic exponential array:  

<table>
<thead>
<tr>
<th>a</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

previous cycle

last written value
Example

entire array:

| a | b | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

cyclic exponential array:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

previous cycle

a

b

last written value
Example

entire array:

```
[ a, b, c, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ]
```

cyclic exponential array:

```
[ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ]
```

previous cycle

```
[ a, b ]
```

last written value

```
[ c ]
```
Example

entire array:

- a
- b
- c
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

cyclic exponential array:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

previous cycle

And so on...

last written value
Let:
- Blocklength $N$
- Channel with or without memory ($S$ states)
- List size $L$

**Theorem**

*Our SCL decoder for universal polar codes has:*
- **Running time:** $O(L \cdot S^3 \cdot N \log N)$
- **Space complexity:** $O(L \cdot S^2 \cdot N)$
We made the deadline, thanks to

- Computing resources (cores galore)
  - Amir Baer
  - Goel Samuel
- And we parallelized using the excellent
  - GNU Parallel by Ole Tange\(^1\)
- Origami artwork by Bernie Peyton

---
Something to look forward to

Look out for our code on github!

For the impatient: email us at polarbear@technion.ac.il