The Local Weight Distributions of Transitive Invariant Codes and Their Punctured Codes

Kenji YASUNAGA and Toru FUJIWARA

Osaka University, Osaka, Japan

HISC2005, Hawaii, USA

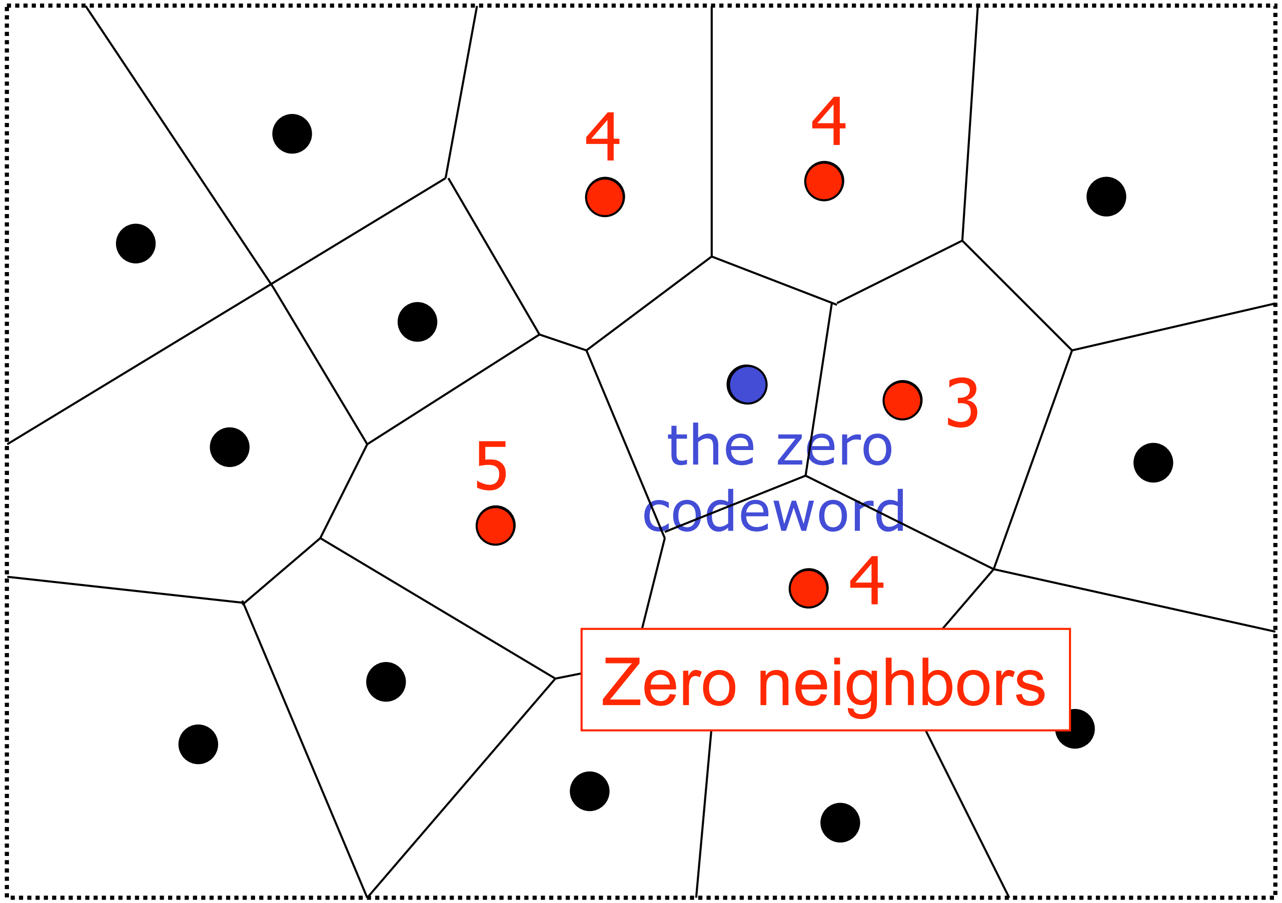
Outline

- Local Weight Distribution (LWD)
  - Definition, known results, motivation
- Relation between LWDs of a transitive invariant code (including ext. BCH, Reed-Muller) and its punctured code
  1. Relation for the extended code and the original code (general case)
  2. Useful relation for the case an extended code is a transitive invariant code
- Results
- Conclusion

Local Weight Distribution (LWD)

The LWD is the weight distribution of the neighbor codewords to the zero codeword (called zero neighbors).

Codewords of C on $\boldsymbol{R}^n$



The LWD of C

weight	*the number of zero neighbors*
3	1
4	3
5	1

What for We Obtain LWD?

- LWD is valuable for the error performance analysis of codes.
  - Union bound is a well-known upper bound of the error probability for soft decision decoding on AWGN channel.
    - We often use the weight distribution to compute the union bound.
  - Using LWD instead of the weight distribution, we could obtain a tighter upper bound than the usual union bound.

Known Results for LWD

- Hamming codes and second-order Reed-Muller codes
  - The formulas for the LWDs are known [Ashikhmin, Barg 98].
- Primitive BCH codes
  - (63, *k*) codes for all *k* [Mohri, Honda, Morii 03]
- Extended primitive BCH codes
  - (128, *k*) codes for *k*≦50 [Yasunaga, Fujiwara 03]
- Reed-Muller codes
  - Third-order RM code of length 128 [Yasunaga, Fujiwara 04]

Numerical Computation

Goal

- Obtain LWDs of (127,43), (127,50) BCH codes

  Motivation:

  - LWDs of (128,43) and (128,50) extended BCH codes are obtained.
  - LWDs of (127,43) or (127,50) BCH codes are not obtained.

Investigate a relation between LWDs of a code C and its extended code C_{ex}

Condition for zero neighbor

- Support set: $\mathrm{Supp}(\boldsymbol{v}) = \{\, i : v_i \neq 0 \text{ for } \boldsymbol{v} = (v_1, v_2, \ldots, v_n) \,\}$

> $\boldsymbol{v}$ is a zero neighbor in C
>
> ⇔ C does not contain $\boldsymbol{v}_1$, $\boldsymbol{v}_2$ such that
>
> $\boldsymbol{v} = \boldsymbol{v}_1 + \boldsymbol{v}_2$, $\mathrm{Supp}(\boldsymbol{v}_1) \cap \mathrm{Supp}(\boldsymbol{v}_2) = \phi$.

$\boldsymbol{v}$	1	1	0	1	0	1	1	1	0
					=				
$\boldsymbol{v}_1$	1	1	0	0	0	1	0	1	0
					+				
$\boldsymbol{v}_2$	0	0	0	1	0	0	1	0	0

If C contains such $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, $\boldsymbol{v}$ is called decomposable.
($\boldsymbol{v}$ is decomposable ⇔ $\boldsymbol{v}$ is not a zero neighbor)

Questions

$\boldsymbol{v}$: a codeword in C
$\boldsymbol{v}^{(ex)}$: the extended codeword of $\boldsymbol{v}$

(1) If $\boldsymbol{v}$ is a zero neighbor in C,
is $\boldsymbol{v}^{(ex)}$ a zero neighbor in C_{ex}?

(2) If $\boldsymbol{v}$ is not a zero neighbor in C,
is $\boldsymbol{v}^{(ex)}$ not a zero neighbor in C_{ex}?

Simple Observation

(1) If $\boldsymbol{v}$ is a zero neighbor in C,

⇒ $\boldsymbol{v}^{(\mathrm{ex})}$ is a zero neighbor in C_{ex}.

(2) If $\boldsymbol{v}$ is not a zero neighbor in C,

⇒ (a) In the case $\mathrm{weight}(\boldsymbol{v})$ is odd,

$\boldsymbol{v}^{(\mathrm{ex})}$ is not a zero neighbor in C_{ex}.

(b) In the case $\mathrm{weight}(\boldsymbol{v})$ is even,

both cases can occur.

(2) v is not a zero neighbor in C
(b) weight(v) is even

v	1	1	0	1	0	1	1	1	0
					=				
v_1	1	1	0	0	0	1	0	1	0
					+				
v_2	0	0	0	1	0	0	1	0	0

v is decomposable into $v_1 + v_2$, $v_1, v_2 \in C$

(2) $\boldsymbol{v}$ is not a zero neighbor in C
(b) weight($\boldsymbol{v}$) is even

(i) Both $\boldsymbol{v}_1, \boldsymbol{v}_2$ are even weight

$\boldsymbol{v}^{(ex)}$	1	1	0	1	0	1	1	1	0	0
=										
$\boldsymbol{v}_1^{(ex)}$	1	1	0	0	0	1	0	1	0	0
+										
$\boldsymbol{v}_2^{(ex)}$	0	0	0	1	0	0	1	0	0	0

parity bit

$\boldsymbol{v}^{(ex)}$ is decomposable (not a zero neighbor)

(ii) Both $\boldsymbol{v}_1, \boldsymbol{v}_2$ are odd weight

$\boldsymbol{v}^{(ex)}$	1	1	0	1	0	1	1	1	0	0
=										
$\boldsymbol{v}_1^{(ex)}$	0	1	0	0	0	1	0	1	0	1
+										
$\boldsymbol{v}_2^{(ex)}$	1	0	0	1	0	0	1	0	0	1

parity bit

$\boldsymbol{v}^{(ex)}$ is not decomposable into $\boldsymbol{v}_1^{(ex)}+\boldsymbol{v}_2^{(ex)}$

If (ii) occurs for all $\boldsymbol{v}$'s decompositions, $\boldsymbol{v}$ is called only-odd decomposable, and $\boldsymbol{v}^{(ex)}$ is a zero neighbor in C_{ex}.

Relation of zero neighborship in C and C_{ex}

- If C contains no only-odd decomposable codeword,

  $\boldsymbol{v}$ is a zero neighbor in C
  $\Leftrightarrow$ $\boldsymbol{v}^{(\mathrm{ex})}$ is a zero neighbor in C_{ex}

What is a condition of C to contain no only-odd decomposable codeword?

Theorem 3:

If all the weights of codewords in C_{ex} are multiples of four, C contains no only-odd decomposable codeword.

- Examples of above C_{ex}:
  - (128,*k*) extended primitive BCH codes with $k \leqq 57$
  - Third-order Reed-Muller codes of length 128, 256, 512

Relation between LWDs of C and C_{ex}

- If C contains no only-odd decomposable codeword,
  ⇒ we can obtain LWD of C_{ex} from LWD of C

but

We'd like to obtain LWD of C from LWD of C_{ex}.

- To do this, we need to know the number of zero neighbors with parity bit one.

For a transitive invariant code C_{ex}

Lemma 3:
For a transitive invariant code C_{ex} of length $n+1$,

$$\#\left(\begin{array}{l}\text{zero neighbors of parity} \\ \text{bit one in } \mathrm{C}_{\mathrm{ex}} \text{ with weight } w\end{array}\right) = \frac{w L_w(C_{\mathrm{ex}})}{n+1}$$

$L_w(\mathrm{C}_{\mathrm{ex}})$:= #(zero neighbors in C_{ex} with weight w)

Transitive invariant codes:
extended primitive BCH codes, Reed-Muller codes

Theorem 8.15,
W. W. Peterson and E. J. Weldon, Jr., *Error correcting codes*, *2nd Edition*, 1972

Results

- We obtained LWDs of (127,43) and (127,50) primitive BCH codes.
  - From
    LWDs of (128,43) and (128,50) extended primitive BCH codes

LWD of (127,50) primitive BCH code

weight	*the number of zero neighbors*
27	40894
28	146050
31	4853051
32	14559153
35	310454802
36	793384494
39	10538703840
40	23185148448
43	199123183160
44	380144258760
47	2154195406104
48	3590325676840
51	13633106229288

weight	*the number of zero neighbors*
52	19925309104344
55	51285782220204
56	65938862854548
59	115927157830260
60	131384112207628
63	158486906385472
64	158486906385472
67	131258388369668
68	115816225032060
71	64917266933304
72	50491207614792
75	15345182164032
76	10499335164864

For a code C containing only-odd decomposable codewords

> Corollary 1:
> An only-odd decomposable codeword has only one decomposition.

- The number of only-odd decomposable codewords in C
  - Using a known efficient algorithm for LWD of C ⇒ Computable
  - Using a known efficient algorithm for LWD of C_{ex} ⇒ Open problem

more efficient when C is BCH

Conclusion

- LWD is valuable for an error performance analysis.
- Relation between LWDs of a transitive invariant code and its punctured code
  - Relation between C and C_{ex}
  - We give a useful relation in the case
    - C_{ex} is a transitive invariant code
    - C contains no only-odd decomposable codeword
- We obtained LWDs of (127,43) and (127,50) primitive BCH codes.

Relation between LWDs of C and Cex

Theorem 4:
If C_{ex} is a transitive invariant code length $n+1$,

$$L_w(C) = \frac{w+1}{n+1} L_{w+1}(C_{\mathrm{ex}}), \qquad \text{for odd } w,$$

$$L_w(C) = \frac{n+1-w}{n+1} L_w(C_{\mathrm{ex}}) - N_w, \qquad \text{for even } w.$$

N_w := #(only-odd decomposable codewords with weight w)

If we know N_w,
We can obtain LWD of C from LWD of C_{ex}.

Corollary 2:

For a code C of length n,

$$L_{2i}(C_{\mathrm{ex}}) = L_{2i-1}(C) + L_{2i}(C) + N_{2i},\ 0 \le i \le n/2.$$

$L_w(\mathrm{C})$:= #(zero neighbors in C with weight w)
N_w := #(only-odd decomposable codewords with weight w)

If we know N_w,
we could obtain LWD of C_{ex} from LWD of C.