Arbitrarily Low Redundancy Construction of Balanced Codes Using Realistic Resources

Danny Dubé

Quebec City, Quebec, Canada

Workshop on Error-Correcting Codes — September 3, 2020

Contents of the Talk

- Introduction to balanced blocks
- Non-scalable mathematical constructions
- A few previous practical constructions
- Our per-block optimal construction:
  based on permutations, Pacman,
  and small integers
- Our arbitrarily low redundancy construction

Contents of the Talk

$\Rightarrow$ Introduction to balanced blocks

- Non-scalable mathematical constructions
- A few previous practical constructions
- Our per-block optimal construction:
  based on permutations, Pacman,
  and small integers
- Our arbitrarily low redundancy construction

Balanced Codes

Example of an encoding with $Q = 4$ and $M = 6$.

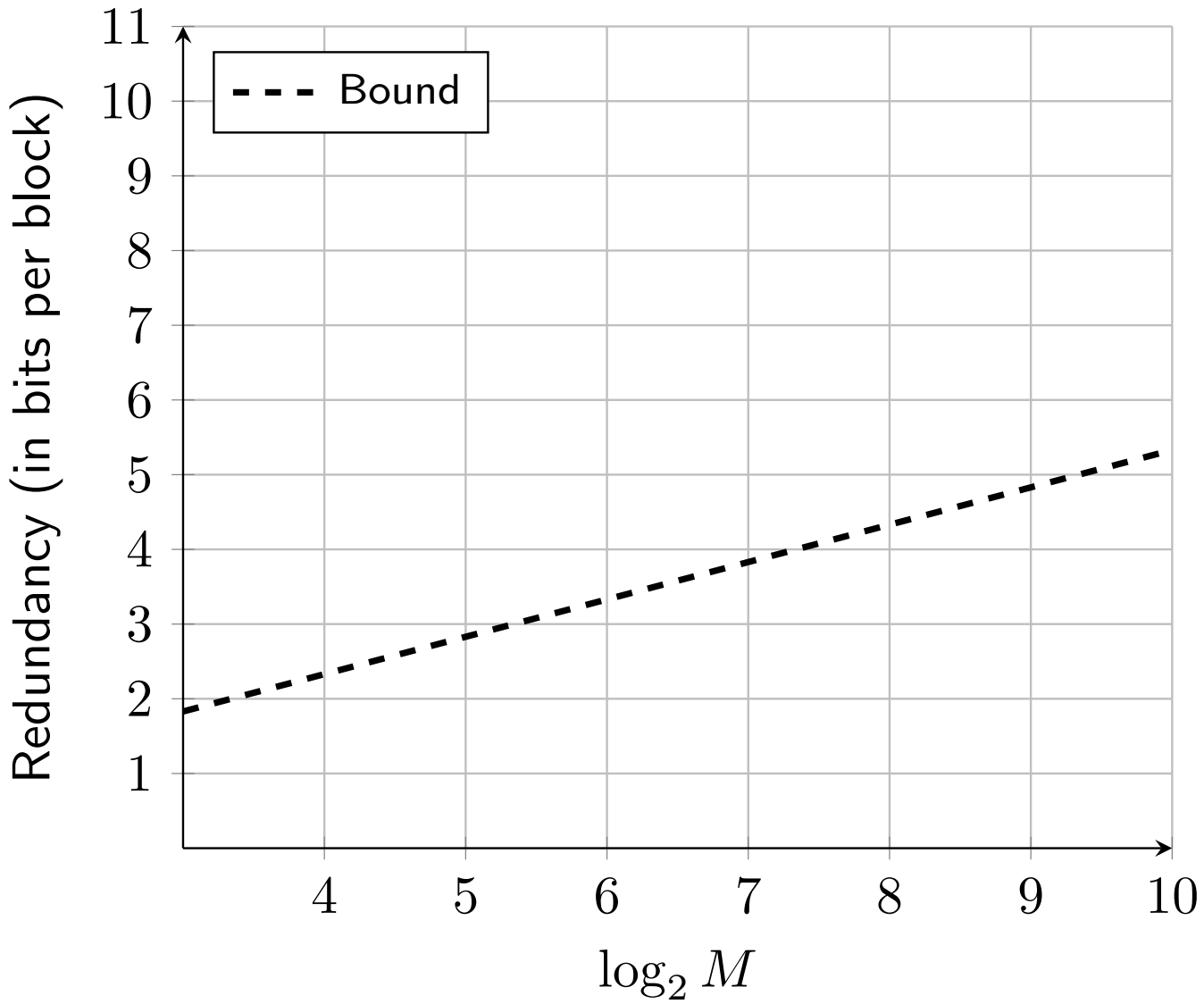
Input	Balanced
0000	000111
0001	001011
0010	001101
0011	001110
0100	010011
0101	010101

Input	Balanced
0110	010110
0111	011001
1000	011010
1001	011100
1010	100011
1011	100101

Input	Balanced
1100	100110
1101	101001
1110	101010
1111	101100

Balanced codes are needed for various storage/transmission devices.

Redundancy


Bound
Redundancy (in bits per block)
11
10
9
8
7
6
5
4
3
2
1
4
5
6
7
8
9
10
$\log_2 M$

Contents of the Talk

- Introduction to balanced blocks

$\Rightarrow$ Non-scalable mathematical constructions

- A few previous practical constructions
- Our per-block optimal construction:
  based on permutations, Pacman,
  and small integers
- Our arbitrarily low redundancy construction

Enumerative Coding

Let M be the size of the balanced blocks:

Number of balanced blocks of size M:

$$\binom{M}{M/2}$$

Theoretical data-embedding capacity, in bits, of one balanced block:

$$\log_2 \binom{M}{M/2}$$

Optimal embedding capacity of an (integer) number of bits in a per-block setting:

$$\left\lfloor \log_2 \binom{M}{M/2} \right\rfloor$$

Enumerative Coding

Enumerative coding consists in choosing an injective mapping of blocks of Q bits into balanced blocks of M bits

I.e. the mapping has type $2^Q \to \mathcal{B}_M$

E.g., $Q = 2$ and $M = 4$:

Input	Balanced
00	0011
01	0101
10	0110
11	1001

Enumerative Coding

E.g., $Q = 4$ and $M = 6$:

Input	Balanced
0000	000111
0001	001011
0010	001101
0011	001110
0100	010011
0101	010101
0110	010110
0111	011001

Input	Balanced
1000	011010
1001	011100
1010	100011
1011	100101
1100	100110
1101	101001
1110	101010
1111	101100

Enumerative Coding

E.g., $Q = 6$ and $M = 8$:

Input	Balanced
000000	00001111
000001	00010111
000010	00011011
000011	00011101
000100	00011110
000101	00100111
000110	00101011
000111	00101101
001000	00101110
001001	00110011
001010	00110101
001011	00110110
001100	00111001
001101	00111010
001110	00111100
001111	01000111

Input	Balanced
010000	01001011
010001	01001101
010010	01001110
010011	01010011
010100	01010101
010101	01010110
010110	01011001
010111	01011010
011000	01011100
011001	01100011
011010	01100101
011011	01100110
011100	01101001
011101	01101010
011110	01101100
011111	01110001

Input	Balanced
100000	01110010
100001	01110100
100010	01111000
100011	10000111
100100	10001011
100101	10001101
100110	10001110
100111	10010011
101000	10010101
101001	10010110
101010	10011001
101011	10011010
101100	10011100
101101	10100011
101110	10100101
101111	10100110

Input	Balanced
110000	10101001
110001	10101010
110010	10101100
110011	10110001
110100	10110010
110101	10110100
110110	10111000
110111	11000011
111000	11000101
111001	11000110
111010	11001001
111011	11001010
111100	11001100
111101	11010001
111110	11010010
111111	11010100

Enumerative Coding

Doesn't scale for larger M !

Keeping lookup tables in memory requires too much space

Computing the mapping on demand requires the use of large integers: slow

Contents of the Talk

- Introduction to balanced blocks
- Non-scalable mathematical constructions

$\Rightarrow$ A few previous practical constructions

- Our per-block optimal construction:
  based on permutations, Pacman,
  and small integers
- Our arbitrarily low redundancy construction

Knuth's Technique

Technique:

1. Identify some prefix of the source block that we may flip to reach balance
2. Add a prefix to the now balanced source block to remember how many bits were flipped

Knuth's Technique

Balancing of 01011101:

Original data:

0	1	0	1	1	1	0	1

Knuth's Technique

Balancing of 01011101:

Original data:

0	1	0	1	1	1	0	1

Trying 1-bit flip:

$\overline{0}$	1	0	1	1	1	0	1

Knuth's Technique

Balancing of 01011101:

Original data:

0	1	0	1	1	1	0	1

Trying 1-bit flip:

$\overline{1}$	1	0	1	1	1	0	1

Unbalanced!

Knuth's Technique

Balancing of 01011101:

Original data:	0	1	0	1	1	1	0	1
Trying 1-bit flip:	$\overline{1}$	1	0	1	1	1	0	1
Trying 2-bit flip:	$\overline{1}$	$\overline{0}$	0	1	1	1	0	1

Unbalanced!

Knuth's Technique

Balancing of 01011101:

Original data:

0	1	0	1	1	1	0	1

Trying 1-bit flip:

1	1	0	1	1	1	0	1

Unbalanced!

Trying 2-bit flip:

1	0	0	1	1	1	0	1

Unbalanced!

Knuth's Technique

Balancing of 01011101:

Original data:	0	1	0	1	1	1	0	1
Trying 1-bit flip:	$\overline{1}$	1	0	1	1	1	0	1
Trying 2-bit flip:	$\overline{1}$	$\overline{0}$	0	1	1	1	0	1
Trying 3-bit flip:	$\overline{1}$	$\overline{0}$	$\overline{1}$	1	1	1	0	1

Unbalanced!

Unbalanced!

Knuth's Technique

Balancing of 01011101:

Original data:	0	1	0	1	1	1	0	1	
Trying 1-bit flip:	$\overline{1}$	1	0	1	1	1	0	1	Unbalanced!
Trying 2-bit flip:	$\overline{1}$	$\overline{0}$	0	1	1	1	0	1	Unbalanced!
Trying 3-bit flip:	$\overline{1}$	$\overline{0}$	$\overline{1}$	1	1	1	0	1	Unbalanced!

Knuth's Technique

Balancing of 01011101:

Original data:	0	1	0	1	1	1	0	1
Trying 1-bit flip:	1	1	0	1	1	1	0	1
Trying 2-bit flip:	1	0	0	1	1	1	0	1
Trying 3-bit flip:	1	0	1	1	1	1	0	1
Trying 4-bit flip:	1	0	1	0	1	1	0	1

Unbalanced!

Unbalanced!

Unbalanced!

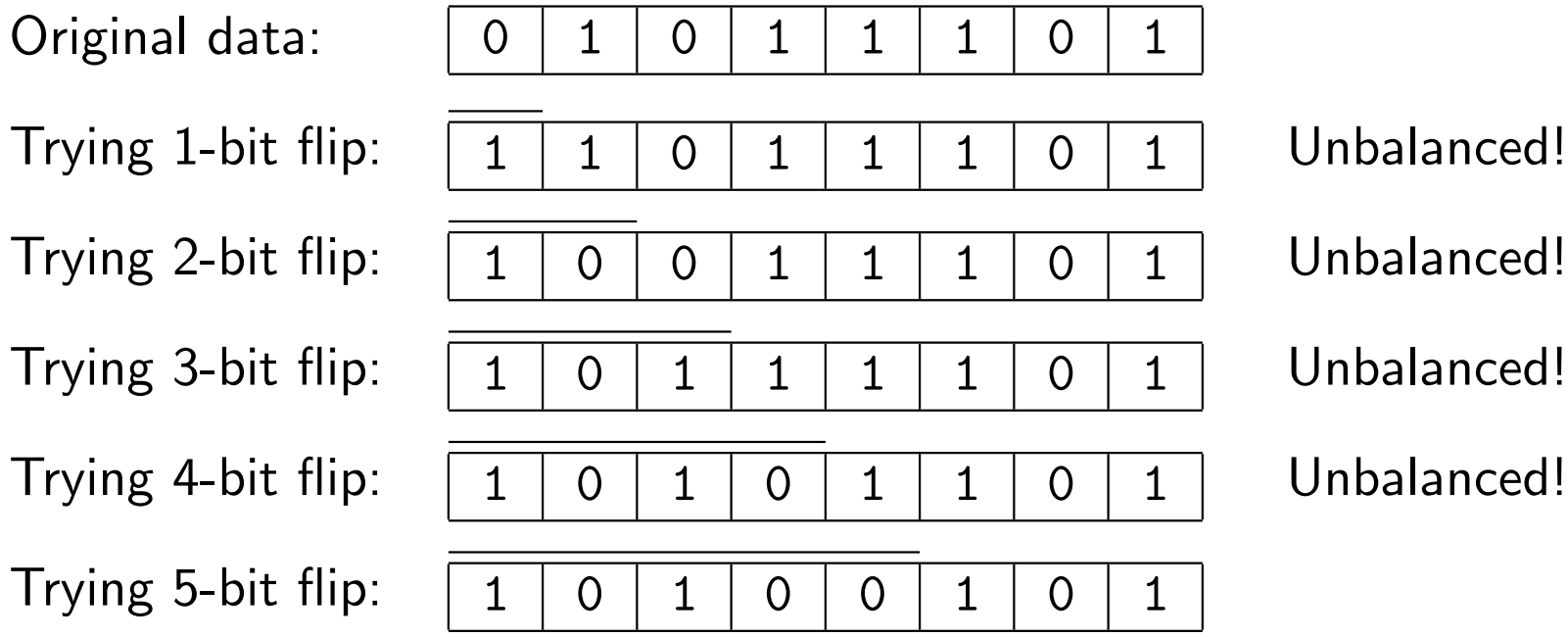
Knuth's Technique

Balancing of 01011101:

Original data:	0	1	0	1	1	1	0	1	
Trying 1-bit flip:	$\overline{1}$	1	0	1	1	1	0	1	Unbalanced!
Trying 2-bit flip:	$\overline{1}$	$\overline{0}$	0	1	1	1	0	1	Unbalanced!
Trying 3-bit flip:	$\overline{1}$	$\overline{0}$	$\overline{1}$	1	1	1	0	1	Unbalanced!
Trying 4-bit flip:	$\overline{1}$	$\overline{0}$	$\overline{1}$	$\overline{0}$	1	1	0	1	Unbalanced!

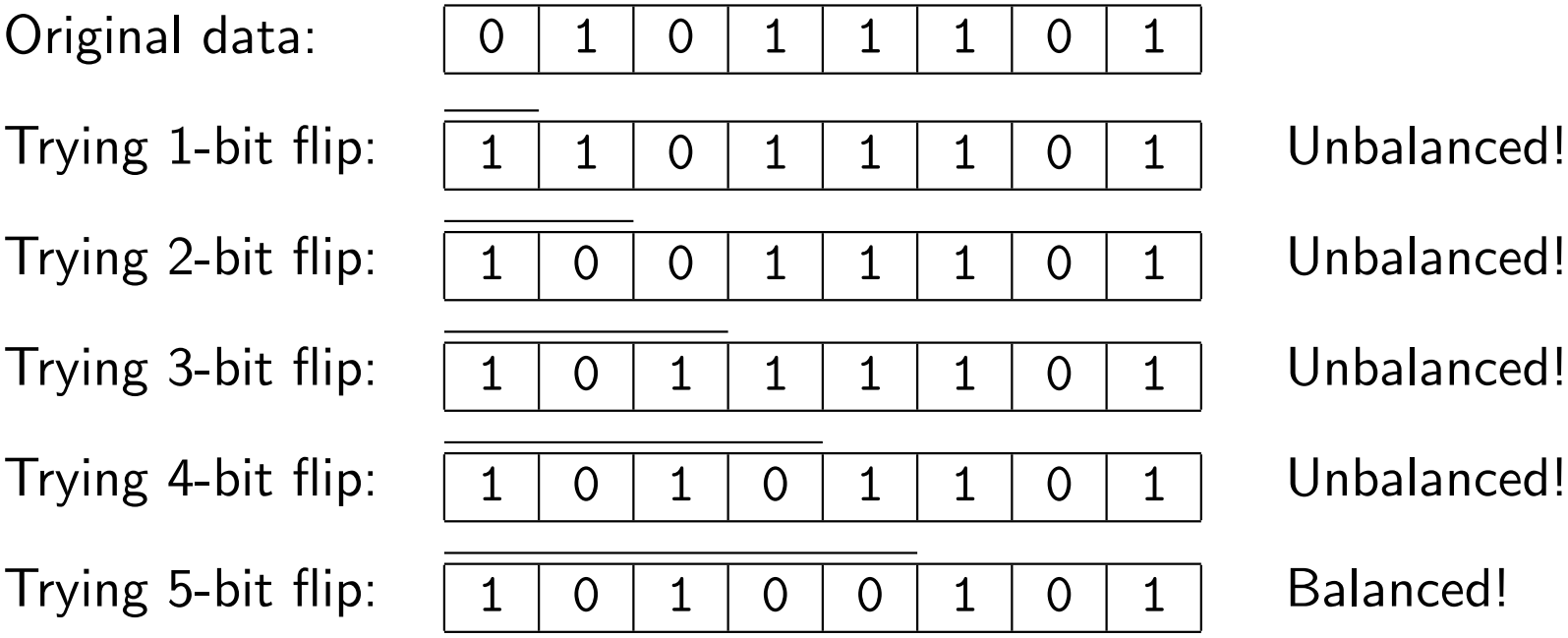
Knuth's Technique

Balancing of 01011101:



Knuth's Technique

Balancing of 01011101:



Knuth's Technique

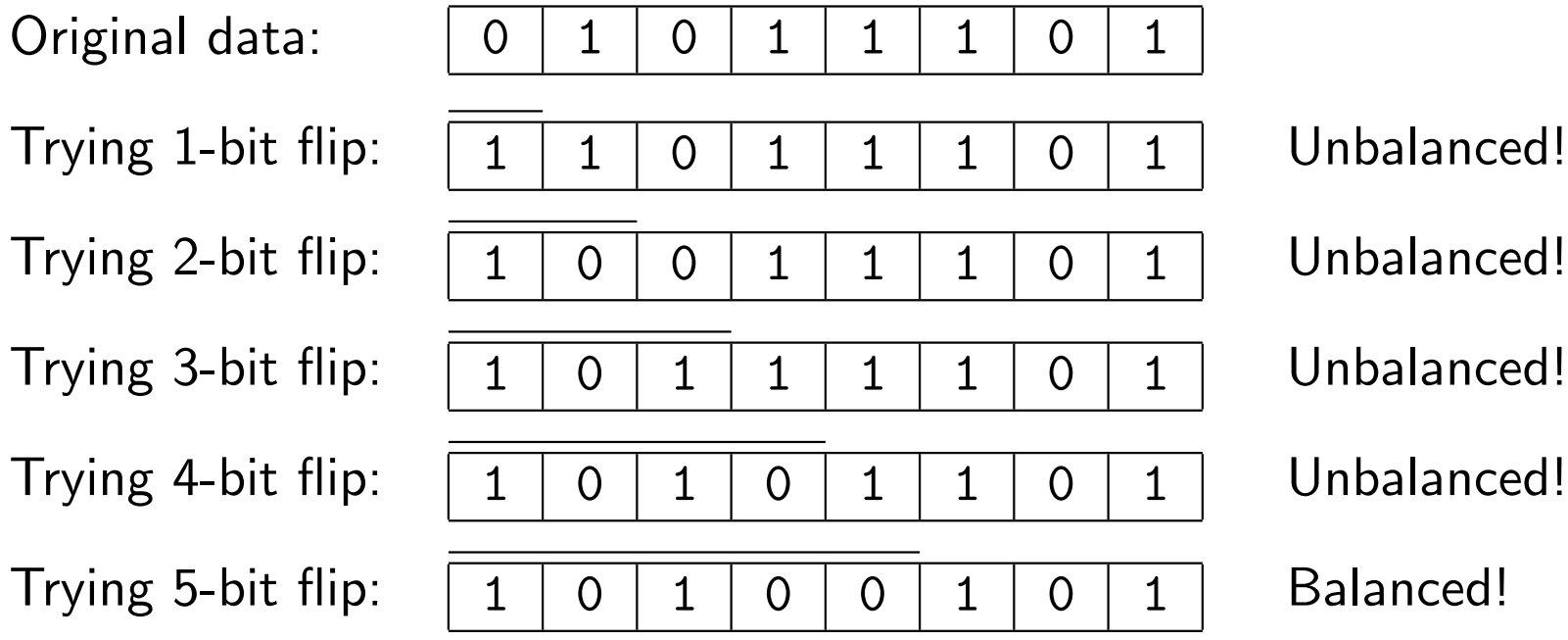
Balancing of 01011101:

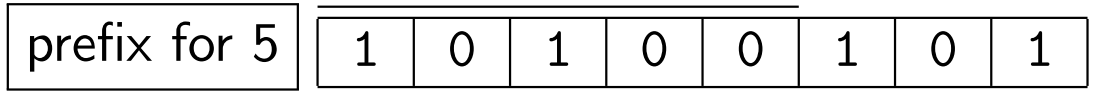
Original data:	0 1 0 1 1 1 0 1	
Trying 1-bit flip:	$\overline{1}$ 1 0 1 1 1 0 1	Unbalanced!
Trying 2-bit flip:	$\overline{1\ 0}$ 0 1 1 1 0 1	Unbalanced!
Trying 3-bit flip:	$\overline{1\ 0\ 1}$ 1 1 1 0 1	Unbalanced!
Trying 4-bit flip:	$\overline{1\ 0\ 1\ 0}$ 1 1 0 1	Unbalanced!
Trying 5-bit flip:	$\overline{1\ 0\ 1\ 0\ 0}$ 1 0 1	Balanced!

Encoding: prefix for 5 $\overline{1\ 0\ 1\ 0\ 0}$ 1 0 1

Knuth's Technique

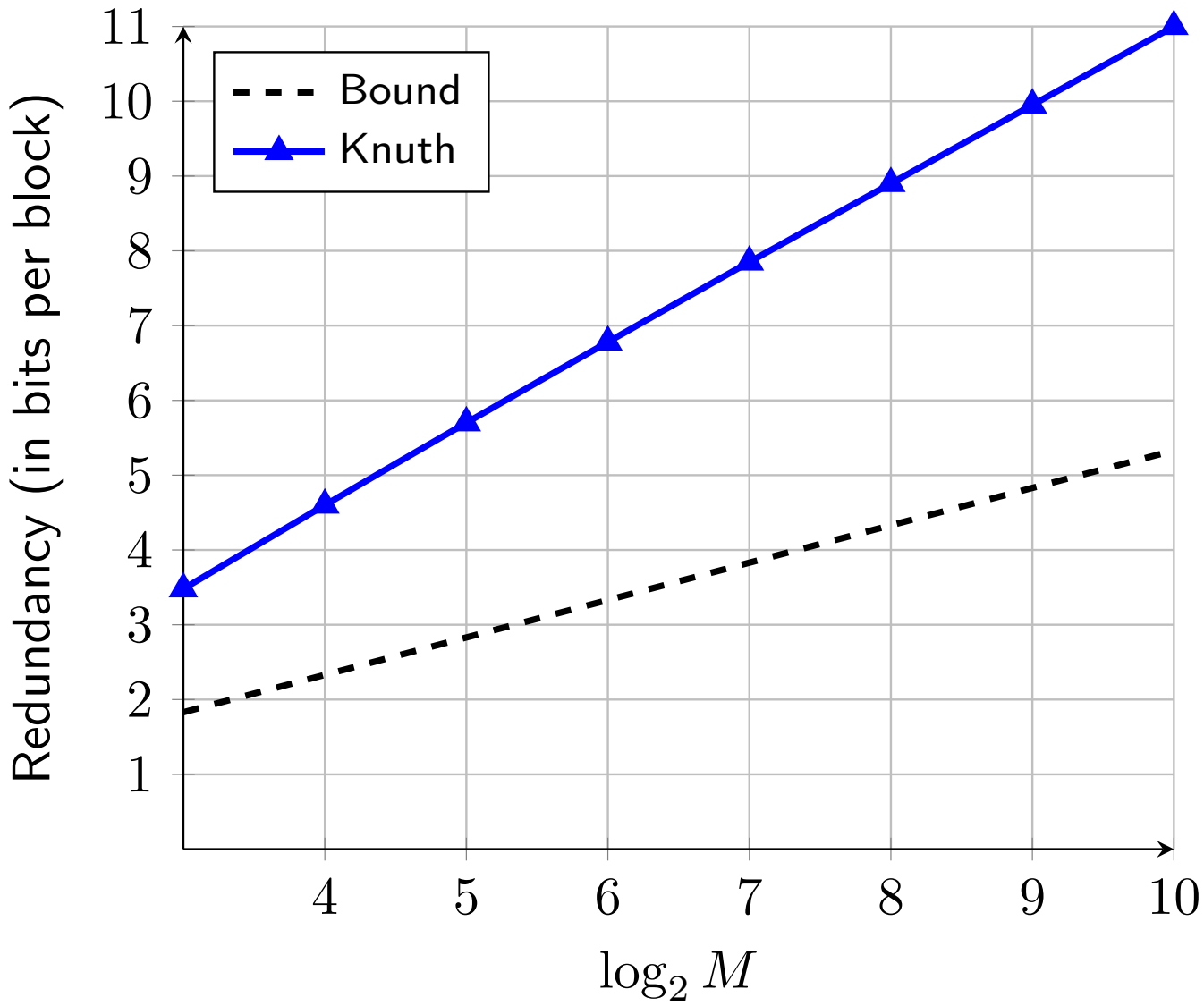
Balancing of 01011101:





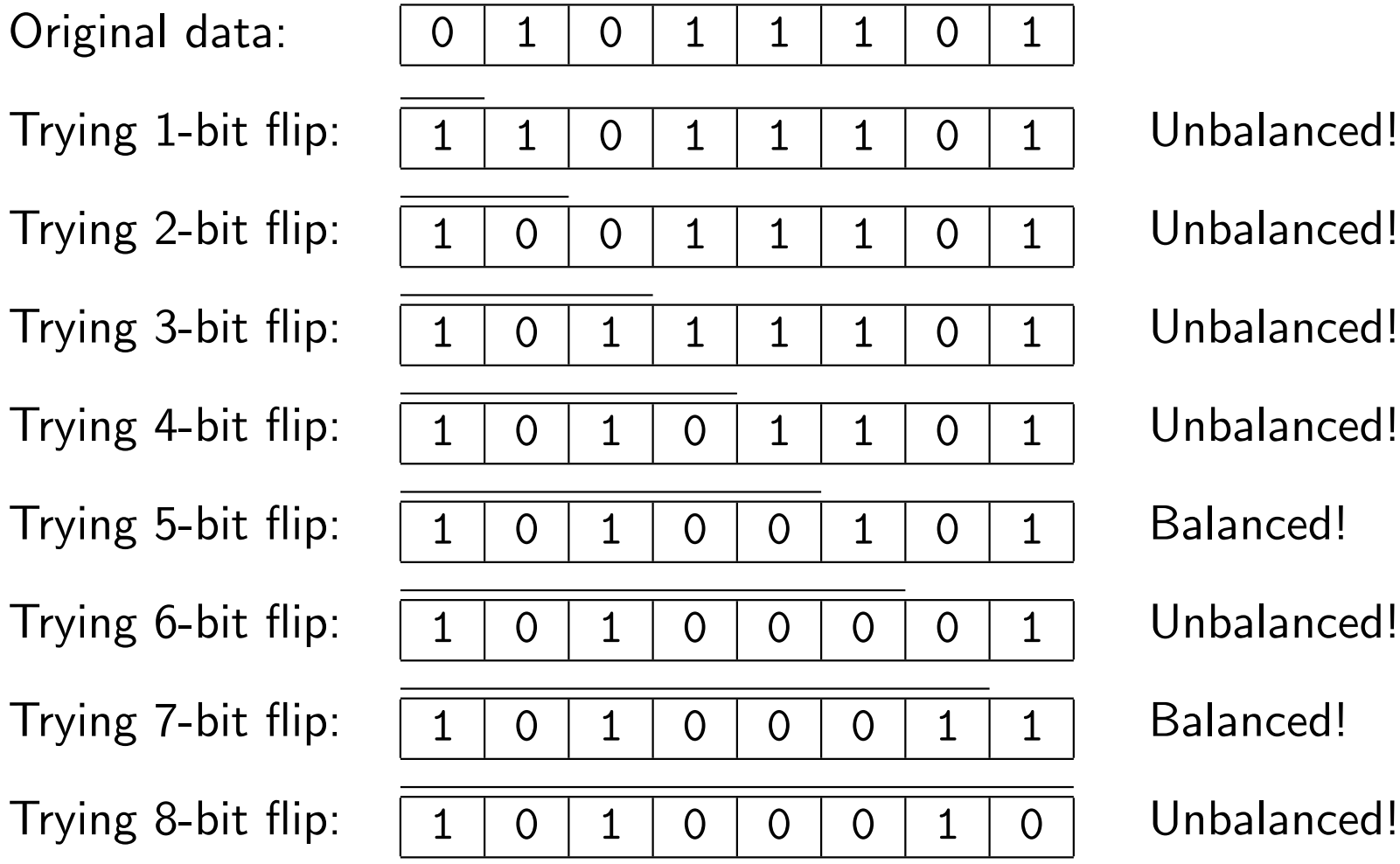
The "prefix for 5" must be a balanced block too (of size 6 here)

Redundancy


Bound
Knuth
Redundancy (in bits per block)
11
10
9
8
7
6
5
4
3
2
1
4
5
6
7
8
9
10
$\log_2 M$

Immink and Weber's Technique

They consider all possible flip points.



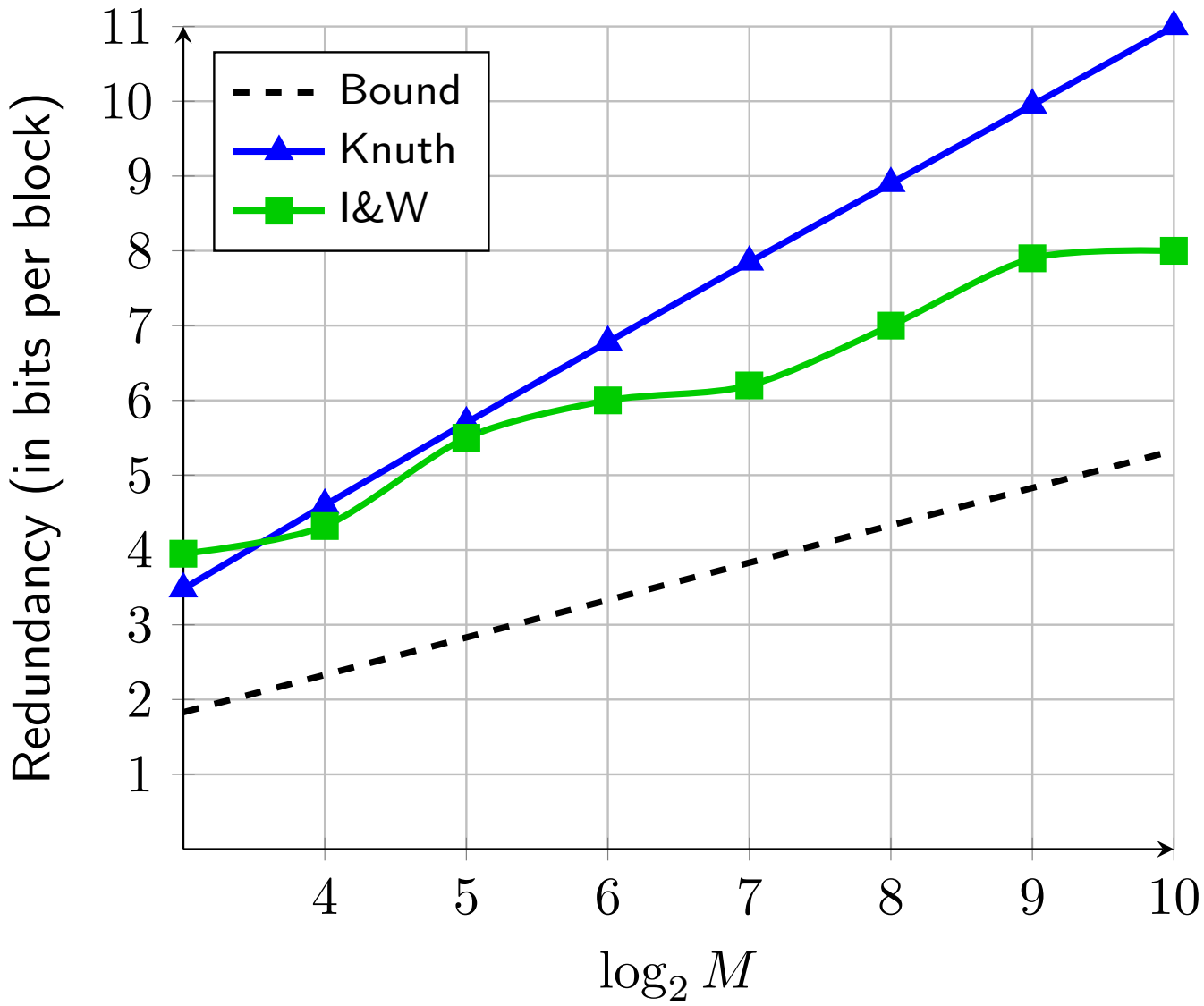
Immink and Weber's Technique

Observation: some source blocks have more than one flip point.

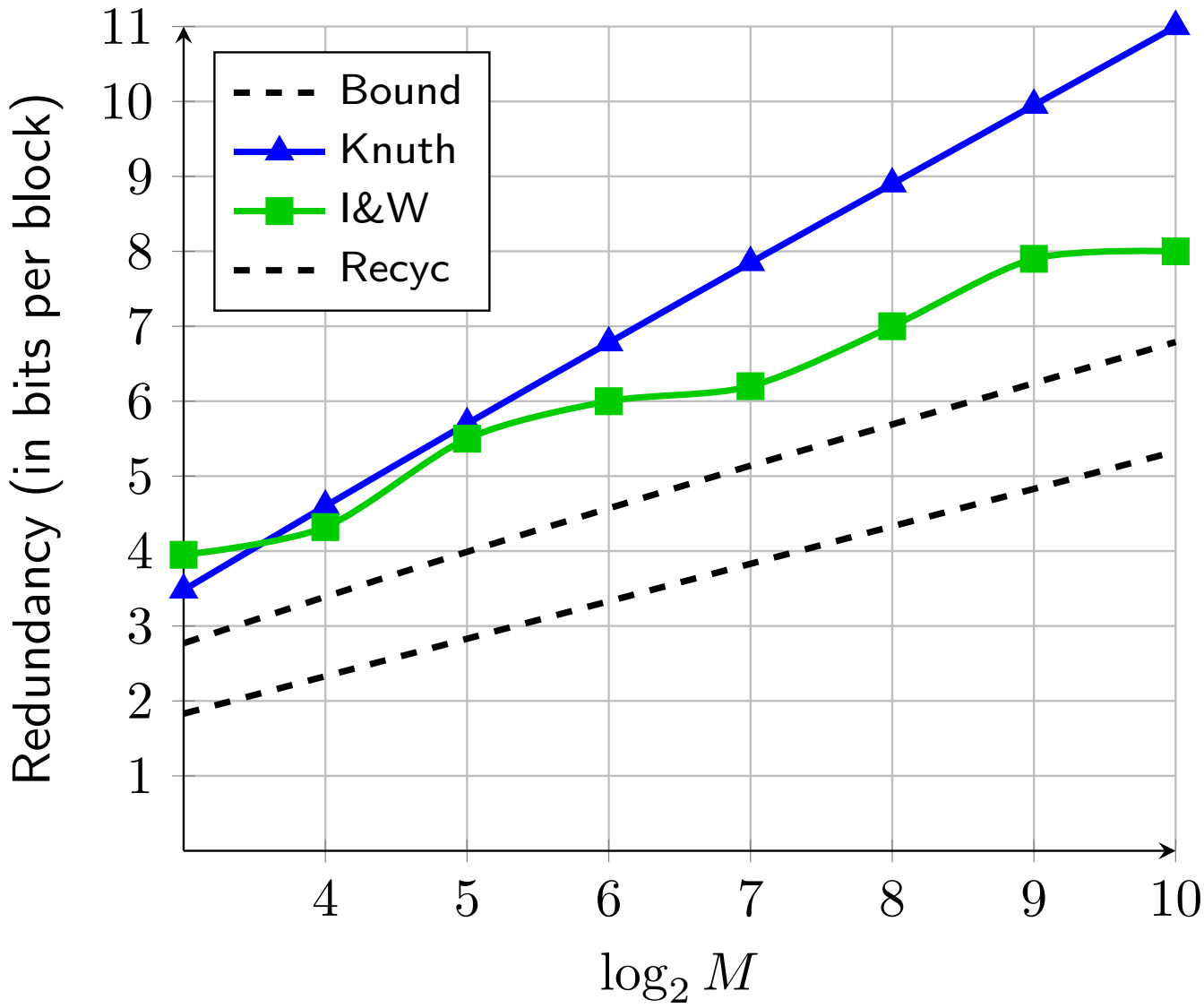
There exist between 1 and $Q/2$ adequate flip points.

They proposed a rather complex construction technique to take advantage of these multiple flip points.

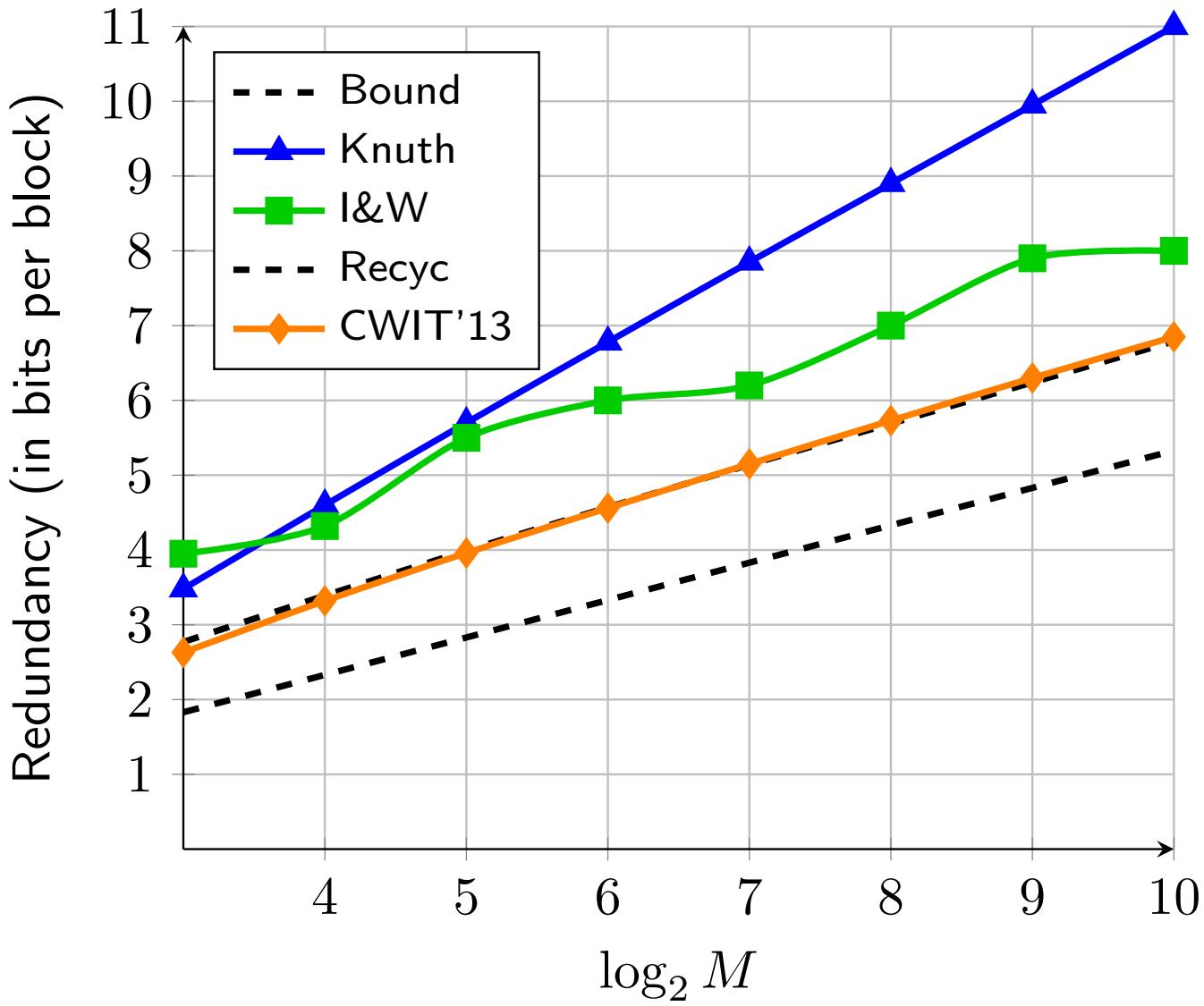
Redundancy


Redundancy (in bits per block)
11
10
9
8
7
6
5
4
3
2
1
Bound
Knuth
I&W
4
5
6
7
8
9
10
$\log_2 M$

Redundancy


Redundancy (in bits per block)
11
10
9
8
7
6
5
4
3
2
1
Bound
Knuth
I&W
Recyc
4
5
6
7
8
9
10
$\log_2 M$

Redundancy


Redundancy (in bits per block)
11
10
9
8
7
6
5
4
3
2
1
Bound
Knuth
I&W
Recyc
CWIT'13
4
5
6
7
8
9
10
$\log_2 M$

Contents of the Talk

- Introduction to balanced blocks
- Non-scalable mathematical constructions
- A few previous practical constructions

$\Rightarrow$ Our per-block optimal construction:
based on permutations, Pacman,
and small integers

- Our arbitrarily low redundancy construction

Recent Previous Work: Permutation-Based Technique [CWIT'17]

Based on:

- Permutations;
- Computations using small numbers;
- Pacman.

Resolutely departs from Knuth's Technique.

Permutation-Based Technique [CWIT'17]

Why permutations?

Let's look at an 8-element permutation.

$$(4, 1, 8, 5, 7, 6, 2, 3)$$

Now, let's look at the parity of the elements.

E O E O O E E O

Necessarily, half of the elements are even; the other half are odd.

Permutation-Based Technique [CWIT'17]

Why small numbers?

[Conventional] Enumerative Coding:

$$\underset{\in 2^{Q}}{0010\ldots0101} \quad \mapsto \quad \underset{\text{(one large integer)}}{7811\ldots2915} \quad \mapsto \quad \underset{\in \mathcal{B}_M}{1001\ldots1101}$$

"Small-integers" Enumerative Coding:

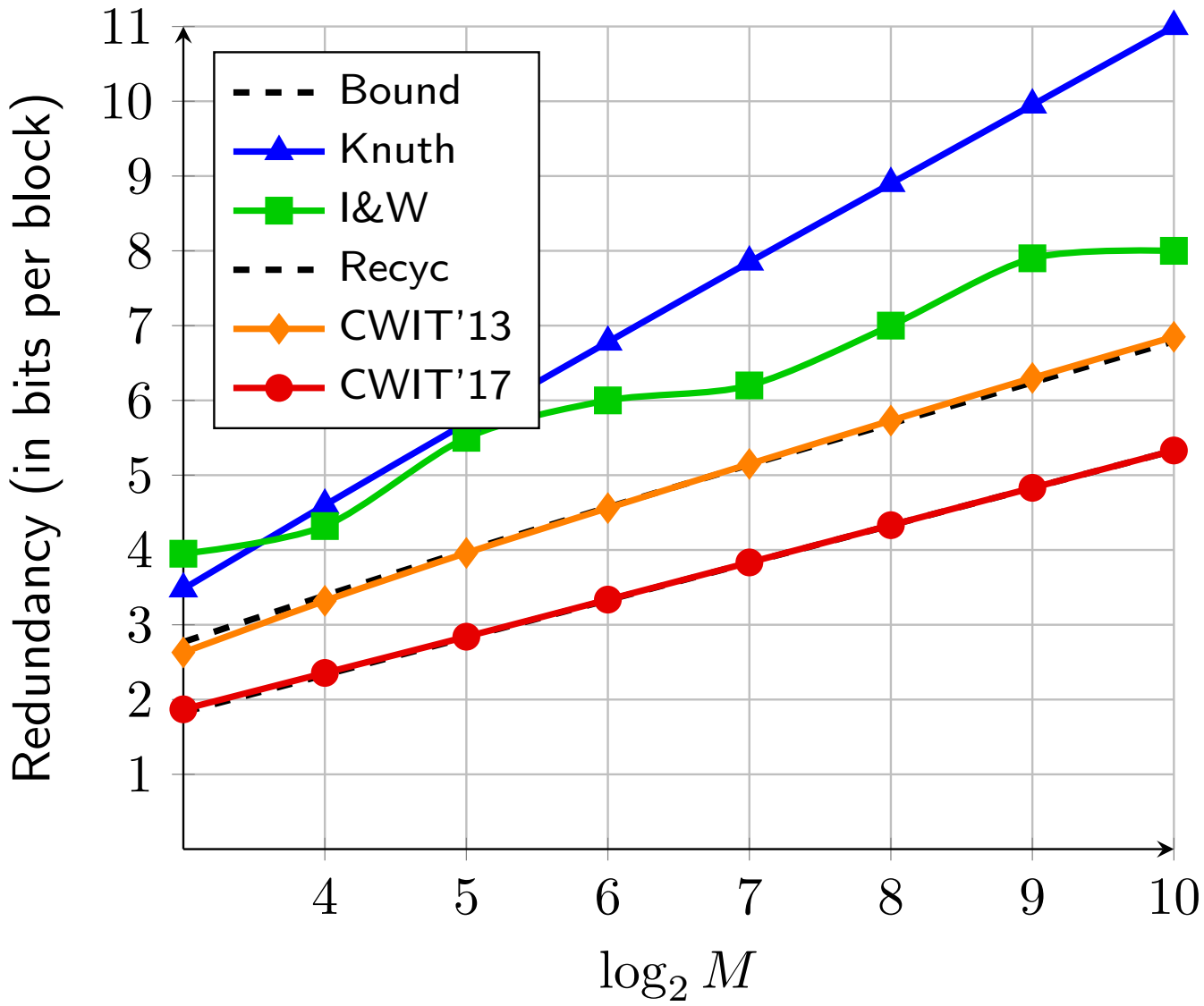
$$\underset{\in 2^{Q}}{0010\ldots0101} \quad \mapsto \quad \underset{\text{(many small integers)}}{\begin{matrix}4\\1\\2\\\ldots\\13\\5\\6\end{matrix}} \quad \mapsto \quad \underset{\in \mathcal{B}_M}{1100\ldots0010}$$

Permutation-Based Technique [CWIT'17]

Why Pacman?

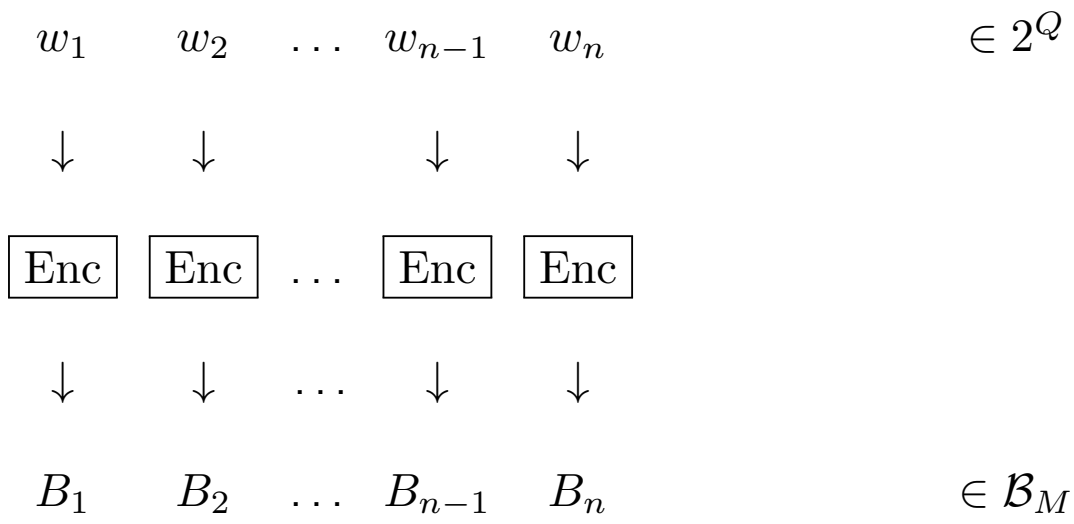
Pacman, which normally *eats* pills,
can be turned into a device
that both *consumes* and *produces*
information pills.

Redundancy


Redundancy (in bits per block)
11
10
9
8
7
6
5
4
3
2
1
Bound
Knuth
I&W
Recyc
CWIT'13
CWIT'17
4
5
6
7
8
9
10
$\log_2 M$

Permutation-Based Technique [CWIT'17]

Encoding process, if we omit the states:



Permutation-Based Technique [CWIT'17]

Encoding process, with states:

$$
\begin{array}{ccccccccccc}
 & w_1 & & w_2 & & \cdots & & w_{n-1} & & w_n & \in 2^Q \\
 & \downarrow & & \downarrow & & & & \downarrow & & \downarrow & \\
\xrightarrow{\Sigma_0} & \boxed{\text{Enc}} & \xrightarrow{\Sigma_1} & \boxed{\text{Enc}} & \xrightarrow{\Sigma_2} & \cdots & \xrightarrow{\Sigma_{n-2}} & \boxed{\text{Enc}} & \xrightarrow{\Sigma_{n-1}} & \boxed{\text{Enc}} & \\
 & \downarrow & & \downarrow & & \cdots & & \downarrow & & \downarrow & \\
 & B_1 & & B_2 & & \cdots & & B_{n-1} & & B_n & \in \mathcal{B}_M
\end{array}
$$

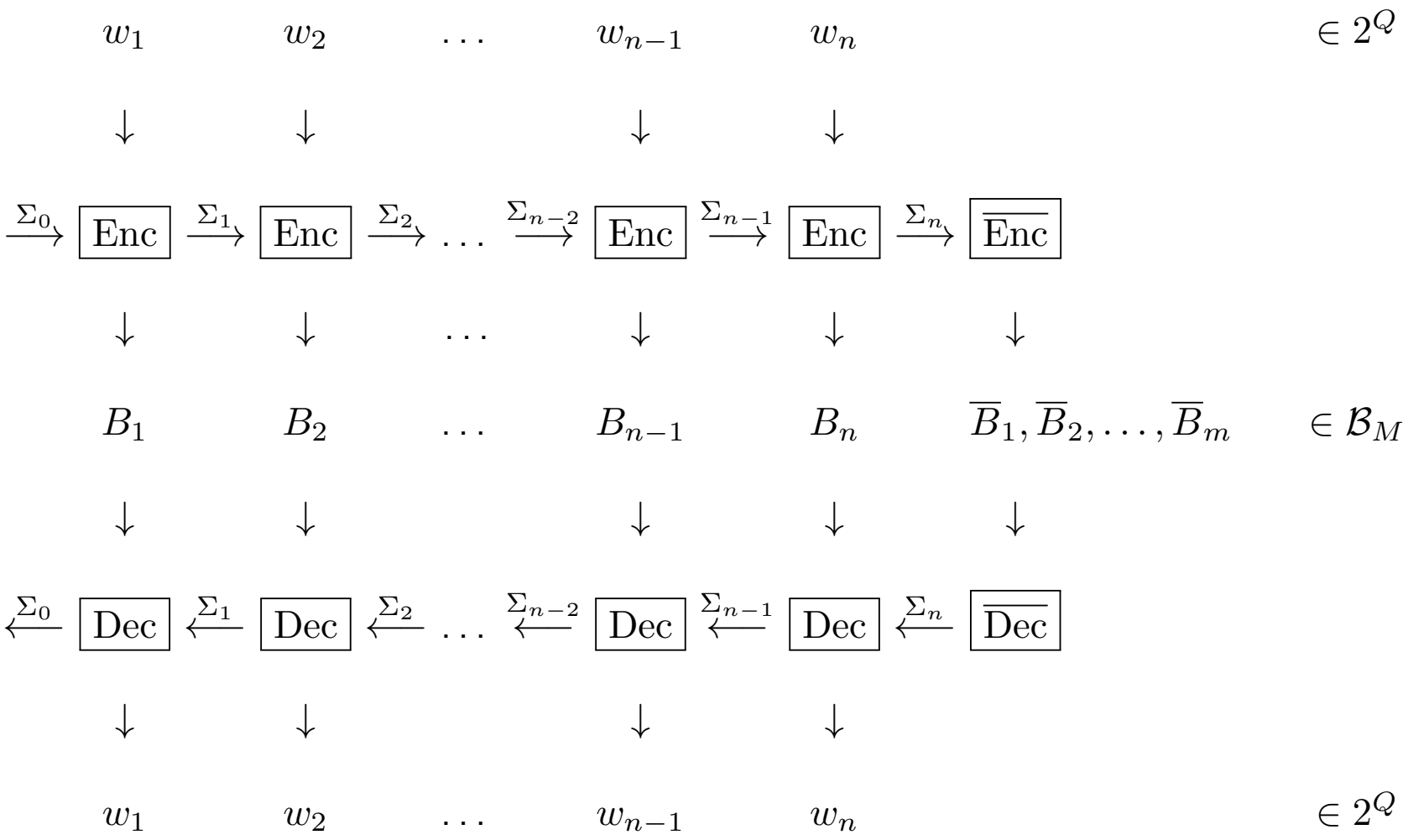
Permutation-Based Technique [CWIT'17]

Encoding process, in full:

$$
\begin{array}{ccccccccccccc}
w_1 & & w_2 & & \cdots & & w_{n-1} & & w_n & & & & \in 2^Q \\
\downarrow & & \downarrow & & & & \downarrow & & \downarrow & & & & \\
\xrightarrow{\Sigma_0} & \boxed{\text{Enc}} & \xrightarrow{\Sigma_1} & \boxed{\text{Enc}} & \xrightarrow{\Sigma_2} \cdots & \xrightarrow{\Sigma_{n-2}} & \boxed{\text{Enc}} & \xrightarrow{\Sigma_{n-1}} & \boxed{\text{Enc}} & \xrightarrow{\Sigma_n} & \boxed{\overline{\text{Enc}}} & & \\
\downarrow & & \downarrow & & \cdots & & \downarrow & & \downarrow & & \downarrow & & \\
B_1 & & B_2 & & \cdots & & B_{n-1} & & B_n & & \overline{B}_1, \overline{B}_2, \ldots, \overline{B}_m & & \in \mathcal{B}_M
\end{array}
$$

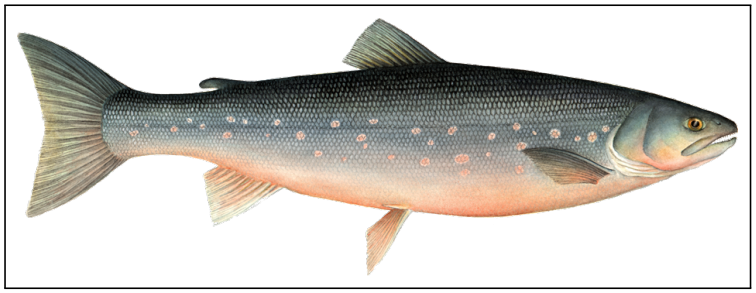
Permutation-Based Technique [CWIT'17]

Encoding and decoding process:



Permutation-Based Technique [CWIT'17]

WARNING! Information transformation in [variants of] Knuth's technique:



Source: seafoodwatch.org

$\mapsto$

Source: metro.co.uk

and in our permutation-based technique:

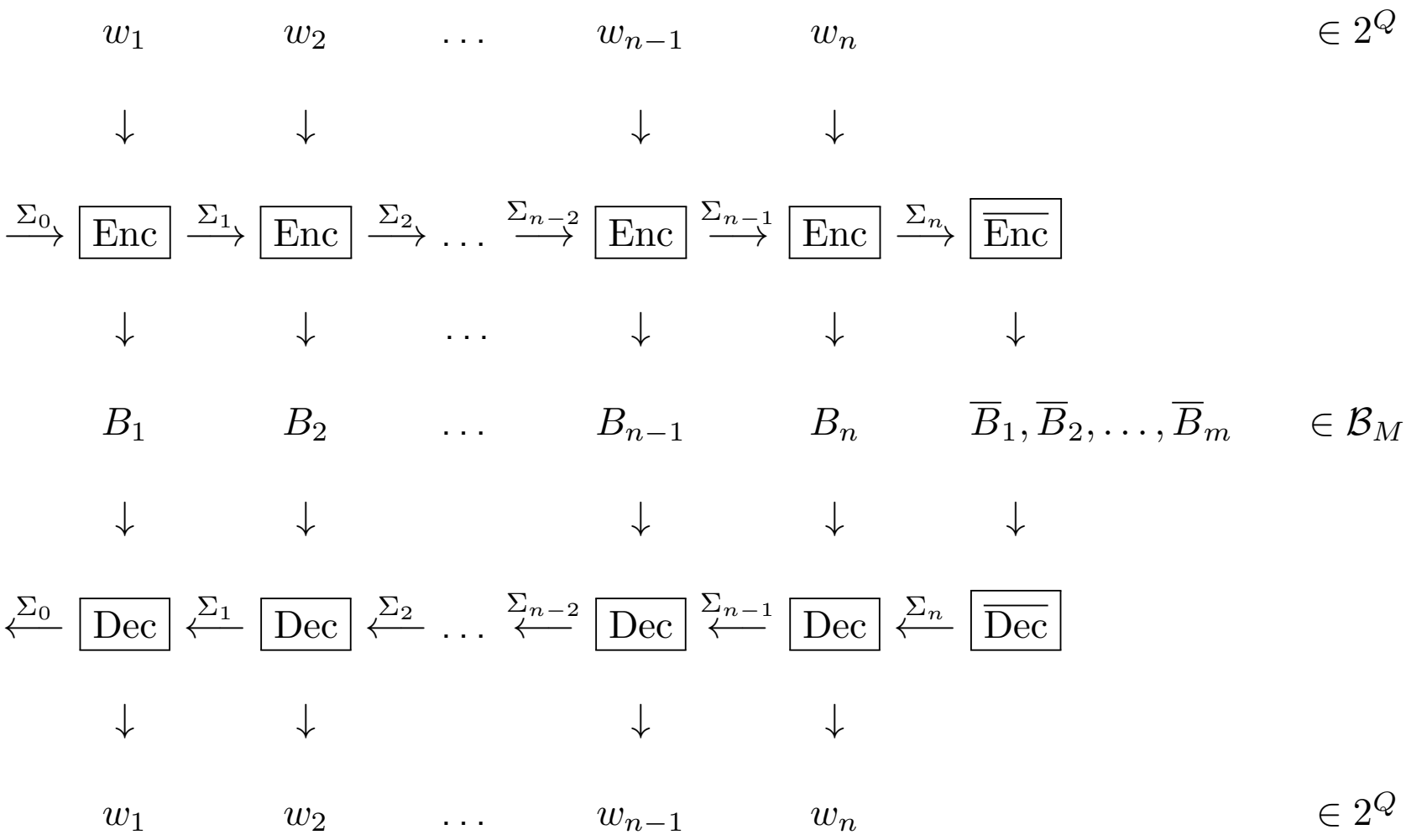
Source: farmersweekly.co.za

$\mapsto$

Source: zacmeat.com.sg

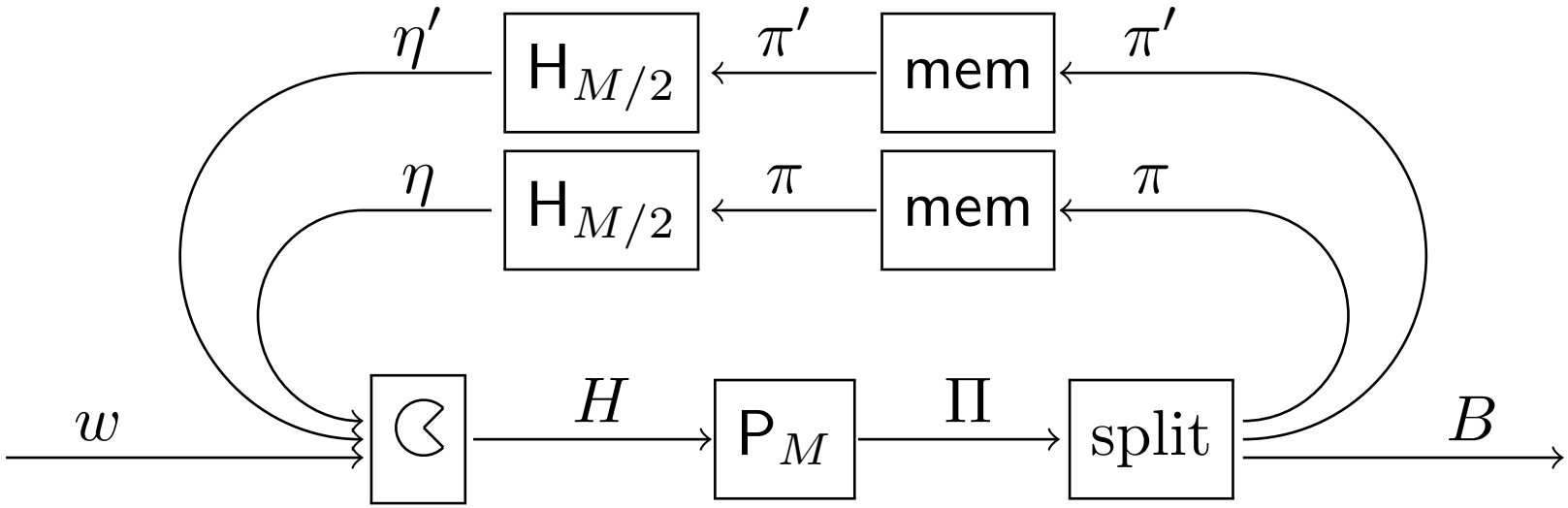
Permutation-Based Technique [CWIT'17]

Encoding and decoding process:



Permutation-Based Technique [CWIT'17]

Information flow inside of function 'Enc':



Permutation-Based Technique [CWIT'17]

Correspondence between the permutations in $\mathcal{P}_4$ and $\mathcal{H}_4$:

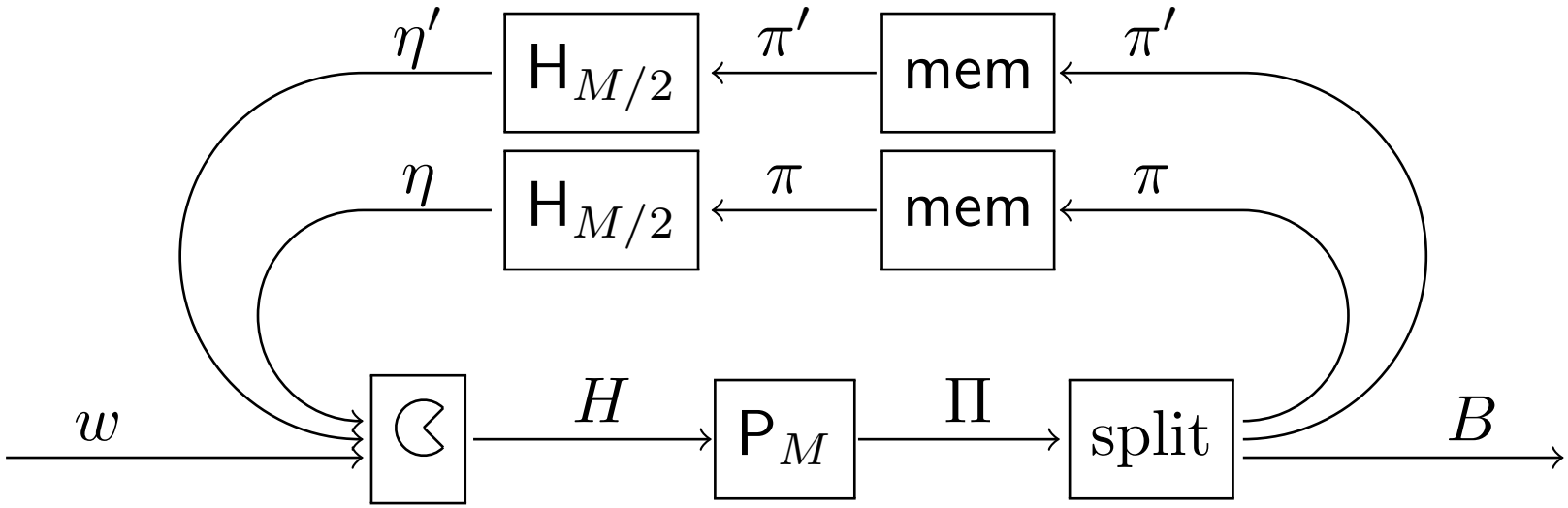
Convent.	Indexed
$(1,2,3,4)$	$\langle 1,2,3,4\rangle$
$(1,2,4,3)$	$\langle 1,2,3,3\rangle$
$(1,3,2,4)$	$\langle 1,2,2,4\rangle$
$(1,3,4,2)$	$\langle 1,2,2,3\rangle$
$(1,4,2,3)$	$\langle 1,2,3,2\rangle$
$(1,4,3,2)$	$\langle 1,2,2,2\rangle$
$(2,1,3,4)$	$\langle 1,1,3,4\rangle$
$(2,1,4,3)$	$\langle 1,1,3,3\rangle$

Convent.	Indexed
$(2,3,1,4)$	$\langle 1,1,2,4\rangle$
$(2,3,4,1)$	$\langle 1,1,2,3\rangle$
$(2,4,1,3)$	$\langle 1,1,3,2\rangle$
$(2,4,3,1)$	$\langle 1,1,2,2\rangle$
$(3,1,2,4)$	$\langle 1,2,1,4\rangle$
$(3,1,4,2)$	$\langle 1,2,1,3\rangle$
$(3,2,1,4)$	$\langle 1,1,1,4\rangle$
$(3,2,4,1)$	$\langle 1,1,1,3\rangle$

Convent.	Indexed
$(3,4,1,2)$	$\langle 1,2,1,2\rangle$
$(3,4,2,1)$	$\langle 1,1,1,2\rangle$
$(4,1,2,3)$	$\langle 1,2,3,1\rangle$
$(4,1,3,2)$	$\langle 1,2,2,1\rangle$
$(4,2,1,3)$	$\langle 1,1,3,1\rangle$
$(4,2,3,1)$	$\langle 1,1,2,1\rangle$
$(4,3,1,2)$	$\langle 1,2,1,1\rangle$
$(4,3,2,1)$	$\langle 1,1,1,1\rangle$

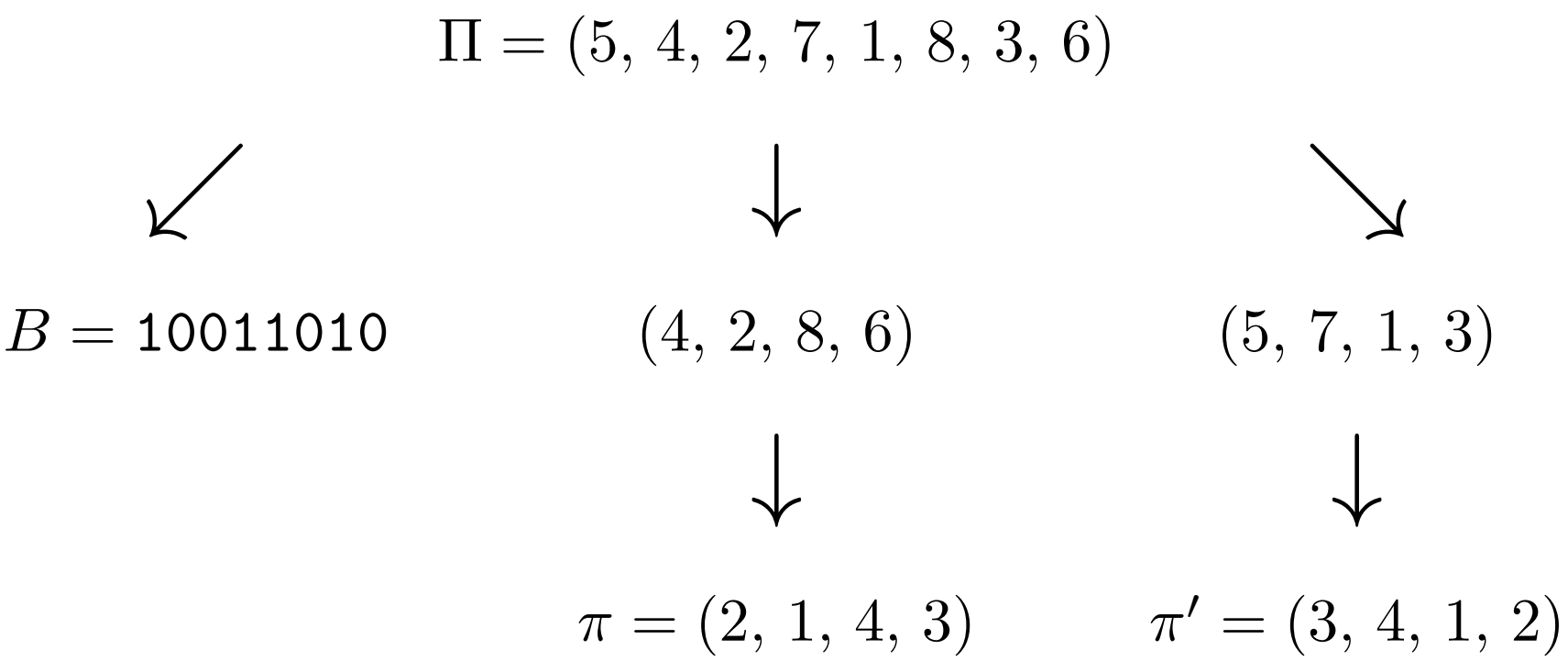
Permutation-Based Technique [CWIT'17]

Information flow inside of function 'Enc':



Permutation-Based Technique [CWIT'17]

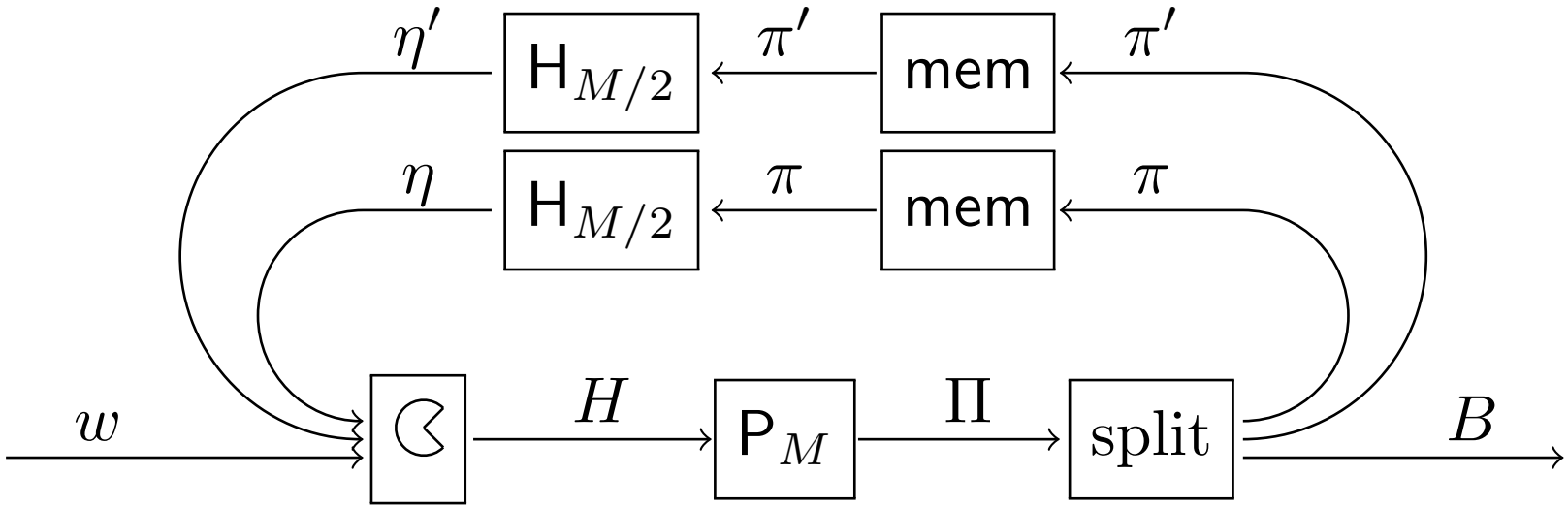
Illustration of the operation 'split':



$(B, \pi, \pi') = \text{split}(\Pi)$

Permutation-Based Technique [CWIT'17]

Information flow inside of function 'Enc':



Permutation-Based Technique [CWIT'17]

Progressive information transfer during the execution of Pacman's programming:

	Before :	*During* :	*After* :
w	$b_1 \;\; b_2 \;\; b_3 \;\; \ldots \;\; b_Q$	$b_1 \; \square_2 \;\; b_3 \;\; \ldots \;\; b_Q$	$\square_1 \; \square_2 \; \square_3 \;\; \ldots \;\; \square_Q$
η	$\langle \iota_1, \; \iota_2, \; \iota_3, \ldots, \iota_{M/2} \rangle$	$\langle \square_1, \; \iota_2, \square_3, \ldots, \; \iota_{M/2} \rangle$	$\langle \square_1, \square_2, \square_3, \ldots, \square_{M/2} \rangle$
η'	$\langle \iota'_1, \; \iota'_2, \; \iota'_3, \ldots, \iota'_{M/2} \rangle$	$\langle \square_1, \square_2, \; \iota'_3, \ldots, \square_{M/2} \rangle$	$\langle \square_1, \square_2, \square_3, \ldots, \square_{M/2} \rangle$
$\mapsto$	$\mapsto$	$\mapsto$	$\mapsto$
H	$\langle \square_1, \square_2, \square_3, \ldots, \; \square_M \rangle$	$\langle \square_1, \; \iota''_2, \; \iota''_3, \ldots, \quad \square_M \rangle$	$\langle \iota''_1, \; \iota''_2, \; \iota''_3, \ldots, \quad \iota''_M \rangle$

Permutation-Based Technique [CWIT'17]

Effect of the instructions on Pacman's memory size:

Consumption (B_i, E_i, or O_i):	Production (L_i):
Before : $\ldots$  b_i $\ldots$	*Before* : $\ldots$  $\square$ $\ldots$
After : $\ldots$ $\square$  $\ldots$	*After* : $\ldots$ ι_i'' $\ldots$

Consumption:

$$\{1, \ldots, \sigma\} \times \{1, \ldots, \rho\} \quad \rightarrow \quad \{1, \ldots, \sigma \cdot \rho\}$$

Production:

$$\{1, \ldots, \sigma\} \quad \rightarrow \quad \{1, \ldots, \lceil \sigma/\rho \rceil\} \times \{1, \ldots, \rho\}$$

Permutation-Based Technique [CWIT'17]

A programming $\mathbb{P}$ is a sequence of $2 \times M + Q$ instructions.

Each of the following instructions has to appear exactly once in $\mathbb{P}$:

- $\mathsf{E}_1, \ldots, \mathsf{E}_{M/2}$ (for the consumption of an index of η),
- $\mathsf{O}_1, \ldots, \mathsf{O}_{M/2}$ (for the consumption of an index of η'),
- $\mathsf{B}_1, \ldots, \mathsf{B}_Q$ (for the consumption of a bit of w), and
- $\mathsf{L}_1, \ldots, \mathsf{L}_M$ (for the production of an index of H).

Permutation-Based Technique [CWIT'17]

$$\begin{array}{ccccc} w & \eta & \eta' & \mapsto & H \\ b_1\, b_2\, b_3\, b_4 & \langle \iota_1,\, \iota_2,\, \iota_3 \rangle & \langle \iota'_1,\, \iota'_2,\, \iota'_3 \rangle & \mapsto & \langle \iota''_1,\, \iota''_2,\, \iota''_3,\, \iota''_4,\, \iota''_5,\, \iota''_6 \rangle \end{array}$$

$\mathbb{P}_1 = \mathsf{B}_1, \mathsf{B}_2, \mathsf{B}_3, \mathsf{B}_4, \mathsf{E}_1, \mathsf{E}_2, \mathsf{E}_3, \mathsf{O}_1, \mathsf{O}_2, \mathsf{O}_3, \mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3, \mathsf{L}_4, \mathsf{L}_5, \mathsf{L}_6$
$\Rightarrow$ Too big a memory!

$\mathbb{P}_2 = \mathsf{L}_6, \mathsf{L}_5, \mathsf{L}_4, \mathsf{L}_3, \mathsf{L}_2, \mathsf{L}_1, \mathsf{O}_3, \mathsf{O}_2, \mathsf{O}_1, \mathsf{E}_3, \mathsf{E}_2, \mathsf{E}_1, \mathsf{B}_4, \mathsf{B}_3, \mathsf{B}_2, \mathsf{B}_1$
$\Rightarrow$ Again!

$\mathbb{P}_3 = \mathsf{B}_1, \mathsf{L}_2, \mathsf{B}_2, \mathsf{B}_3, \mathsf{L}_4, \mathsf{E}_1, \mathsf{E}_2, \mathsf{E}_3, \mathsf{L}_6, \mathsf{O}_1, \mathsf{O}_2, \mathsf{B}_4, \mathsf{L}_5, \mathsf{O}_3, \mathsf{L}_3, \mathsf{L}_1$
$\Rightarrow$ Maximal memory size: 6.

Contents of the Talk

- Introduction to balanced blocks
- Non-scalable mathematical constructions
- A few previous practical constructions
- Our per-block optimal construction:
  based on permutations, Pacman,
  and small integers

$\Rightarrow$ Our arbitrarily low redundancy construction

Contribution [ISITA'18]

Lookup table for $Q = 5$, $N = 2$, $M = 4$:

Input	Balanced
00000	0011·0011
00001	0011·0101
00010	0011·0110
00011	0011·1001
00100	0011·1010
00101	0011·1100
00110	0101·0011
00111	0101·0101
01000	0101·0110
01001	0101·1001
01010	0101·1010

Input	Balanced
01011	0101·1100
01100	0110·0011
01101	0110·0101
01110	0110·0110
01111	0110·1001
10000	0110·1010
10001	0110·1100
10010	1001·0011
10011	1001·0101
10100	1001·0110
10101	1001·1001

Input	Balanced
10110	1001·1010
10111	1001·1100
11000	1010·0011
11001	1010·0101
11010	1010·0110
11011	1010·1001
11100	1010·1010
11101	1010·1100
11110	1100·0011
11111	1100·0101

Contribution [ISITA'18]

Embedding capacity in a per-block setting [optimal]:

$$Q = \left\lfloor \log_2 \binom{M}{M/2} \right\rfloor$$

Embedding capacity in a multi-block (N) setting:

$$\frac{Q}{N} \leq \log_2 \binom{M}{M/2}$$

Contribution [ISITA'18]

Input	Balanced
00000	0011·0011
00001	0011·0101
00010	0011·0110
00011	0011·1001
00100	0011·1010
00101	0011·1100
00110	0101·0011
00111	0101·0101

Input	Balanced
01000	0101·0110
01001	0101·1001
01010	0101·1010
01011	0101·1100
01100	0110·0011
01101	0110·0101
01110	0110·0110
01111	0110·1001

Input	Balanced
10000	0110·1010
10001	0110·1100
10010	1001·0011
10011	1001·0101
10100	1001·0110
10101	1001·1001
10110	1001·1010
10111	1001·1100

Input	Balanced
11000	1010·0011
11001	1010·0101
11010	1010·0110
11011	1010·1001
11100	1010·1010
11101	1010·1100
11110	1100·0011
11111	1100·0101

Information transfer during the execution of Pacman's programming for the case $Q = 5$, $N = 2$, $M = 4$:

$$w \quad \eta_1 \quad \eta_1' \quad \eta_2 \quad \eta_2' \quad \mapsto \quad H_1 \quad H_2$$

Example program: $\mathbb{P} = \mathsf{B}_1, \mathsf{B}_2, \mathsf{B}_3, \mathsf{B}_4, \mathsf{B}_5, \mathsf{E}_{1,1}, \mathsf{E}_{1,2}, \mathsf{O}_{1,1}, \mathsf{O}_{1,2}, \mathsf{L}_{1,1}, \mathsf{L}_{1,2}, \mathsf{L}_{1,3}, \mathsf{L}_{1,4}, \mathsf{E}_{2,1}, \mathsf{E}_{2,2}, \mathsf{O}_{2,1}, \mathsf{O}_{2,2}, \mathsf{L}_{2,1}, \mathsf{L}_{2,2}, \mathsf{L}_{2,3}, \mathsf{L}_{2,4}$

Experimental Results

M	C	R_I	Q/N	R_X	$\sigma_{\max}$
4	2.585	1.415	2/1	.585	2
			5/2	.0850	8
			18/7	.0135	80
8	6.129	1.871	6/1	.129	24
			49/8	.00428	1,336
16	13.652	2.348	13/1	.652	32
			27/2	.152	64
			68/5	.0517	464
			109/8	.0267	1,264
			150/11	.0154	2,048
32	29.163	2.837	29/1	.163	288
			204/7	.0201	5,312
64	60.669	3.331	60/1	.669	320
			121/2	.169	1,164
			182/3	.00195	142,928
128	124.194	3.806	124/1	.194	4,352
			745/6	.0276	257,440
256	251.673	4.327	251/1	.673	5,632
			503/2	.173	24,064
			755/3	.00618	619,744
512	507.174	4.826	507/1	.174	78,816
			3043/6	.00688	2,604,992
1,024	1,018.674	5.326	1018/1	.674	74,592
			2037/2	.174	330,752
			3056/3	.00723	8,071,168

Conclusion

Contributions:

- Improving efficiency over that of the optimal per-block solution.
- Trying to make a clearer presentation of our permutation-based construction.

Future work:

- Applying "small-integers" enumerative coding to other constrained coding tasks.

Questions?

Web page:
http://www.ift.ulaval.ca/~dadub100/