Lecture 16: Error Correcting Codes

COMP526: Efficient Algorithms

Will Rosenbaum
University of Liverpool

Updated: November 26, 2024

Announcements

1. Programming Assignment 2 posted
   - Due 29 November – this Friday
2. No Quiz This Week!
3. Attendance Code:

38959 1

Meeting Goals

1. Finish discussion of data compression
   - Discuss Burrows-Wheeler inverse analysis
   - Recap of data compression
2. Give some remarks on Programming Assignment 2
3. Mini-unit on error correcting codes
   - Introduce error correcting codes
   - Define block codes and code distance
   - Prove lower bounds for error detection and correction
   - ~~Introduce Hamming codes~~ Thursday

Burrows-Wheeler Transform

Burrows-Wheeler Transform

From last time.

1. Start with an input string S
   - $S = \texttt{banana}\boxed{\texttt{\$}}$

terminating char
first alphabetical char.

`b  a  n  a  n  a  $`

Burrows-Wheeler Transform

From last time.

1. Start with an input string S
   - $S = \texttt{banana\$}$
2. Form all cyclic shifts of S

```
b a n a n a $
a n a n a $ b
n a n a $ b a
a n a $ b a n
n a $ b a n a
a $ b a n a n
$ b a n a n a
```

Burrows-Wheeler Transform

From last time.

1. Start with an input string S
   - $S =$ `banana$`
2. Form all cyclic shifts of S
3. Sort the cyclic shifts alphabetically

rows

b	a	n	a	n	a	$
a	n	a	n	a	$	b
n	a	n	a	$	b	a
a	n	a	$	b	a	n
n	a	$	b	a	n	a
a	$	b	a	n	a	n
$	b	a	n	a	n	a

sort

$	b	a	n	a	n	a
a	$	b	a	n	a	n
a	n	a	$	b	a	n
a	n	a	n	a	$	b
b	a	n	a	n	a	$
n	a	n	a	$	b	a
n	a	$	b	a	n	a

swap for sorting

Burrows-Wheeler Transform

From last time.

1. Start with an input string S
   - $S =$ `banana$`
2. Form all cyclic shifts of S
3. Sort the cyclic shifts alphabetically
4. Return the last **column**
   - $B =$ `annb$aa`

b	a	n	a	n	a	$
a	n	a	n	a	$	b
n	a	n	a	$	b	a
a	n	a	$	b	a	n
n	a	$	b	a	n	a
a	$	b	a	n	a	n
$	b	a	n	a	n	a

$	b	a	n	a	n	a
a	$	b	a	n	a	n
a	n	a	$	b	a	n
a	n	a	n	a	$	b
b	a	n	a	n	a	$
n	a	n	a	$	b	a
n	a	$	b	a	n	a

Burrows-Wheeler Transform

From last time.

1. Start with an input string S
   - $S = \texttt{banana\$}$
2. Form all cyclic shifts of S
3. Sort the cyclic shifts alphabetically
4. Return the last **column**
   - $B = \texttt{annb\$aa}$

Claim. This process is *reversible*

- Given B, we can find the original input S.

Inverse Burrows-Wheeler

1. Form character-index pairs

transformed text B

```
(a  0)
(n  1)
(n  2)
(b  3)
($  4)
(a  5)
(a  6)
```

Inverse Burrows-Wheeler

1. Form character-index pairs
2. Sort pairs stably alphabetically by first character

orig index

```
(a  0)          0  ($  4)
(n  1)          1  (a  0)
(n  2)          2  (a  5)
(b  3)          3  (a  6)
($  4)          4  (b  3)
(a  5)          5  (n  1)
(a  6)          6  (n  2)
```

Inverse Burrows-Wheeler

1. Form character-index pairs
2. Sort pairs stably alphabetically by first character
3. Starting with $, use index as (sorted) index of next character
4. Repeat

```
(a  0)          0  ($  4)
(n  1)          1  (a  0)
(n  2)          2  (a  5)
(b  3)          3  (a  6)
($  4)          4  (b  3)
(a  5)          5  (n  1)
(a  6)          6  (n  2)
```

banana$

Inverse Burrows-Wheeler

1. Form character-index pairs
2. Sort pairs stably alphabetically by first character
3. Starting with $, use index as (sorted) index of next character
4. Repeat

```
(a  0)        0  ($  4)
(n  1)        1  (a  0)
(n  2)        2  (a  5)
(b  3)        3  (a  6)
($  4)        4  (b  3)
(a  5)        5  (n  1)
(a  6)        6  (n  2)
```

Question. Why does this work?

Inverse Burrows-Wheeler

1. Form character-index pairs
2. Sort pairs stably alphabetically by first character
3. Starting with \$, use index as (sorted) index of next character
4. Repeat

(a	0)	0	(\$	4)	\$ b a n a n a
(n	1)	1	(a	0)	a \$ b a n a n
(n	2)	2	(a	5)	a n a \$ b a n
(b	3)	3	(a	6)	a n a n a \$ b
(\$	4)	4	(b	3)	b a n a n a \$
(a	5)	5	(n	1)	n a n a \$ b a
(a	6)	6	(n	2)	n a \$ b a n a

next

Question. Why does this work?

- c a character in B, consider c's row
- where is c's next character in S?

\$'s next char is first in its row

b's next char is a

Inverse Burrows-Wheeler

1. Form character-index pairs
2. Sort pairs stably alphabetically by first character
3. Starting with $, use index as (sorted) index of next character
4. Repeat

(a	0)
(n	1)
(n	2)
(b	3)
($	4)
(a	5)
(a	6)

0	($	4)
1	(a	0)
2	(a	5)
3	(a	6)
4	(b	3)
5	(n	1)
6	(n	2)

$	b	a	n	a	n	a	0
a	$	b	a	n	a	n	1
a	n	a	$	b	a	n	2
a	n	a	n	a	$	b	3
b	~~a~~	~~n~~	~~a~~	~~n~~	~~a~~	$	← 4
n	a	n	a	$	b	a	
n	a	$	b	a	n	a	

($ 4)
4

Question. Why does this work?

- *c* a character in *B*, consider *c*'s row
- where is *c*'s next character in *S*?
- when we sort the last column, *c*'s next character ends up in *c*'s original row!

BWT Discussion

What do we know about the Burrows-Wheeler Transform?

- Running time $\Theta(n)$ — "Naive" alg. $\Theta(n^2 \log n)$ Slowwwww.
    - encoding uses **suffix sorting** (future reference)
    - decoding can be done in $\Theta(n)$ time with counting sort
    - decoding is simpler/faster!
- Typically slower than other methods
- Needs access to entire text (or apply to smaller blocks)
- ~~WBT~~ BWT → MTF → RLE → Huffman has great compression! ←

move to front

run length encoding

Huffman

Summary of Compression

Huffman	Variable-width, single-character (optimal in this case)
RLE	Variable-width, multiple-character encoding
LZW	Adaptive, fixed-width, multiple-character encoding Augments dictionary with repeated substrings
MTF	Adaptive, transforms to smaller integers should be followed by variable-width integer encoding
BWT	Block compression method, should be followed by MTF

Summary of Compression

Huffman	Variable-width, single-character (optimal in this case)
RLE	Variable-width, multiple-character encoding
LZW	Adaptive, fixed-width, multiple-character encoding Augments dictionary with repeated substrings
MTF	Adaptive, transforms to smaller integers should be followed by variable-width integer encoding
BWT	Block compression method, should be followed by MTF

Going farther. Compression is an active area of research!

- Improved compression schemes can have immediate impact.
- **Hutter Prize** 5,000 euro per 1% improvement of compression of a single 1GB English text file (from Wikipedia).
  - made to encourage research in artificial intelligence

Summary of Compression

Huffman	Variable-width, single-character (optimal in this case)
RLE	Variable-width, multiple-character encoding
LZW	Adaptive, fixed-width, multiple-character encoding Augments dictionary with repeated substrings
MTF	Adaptive, transforms to smaller integers should be followed by variable-width integer encoding
BWT	Block compression method, should be followed by MTF

Going farther. Compression is an active area of research!

- Improved compression schemes can have immediate impact.
- **Hutter Prize** 5,000 euro per 1% improvement of compression of a single 1GB English text file (from Wikipedia).
  - made to encourage research in artificial intelligence
  - what does compression have to do with AI?

Programming Assignment 2

Your Assignment

Three Pieces:

- $B = B[0..20)$ the correct solutions to the exam
  - expressed in binary 1 for true, 0 for false
  - known only to your hacker friend
- $M = M[0..10)$ the message your friend sends you
  - also expressed in binary
- $A = A[0..20)$ the answers your record for the exam, in binary

Exam answers
0110 0111 ..

Two Procedures:

- *Encode* the correct exam solutions B to a message M
  - preformed by your hacker friend
- *Decode* the message M to exam solutions A
  - performed by you during the exam

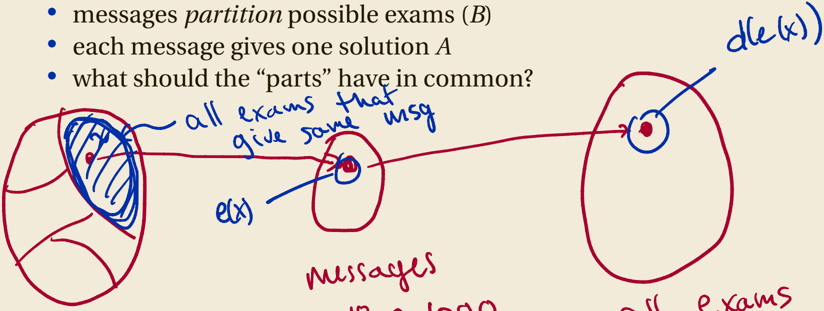
One Goal: Achieve the maximum *guaranteed* score.

- $\underline{20} - \boxed{\max\left\{d_H(A,B) \,\middle|\, B \in \{0,1\}^{20}\right\}}$
- $d_H(A, B)$ is **Hamming distance** = number of indices where solutions differ

incorrect sol'n achieved
~~worst score~~
over all exams

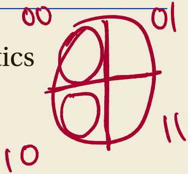
Two Suggestions

1. Abstract away from algorithms and message semantics
    - messages *partition* possible exams (B)
    - each message gives one solution A
    - what should the "parts" have in common?



Want: make partition of exams s.t. for each <u>part</u> there is one solution that is close to all exams in part

Two Suggestions



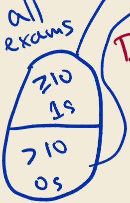
1. Abstract away from algorithms and message semantics
   - messages *partition* possible exams (B)
   - each message gives one solution A
   - what should the “parts” have in common?

Example. It is possible to guarantee a score of 10 with only *a single bit message!* (How?)

Majority:

msg 1: if # 1s ≥ 10] encode

msg 0: if # 0s > 10



Decoding:

1 → respond (1111111 ... 1) ←

0 → respond (00.... 0) ←

Two Suggestions

1. Abstract away from algorithms and message semantics
   - messages *partition* possible exams (B)
   - each message gives one solution A
   - what should the "parts" have in common?

   Example. It is possible to guarantee a score of 10 with only *a single bit message!* (How?)

2. Consider concrete smaller cases
   - what is special about 20 and 10?
   - try solving the problem by hand for smaller cases: 2 questions, 1-bit message, etc.

Error Correcting Codes

Motivation: Noisy Communication

Implicit Assumptions. So far:

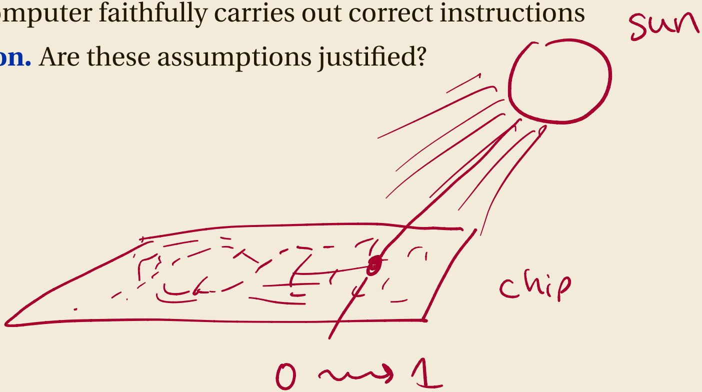
- Data is never corrupted
- Computer faithfully carries out correct instructions

Motivation: Noisy Communication

Implicit Assumptions. So far:

- Data is never corrupted
- Computer faithfully carries out correct instructions

Question. Are these assumptions justified?



Aerospace

Motivation: Noisy Communication

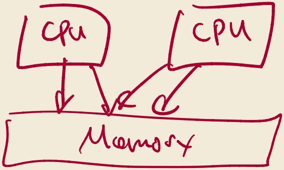
Implicit Assumptions. So far:

- Data is never corrupted
- Computer faithfully carries out correct instructions

Question. Are these assumptions justified?

Weak Point. Communication

- reading from disk
- writing to shared memory
- sending data between processors
- sending data between cities? countries? continents? planets?



Motivation: Noisy Communication

Implicit Assumptions. So far:

- Data is never corrupted
- Computer faithfully carries out correct instructions

Question. Are these assumptions justified?

Weak Point. Communication

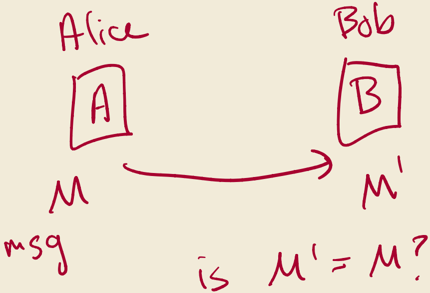
- reading from disk
- writing to shared memory
- sending data between processors
- sending data between cities? countries? continents? planets?

Question. How do we deal with errors in communication?

Goals for Error Correction

Two Goals:

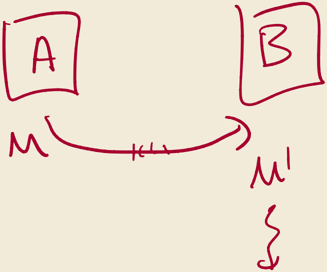
- **Detect** errors in communication
  - Given the sent (intended) message M and received message M', how can we determine if $M \neq M'$



Goals for Error Correction

Two Goals:

- **Detect** errors in communication
  - Given the sent (intended) message M and received message M', how can we determine if $M \neq M'$
- **Correct** errors automatically
  - Given received message M', how could we *automatically* determine the sent message M even if $M' \neq M$?



find M
from M'
automatically

Goals for Error Correction

Two Goals:

- **Detect** errors in communication
  - Given the sent (intended) message M and received message M', how can we determine if $M \neq M'$
- **Correct** errors automatically
  - Given received message M', how could we *automatically* determine the sent message M even if $M' \neq M$?

Question. How much noise can the system tolerate?

- Some *redundancy* is necessary.

Observation. Can't do anything w/ unaltered msgs.

$\downarrow$

must add additional info $\rightsquigarrow$ redundancy

Goals for Error Correction

single bit flipped in Msg

Alice sends → $b_0 b_1 \cdots - \cdots b_{99}$

Two Goals:

- **Detect** errors in communication
  - Given the sent (intended) message M and received message M', how can we determine if $M \neq M'$
- **Correct** errors automatically
  - Given received message M', how could we *automatically* determine the sent message M even if $M' \neq M$?

Question. How much noise can the system tolerate?

- Some *redundancy* is necessary.

PollEverywhere Question

Suppose we wish to send a string S of size 100 bits. How many additional bits must we send to **detect** a 1 bit error in the transmitted message?

pollev.com/comp526

Goals for Error Correction

Two Goals:

- **Detect** errors in communication
  - Given the sent (intended) message M and received message M', how can we determine if $M \neq M'$
- **Correct** errors automatically
  - Given received message M', how could we *automatically* determine the sent message M even if $M' \neq M$?

Question. How much noise can the system tolerate?

- Some *redundancy* is necessary.

PollEverywhere Question

Suppose we wish to send a string S of size 100 bits. How many additional bits must we send to correct ~~detect~~ a 1 bit error in the transmitted message?

pollev.com/comp526

Modeling Errors and Correction

Model & Block Codes

Communcation Model.

arb length bitstream

- Goal: send a text $S \in \{0,1\}^*$ (bitstream) across a communication channel

Model & Block Codes

Communcation Model.

- Goal: send a text $S \in \{0,1\}^*$ (bitstream) across a communication channel
- Any bit transmitted through the channel might **flip**
  - $0 \mapsto 1$ or $1 \mapsto 0$
  - *no* erasures or insertions

sent $0\,1\,1\,0\,1\,\boxed{1}\,0\,0\,0\,1\,\boxed{1}\,1$

$\downarrow$

rec'd $0\,1\,1\,0\,1\,\boxed{0}\,0\,0\,0\,1\,\boxed{0}\,1$

Model & Block Codes

Communcation Model.

- Goal: send a text $S \in \{0,1\}^*$ (bitstream) across a communication channel
- Any bit transmitted through the channel might **flip**
    - $0 \mapsto 1$ or $1 \mapsto 0$
    - *no* erasures or insertions
- To cope with errors:
    - compute and send an encoded bitstream $C(S)$
    - receiver decodes C to get S

0110

$\downarrow$ encoded

00111100

$\downarrow$ sent

01110100

$\downarrow$ decoded

0110

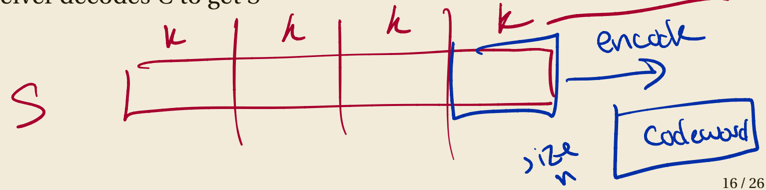
Model & Block Codes

Communcation Model.

- Goal: send a text $S \in \{0,1\}^*$ (bitstream) across a communication channel
- Any bit transmitted through the channel might **flip**
  - $0 \mapsto 1$ or $1 \mapsto 0$
  - *no* erasures or insertions
- To cope with errors:
  - compute and send an encoded bitstream $C(S)$
  - receiver decodes C to get S

Block Codes. Assumptions

- Messages consists of fixed sized blocks
  - k = **message length**
  - $m \in \{0,1\}^k$
- Encode each message separate as $C(m) \in \{0,1\}^n$
  - $C(m)$ is **codeword** for m
- n is the **block length**



Hamming Distance

Definition. Given two texts $x, y \in \{0,1\}^n$, the **Hamming distance** $d_H(x,y)$ between x and y is the number of indices at which x and y differ.

Hamming Distance

Definition. Given two texts $x, y \in \{0,1\}^n$, the **Hamming distance** $d_H(x, y)$ between x and y is the number of indices at which x and y differ.

1001011001
1011010101

$3 = d_H(x, y)$

PollEverywhere Question

What is the Hamming distance between `1001011001` and `1011010101`?

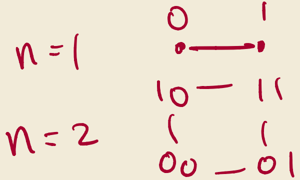
`pollev.com/comp526`

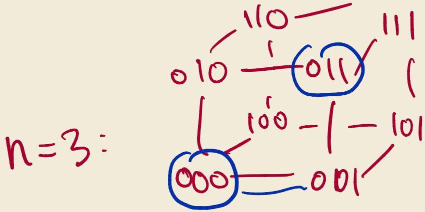
Hamming Distance

Definition. Given two texts $x, y \in \{0,1\}^n$, the **Hamming distance** $d_H(x, y)$ between x and y is the number of indices at which x and y differ.

Geometric View. Hamming distance allows us to think about binary strings *geometrically*.

- **Hamming cube** of dimension n is the set of all bit strings x of length n
  - x and y are neighbors if they differ on exactly one bit

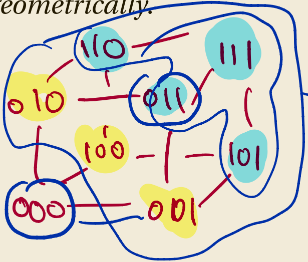




Hamming Distance

Definition. Given two texts $x, y \in \{0,1\}^n$, the **Hamming distance** $d_H(x, y)$ between x and y is the number of indices at which x and y differ.

Geometric View. Hamming distance allows us to think about binary strings *geometrically*.



- **Hamming cube** of dimension n is the set of all bit strings x of length n
  - x and y are neighbors if they differ on exactly one bit
- **Hamming ball** of radius d centered at x contains all bitstrings y whose Hamming distance from x is at most d.

1-ball @ 111

2-ball @ 111

Code Distance

Block Codes, Geometrically. Recall a block code is a function from k-bit messages to n-bit encoded messages: $C : \{0,1\}^k \to \{0,1\}^n$

- C must be *injective*

Define $\mathscr{C} = C(\{0,1\}^k)$ to be the set of all codewords.

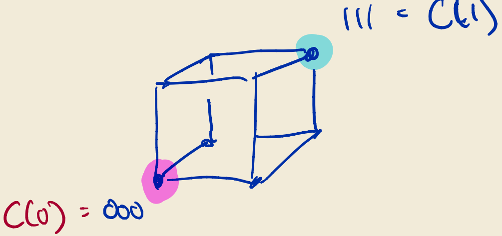
Code Distance

Block Codes, Geometrically. Recall a block code is a function from k-bit messages to n-bit encoded messages: $C : \{0,1\}^k \to \{0,1\}^n$

- C must be *injective*

Define $\mathscr{C} = C(\{0,1\}^k)$ to be the set of all codewords.

Example. $k = 1, n = 3$. Define $C(b) = (b, b, b)$.



Code Distance

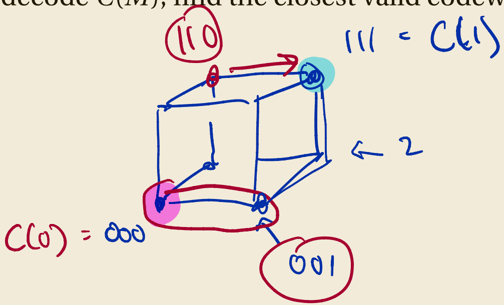
Block Codes, Geometrically. Recall a block code is a function from k-bit messages to n-bit encoded messages: $C : \{0,1\}^k \to \{0,1\}^n$

- C must be *injective*

Define $\mathscr{C} = C(\{0,1\}^k)$ to be the set of all codewords.

Example. $k = 1, n = 3$. Define $C(b) = (b, b, b)$.

Decoding. To decode $C(M)$, find the closest valid codeword x and take $S = C^{-1}(x)$.



Code Distance

Block Codes, Geometrically. Recall a block code is a function from k-bit messages to n-bit encoded messages: $C : \{0,1\}^k \to \boxed{\{0,1\}^n}$

- C must be *injective*

Define $\mathscr{C} = C(\{0,1\}^k)$ to be the set of all codewords.

Example. $k = 1, n = 3$. Define $C(b) = (b, b, b)$.

Decoding. To decode $C(M)$, find the closest valid codeword x and take $S = C^{-1}(x)$.

Definition. The **code distance** of C is the minimum (Hamming) distance between any two valid codewords:

- $d = \min_{x,y \in \mathscr{C}} d_H(x, y)$

Code Distance

Block Codes, Geometrically. Recall a block code is a function from k-bit messages to n-bit encoded messages: $C : \{0,1\}^k \to \{0,1\}^n$

- C must be *injective*

Define $\mathscr{C} = C(\{0,1\}^k)$ to be the set of all codewords.

Example. $k = 1, n = 3$. Define $C(b) = (b, b, b)$.

Decoding. To decode $C(M)$, find the closest valid codeword x and take $S = C^{-1}(x)$.

Definition. The **code distance** of C is the minimum (Hamming) distance between any two valid codewords:

- $d = \min_{x,y \in \mathscr{C}} d_H(x, y)$

Intuition. Larger code distances should be able to detect/correct more errors.

Lower Bounds

Requirements for Detecting and Correcting

Detecting Requirement. Suppose C can detect errors of flipping up to b bits. Then C has distance $\boxed{d \geq b+1}$.

n

b bits corrupted

Can detected error

Proof by contraposition.

Suppose $d \leq b$

$x \xleftrightarrow{b} y$

Consider

(1) A sends y, B receives y (no errors)

(2) A sends x, B receives y ($\leq b$ errors)

Bob cannot distinguish cases ⇒ error not detected! in (2).

Requirements for Detecting and Correcting

Detecting Requirement. Suppose C can detect errors of flipping up to b bits. Then C has distance $d \geq b+1$.

Correcting Requirement. Suppose C can correct errors of flipping up to b bits. Then C has distance $d \geq 2b+1$

Do for Thursday.

Lower Bounds for Block Codes

Question. For what values of n, k, d is it possible to have a block code of distance d?

Lower Bounds for Block Codes

Question. For what values of n, k, d is it possible to have a block code of distance d?

Singleton Bound. $2^k \leq 2^{n-(d-1)}$, hence $n \geq k + d - 1$

Lower Bounds for Block Codes

Question. For what values of n, k, d is it possible to have a block code of distance d?

Singleton Bound. $2^k \le 2^{n-(d-1)}$, hence $n \ge k + d - 1$
Proof sketch.

- Consider the deleting the first $d-1$ bits of each codeword.
- Remaining codewords are still pair-wise distinct
- There are only $2^{n-(d-1)}$ possible shorter bitstrings

Lower Bounds for Block Codes

Question. For what values of n, k, d is it possible to have a block code of distance d?

Singleton Bound. $2^k \le 2^{n-(d-1)}$, hence $n \ge k + d - 1$

Hamming bound. $2^k \le 2^n / \sum_{f=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{f}$.

Lower Bounds for Block Codes

Question. For what values of n, k, d is it possible to have a block code of distance d?

Singleton Bound. $2^k \le 2^{n-(d-1)}$, hence $n \ge k + d - 1$

Hamming bound. $2^k \le 2^n / \sum_{f=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{f}$.

Proof sketch.

- Codewords must be at distance d away
- Thus balls centered at codewords of radius $\lfloor (d-1)/2 \rfloor$ must be disjoint
- Number of balls $\times$ *volume* of each ball must be at most 2^n

Lower Bounds for Block Codes

Question. For what values of n, k, d is it possible to have a block code of distance d?

Singleton Bound. $2^k \le 2^{n-(d-1)}$, hence $n \ge k + d - 1$

Hamming bound. $2^k \le 2^n / \sum_{f=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{f}$.

Question. These are *im*possibility results. What is possible?

Error Detection: Parity Bits

Question. How can we **detect** a single error?

Error Detection: Parity Bits

Question. How can we **detect** a single error?
Obsevation. If a single bit gets flipped, the number of 1s increases or decreases by exactly 1

- the *parity* of the string changes

Error Detection: Parity Bits

Question. How can we **detect** a single error?

Obsevation. If a single bit gets flipped, the number of 1s increases or decreases by exactly 1

- the *parity* of the string changes

Idea. Form C by adding an extra bit to message m that encodes the parity of m

- the extra bit is called a **parity bit**
- which strings are valid codewords?

Error Detection: Parity Bits

Question. How can we **detect** a single error?

Obsevation. If a single bit gets flipped, the number of 1s increases or decreases by exactly 1

- the *parity* of the string changes

Idea. Form C by adding an extra bit to message m that encodes the parity of m

- the extra bit is called a **parity bit**
- which strings are valid codewords?
  - the parity of valid codewords is always 1!

Error Detection: Parity Bits

Question. How can we **detect** a single error?

Obsevation. If a single bit gets flipped, the number of 1s increases or decreases by exactly 1

- the *parity* of the string changes

Idea. Form C by adding an extra bit to message m that encodes the parity of m

- the extra bit is called a **parity bit**
- which strings are valid codewords?
  - the parity of valid codewords is always 1!

Example. $k = 2$, $n = 3$. What is d? How do we detect errors?

Parity Bit Example

Small Example. Consider $k = 2$, so that $n = 3$ with parity bits.

- Messages $\{00, 01, 10, 11\}$

Parity Bit Example

Small Example. Consider $k = 2$, so that $n = 3$ with parity bits.

- Messages $\{00, 01, 10, 11\}$
- $\mathscr{C} = \{000, 011, 101, 110\}$

Parity Bit Example

Small Example. Consider $k = 2$, so that $n = 3$ with parity bits.

- Messages $\{00, 01, 10, 11\}$
- $\mathscr{C} = \{000, 011, 101, 110\}$

PollEverywhere Question

Consider the code C with $k = 2$ bit messages and one parity bit. What is the distance d of C?

pollev.com/comp526

Parity Bit Example

Small Example. Consider $k = 2$, so that $n = 3$ with parity bits.

- Messages $\{00, 01, 10, 11\}$
- $\mathscr{C} = \{000, 011, 101, 110\}$
- What is the distance of C?

Parity Bit Example

Small Example. Consider $k = 2$, so that $n = 3$ with parity bits.

- Messages $\{00, 01, 10, 11\}$
- $\mathscr{C} = \{000, 011, 101, 110\}$
- What is the distance of C?
- How do we detect errors?

Error Correction through Duplication

Suppose we want to **correct** a single error. How is this even possible?

Error Correction through Duplication

Suppose we want to **correct** a single error. How is this even possible?

Simple Solution. Duplicate each bit 3 times and send the duplicates!

- $k = 1$, $n = 3$
- $C(b) = bbb$
- How do we decode a message?

Error Correction through Duplication

Suppose we want to **correct** a single error. How is this even possible?
Simple Solution. Duplicate each bit 3 times and send the duplicates!

- $k = 1$, $n = 3$
- $C(b) = bbb$
- How do we decode a message?
- View on Hamming cube!

Error Correction through Duplication

Suppose we want to **correct** a single error. How is this even possible?
Simple Solution. Duplicate each bit 3 times and send the duplicates!

- $k = 1$, $n = 3$
- $C(b) = bbb$
- How do we decode a message?

Inefficiency. To correct a single error, we must **triple** the length of the message?!

Error Correction through Duplication

Suppose we want to **correct** a single error. How is this even possible?

Simple Solution. Duplicate each bit 3 times and send the duplicates!

- $k = 1$, $n = 3$
- $C(b) = bbb$
- How do we decode a message?

Inefficiency. To correct a single error, we must **triple** the length of the message?!

A Puzzle. How can we correct a single error more efficiently?

- Don't need to duplicate every bit!
- Idea: use parity checks on *parts* of the string to identify the index where error occurred!

Next Time

- Finish error correcting codes!
- Start parallel algorithms!

Scratch Notes