Error Checking Codes

Dr. Carmen Bruni

University of Waterloo

December 11th, 2015

In the beginning...

We had very primitive methods of long distance communication...

Smoke Signals

String Phone

Telephone

Wired Internet

Wireless Internet

Transmitting Messages

Humanity has come a long way when it comes to transmitting messages but all of these methods of communication suffer from the same flaw:

Transmitting Messages

Humanity has come a long way when it comes to transmitting messages but all of these methods of communication suffer from the same flaw:

Transmitting messages is not an error free process!

Transmitting Messages

Humanity has come a long way when it comes to transmitting messages but all of these methods of communication suffer from the same flaw:

Transmitting messages is not an error free process!

For example: Telephone Game.

Transmitting Messages

Humanity has come a long way when it comes to transmitting messages but all of these methods of communication suffer from the same flaw:

Transmitting messages is not an error free process!

For example: Telephone Game.

What has changed over time is what is causing the error but nonetheless errors occur in communications.

A Problem

- Suppose you wanted to send a message "Call me at 519-888-4567".
- If the recipient receives the message "**R**all me **t**t 519-888-4567", the recipient might be annoyed but could probably determine the message.
- However, if the recipient receives the message "Call me at 519-888-45**2**7", then we have a big issue on our hands and all we did was change one number.

Issue to Solve

We want to send messages without any mistakes using error prone devices that can be influenced by very tiny external factors.

Issue to Solve

We want to send messages without any mistakes using error prone devices that can be influenced by very tiny external factors.

What are some sources of errors?

Sources of Error

Samples of sources of errors:

- CDs and DVDs can have scratches
- Wifi has multiple competing and interferring signals.
- Data on a hard drive is largely influenced by magnetic interruptions.
- Optical fibre wires suffer from noise, cable breaks, etc.

Attempt 1: Repetition

If there's worry about receiving the message properly, why not just send it multiple times and just take the majority vote for each character of the message?

Attempt 1: Repetition

If there's worry about receiving the message properly, why not just send it multiple times and just take the majority vote for each character of the message?

Message: "Transfer $3592.15 into account number 123456789"

Transferred Messages:

- "Transfer $3**3**92.15 into account number 123456789"
- "Tra**m**sfer $359**3**.15 into account number 1234**3**6789"
- "Tr**G**nsfer $3592.15 into acc**l**unt number 12**9**4567**4**9"
- "Transfer $359**8**.15 into account number 123456**8**89"
- "Transfer $3592.15 into account number 123456789"

Here the repeated trick worked and the message received would be the most common message in each character position, which in this case is the original message.

Attempt 1: Repetition Gone Awry

However we might not get so lucky...

Attempt 1: Repetition Gone Awry

However we might not get so lucky...

Message: "Transfer $3592.15 into account number 123456789"

Transferred Messages:

- "Transfer $3**3**92.15 into account number 1234**3**6789"
- "Tra**m**sfer $359**3**.15 into account number 1234**3**6789"
- "Tr**G**nsfer $3592.15 into acc**l**unt number 12**9**4567**4**9"
- "Transfer $359**8**.15 into account number 1234**3**6**8**89"
- "Transfer $3592.15 into account number 123456789"

Attempt 1: Repetition Gone Awry

However we might not get so lucky...

Message: "Transfer $3592.15 into account number 123456789"

Transferred Messages:

- "Transfer $3**3**92.15 into account number 1234**3**6789"
- "Tra**m**sfer $359**3**.15 into account number 1234**3**6789"
- "Tr**G**nsfer $3592.15 into acc**l**unt number 12**9**4567**4**9"
- "Transfer $359**8**.15 into account number 1234**3**6**8**89"
- "Transfer $3592.15 into account number 123456789"

Here the repeated trick method fails and the message received would be"Transfer $3592.15 into account number 1234**3**6789" even though we did mange to get the message sent error free one time above! This might not even be that contrived of a situation if the data is being send across some wire with damage in a specific spot always causing a certain bit to change.

Pros and Cons

What are some pros and cons to this method?

Attempt 1: Repetition Summary

Pros:

Attempt 1: Repetition Summary

Pros:

- Provides at least some improved chance of working.
- Detects and corrects errors often.

Attempt 1: Repetition Summary

Pros:

- Provides at least some improved chance of working.
- Detects and corrects errors often.

Cons:

- How many times do we send the message? Lots of extra overhead each time we send.
- Don't know when we actually receive the correct message.
- If there is a specific error that always causes a specific bit to differ, this method will never work.

This is actually used in practice! (usually for storage but not for transmission).

Attempt 2: Redundancy

A possibly source of the previous problem is that looking at the individual numbers themselves have far too much importance. Why don't we lengthen each digit and send the important data redundantly?

One way to do this is to use words to describe the numbers.

Attempt 2: Redundancy Redundancy

Message to send: “123456789”

Message we send: “one two three four five six seven eight nine”

Message received: “pne tto thrxp gour eive sik teven night pwne”

Attempt 2: Redundancy Redundancy

Message to send: “123456789”

Message we send: “one two three four five six seven eight nine”

Message received: “pne tto thrxp gour eive sik teven night pwne”

Even with 11 errors out of 36 non blank characters, we could still read the message.

Attempt 2: Redundancy Redundancy

Message to send: "123456789"

Message we send: "one two three four five six seven eight nine"

Message received: "pne tto thrxp gour eive sik teven night pwne"

Even with 11 errors out of 36 non blank characters, we could still read the message.

In practice this isn't the exact method used but gives the general idea for Hamming Codes (we'll discuss these later!).

Humans and computers can decipher this

For emaxlpe, it deson?t mttaer in waht oredr the ltteers in a wrod aepapr, the olny iprmoatnt tihng is taht the frist and lsat ltteer are in the rghit pcale. The rset can be a toatl mses and you can sitll raed it wouthit pobelrm.

S1M1L4RLY, YOUR M1ND 15 R34D1NG 7H15 4U70M471C4LLY W17HOU7 3V3N 7H1NK1NG 4BOU7 17.

http://www.livescience.com/18392-reading-jumbled-words.html

Pros and Cons

What are some pros and cons to this method?

Attempt 2: Redundancy Summary

Pros:

Attempt 2: Redundancy Summary

Pros:

- Provides a good chance of working.
- Detects and corrects errors often.
- Have an idea when the correct message is sent.

Attempt 2: Redundancy Summary

Pros:

- Provides a good chance of working.
- Detects and corrects errors often.
- Have an idea when the correct message is sent.

Cons:

- How should we encode the data? What redundancy should we used that's optimal?
- If there is a specific error that always causes a specific bit to differ, this method will never work.

These techniques when modified are used in practice. (We'll see examples later)

Attempt 3: Checksum

- What if instead of sending a redundancy of each digit, there was a way to encode a certain redundancy in the entire message?

Attempt 3: Checksum

- What if instead of sending a redundancy of each digit, there was a way to encode a certain redundancy in the entire message?
- Let's use an example in binary (using only zeroes and ones). Suppose we wanted to send the message 01100110110.

Attempt 3: Checksum

- What if instead of sending a redundancy of each digit, there was a way to encode a certain redundancy in the entire message?
- Let's use an example in binary (using only zeroes and ones). Suppose we wanted to send the message 01100110110.
- This message has 11 digits and 6 of them are ones.

Attempt 3: Checksum

- What if instead of sending a redundancy of each digit, there was a way to encode a certain redundancy in the entire message?
- Let's use an example in binary (using only zeroes and ones). Suppose we wanted to send the message 01100110110.
- This message has 11 digits and 6 of them are ones.
- Suppose we changed exactly one digit. Then the number of ones would have changed to either 5 or 7 depending on if we changed a 1 to a 0 or a 0 to a 1.

Attempt 3: Checksum

- With the message 01100110110, we send a final checksum digit that is a parity bit:

  1 if the number of ones is odd and 0 if the number of ones is even.

Attempt 3: Checksum

- With the message 01100110110, we send a final checksum digit that is a parity bit:

  1 if the number of ones is odd and 0 if the number of ones is even.

- Here, since we have 6 ones, we will add an extra 0 to the end of the checksum. Thus, we send the message

  011001101100

Attempt 3: Checksum

- In this way, a number can be detected to be incorrect if exactly one digit is incorrect.

Attempt 3: Checksum

- In this way, a number can be detected to be incorrect if exactly one digit is incorrect.
- With the message plus checksum digit from the last example 011001101100, if we received say

$$01\mathbf{0}001101100$$

then we would know the message is incorrect since the check sum digit says there should be an even number of ones but there is an odd number of ones and this is an error.

Attempt 3: Checksum

- In this way, a number can be detected to be incorrect if exactly one digit is incorrect.
- With the message plus checksum digit from the last example 011001101100, if we received say

  $$01\mathbf{0}001101100$$

  then we would know the message is incorrect since the check sum digit says there should be an even number of ones but there is an odd number of ones and this is an error.
- Even though we know there is an error, our scheme cannot determine where the error is. It would just reject the message.

Attempt 3: Checksum

- In this way, a number can be detected to be incorrect if exactly one digit is incorrect.
- With the message plus checksum digit from the last example 011001101100, if we received say

$$01\mathbf{0}001101100$$

  then we would know the message is incorrect since the check sum digit says there should be an even number of ones but there is an odd number of ones and this is an error.
- Even though we know there is an error, our scheme cannot determine where the error is. It would just reject the message.
- We call such a message potentially valid if it passes our checksum test.

Attempt 3: Checksum

Question: Which of the following are potentially valid messages?

1. 01010101010
2. 11101110101
3. 000000000000
4. 1110110101001

Attempt 3: Checksum

Question: Which of the following are potentially valid messages?

1. 01010101010

Attempt 3: Checksum

Question: Which of the following are potentially valid messages?

1. 01010101010

   Invalid. There are 5 ones but the checksum digit is 0.

Attempt 3: Checksum

Question: Which of the following are potentially valid messages?

1. 01010101010

   Invalid. There are 5 ones but the checksum digit is 0.

2. 11101110101

Attempt 3: Checksum

Question: Which of the following are potentially valid messages?

1. 01010101010

   Invalid. There are 5 ones but the checksum digit is 0.

2. 11101110101

   Potentially valid. There are 7 ones excluding the checksum digit which is a 1.

Attempt 3: Checksum

Question: Which of the following are potentially valid messages?

1. 01010101010

   Invalid. There are 5 ones but the checksum digit is 0.

2. 11101110101

   Potentially valid. There are 7 ones excluding the checksum digit which is a 1.

3. 000000000000

Attempt 3: Checksum

Question: Which of the following are potentially valid messages?

1. 01010101010

   Invalid. There are 5 ones but the checksum digit is 0.

2. 11101110101

   Potentially valid. There are 7 ones excluding the checksum digit which is a 1.

3. 000000000000

   Potentially valid. There are no ones and the checksum digit is 0.

Attempt 3: Checksum

Question: Which of the following are potentially valid messages?

1. 01010101010

   Invalid. There are 5 ones but the checksum digit is 0.

2. 11101110101

   Potentially valid. There are 7 ones excluding the checksum digit which is a 1.

3. 000000000000

   Potentially valid. There are no ones and the checksum digit is 0.

4. 10110101001

Attempt 3: Checksum

Question: Which of the following are potentially valid messages?

1. 01010101010

   Invalid. There are 5 ones but the checksum digit is 0.

2. 11101110101

   Potentially valid. There are 7 ones excluding the checksum digit which is a 1.

3. 000000000000

   Potentially valid. There are no ones and the checksum digit is 0.

4. 10110101001

   Potentially valid. There are 5 ones excluding the checksum digit which is a 1.

Pros and Cons

What are some pros and cons to this method?

Attempt 3: Checksum

Pros:

Attempt 3: Checksum

Pros:

- Check sums can be used to detect a single error.
- Provides some verification that the received message is correct.

Attempt 3: Checksum

Pros:

- Check sums can be used to detect a single error.
- Provides some verification that the received message is correct.

Cons:

- Often fails to detect multiple errors.
- Cannot be used to correct an error.

Attempt 3: Checksum

Pros:

- Check sums can be used to detect a single error.
- Provides some verification that the received message is correct.

Cons:

- Often fails to detect multiple errors.
- Cannot be used to correct an error.

However, despite it's primacy, this type of idea is widely used. In what follows we give two examples of this idea being used.

Idea 1: Luhn's Algorithm

- In our everyday life, we tend to deal more with decimal digits than with binary digits.

Idea 1: Luhn's Algorithm

- In our everyday life, we tend to deal more with decimal digits than with binary digits.
- Is there an algorithm that works for decimal numbers and is used?

Idea 1: Luhn's Algorithm

- In our everyday life, we tend to deal more with decimal digits than with binary digits.
- Is there an algorithm that works for decimal numbers and is used?
- Yes! An algorithm due to Luhn is widely used and many people in this room have something that has this algorithm embedded in it.

My Credit Card Number

- Take for example my credit card number:

  4025901376813517

My Credit Card Number

- Take for example my credit card number:

  4025901376813517

- This card number has a check sum digit in place (the last digit). In a credit card number, 15 digits are chosen and the 16th digit is picked to satisfy the following algorithm.

My Credit Card Number

- Take for example my credit card number:

$$4025901376813517$$

- This card number has a check sum digit in place (the last digit). In a credit card number, 15 digits are chosen and the 16th digit is picked to satisfy the following algorithm.
- First, for all the digits in even positions, add them up. Call this number E.

$$E = 0 + 5 + 0 + 3 + 6 + 1 + 5 + 7 = 27$$

My Credit Card Number

Credit Card Number: 4025901376813517 $E = 27$

My Credit Card Number

Credit Card Number: 4025901376813517 $E = 27$

- Then, for all the digits in odd positions, double the digits then add all the numbers. Call this number O.

My Credit Card Number

Credit Card Number: 4025901376813517 $E = 27$

- Then, for all the digits in odd positions, double the digits then add all the numbers. Call this number O.
- The odd positioned digits are $4, 2, 9, 1, 7, 8, 3, 1$.

My Credit Card Number

Credit Card Number: 4025901376813517 $E = 27$

- Then, for all the digits in odd positions, double the digits then add all the numbers. Call this number O.
- The odd positioned digits are $4, 2, 9, 1, 7, 8, 3, 1$.
- Doubling gives $8, 4, 18, 2, 14, 16, 6, 2$.

My Credit Card Number

Credit Card Number: 4025901376813517 $E = 27$

- Then, for all the digits in odd positions, double the digits then add all the numbers. Call this number O.
- The odd positioned digits are $4, 2, 9, 1, 7, 8, 3, 1$.
- Doubling gives $8, 4, 18, 2, 14, 16, 6, 2$.
- Adding the digits together:

$$O = 8 + 4 + 1 + 8 + 2 + 1 + 4 + 1 + 6 + 6 + 2 = 43$$

My Credit Card Number

Credit Card Number: 4025901376813517 $E = 27$

- Then, for all the digits in odd positions, double the digits then add all the numbers. Call this number O.
- The odd positioned digits are $4, 2, 9, 1, 7, 8, 3, 1$.
- Doubling gives $8, 4, 18, 2, 14, 16, 6, 2$.
- Adding the digits together:

$$O = 8 + 4 + 1 + 8 + 2 + 1 + 4 + 1 + 6 + 6 + 2 = 43$$

- Lastly, add E and O. If the sum is divisible by 10, we call the number potentially valid.

My Credit Card Number

Credit Card Number: 4025901376813517 $E = 27$

- Then, for all the digits in odd positions, double the digits then add all the numbers. Call this number O.
- The odd positioned digits are $4, 2, 9, 1, 7, 8, 3, 1$.
- Doubling gives $8, 4, 18, 2, 14, 16, 6, 2$.
- Adding the digits together:

$$O = 8 + 4 + 1 + 8 + 2 + 1 + 4 + 1 + 6 + 6 + 2 = 43$$

- Lastly, add E and O. If the sum is divisible by 10, we call the number potentially valid.
- Notice that

$$E + O = 70$$

and hence my credit card number is potentially valid.

Which of these are potentially valid?

1. 9054328746541532
2. 6154328746541537
3. 4142658749512432
4. 9974865142354173

Solutions

9054328746541532

Solutions

9054328746541532

Even Positions

- $0, 4, 2, 7, 6, 4, 5, 2$
- Sum is

$$0+4+2+7+6+4+5+2 = 30.$$

Odd Positions

- $9, 5, 3, 8, 4, 5, 1, 3$
- Doubling:
  $18, 10, 6, 16, 8, 10, 2, 6$
- Sum of digits is

$$1+8+1+0+6+1+6 \\ +8+1+0+2+6 = 40.$$

Sum is $30 + 40 = 70$ which is divisible by 10.

The number is potentially valid.

Solutions

6154328746541537

Solutions

6154328746541537

Even Positions

- $1, 4, 2, 7, 6, 4, 5, 7$
- Sum is

$$1+4+2+7+6+4+5+7 = 36.$$

Odd Positions

- $6, 5, 3, 8, 4, 5, 1, 3$
- Doubling:
  $12, 10, 6, 16, 8, 10, 2, 6$
- Sum of digits is

$$1+2+1+0+6+1+6 \\ +8+1+0+2+6 = 34.$$

Sum is $36 + 34 = 70$ which is divisible by 10.

The number is potentially valid.

Solutions

4142658749512432

Solutions

4142658749512432

Even Positions

- $1, 2, 5, 7, 9, 1, 4, 2$
- Sum is

$$1+2+5+7+9+1+4+2 = 31.$$

Odd Positions

- $4, 4, 6, 8, 4, 5, 2, 3$
- Doubling:
  $8, 8, 12, 16, 8, 10, 4, 6$
- Sum of digits is

$$8+8+1+2+1+6+8 \\ +1+0+4+6 = 45.$$

Sum is $31 + 45 = 76$ which is not divisible by 10.

The number is not potentially valid (invalid).

Solutions

9974865142354173

Even Positions

- $9, 4, 6, 1, 2, 5, 1, 3$
- Sum is

$$9+4+6+1+2+5+1+3 = 31.$$

Odd Positions

- $9, 7, 8, 5, 4, 3, 4, 7$
- Doubling:
  $18, 14, 16, 10, 8, 6, 8, 14$
- Sum of digits is

$$\begin{aligned} &1+8+1+4+1+6 \\ &+1+0+0+8+6 \\ &+8+1+4 = 49. \end{aligned}$$

Sum is $31 + 49 = 80$ which is divisible by 10.

The number is potentially valid.

Why is Luhn's Algorithm useful?

- Luhn's Algorithm can detect the two most common data entry errors for numbers:

Why is Luhn's Algorithm useful?

- Luhn's Algorithm can detect the two most common data entry errors for numbers:
  1. Getting a single digit incorrect.

Why is Luhn's Algorithm useful?

- Luhn's Algorithm can detect the two most common data entry errors for numbers:
  1. Getting a single digit incorrect.
  2. Switching adjacent digits (with the exception of switching a 9 and 0).

Error Detection in Luhn's Algorithm (single digit).

Actual: 9054328746541532 Received: 9054338746541532

Error Detection in Luhn's Algorithm (single digit).

Actual: 9054328746541532 Received: 9054338746541532

Even Positions

- $0, 4, 3, 7, 6, 4, 5, 2$
- Sum is

$$0+4+3+7+6+4+5+2 = 31.$$

Odd Positions

- $9, 5, 3, 8, 4, 5, 1, 3$
- Doubling:
  $18, 10, 6, 16, 8, 10, 2, 6$
- Sum of digits is

$$1+8+1+0+6+1+6 \\ +8+1+0+2+6 = 40.$$

Sum is $31 + 40 = 71$ which is not divisible by 10.

The number is not potentially valid.

Error Detection in Luhn's Algorithm (single digit).

Actual: 9054328746541532 Received: 9054327846541532

Error Detection in Luhn's Algorithm (single digit).

Actual: 9054328746541532 Received: 9054327846541532

Even Positions

- $0, 4, 2, 8, 6, 4, 5, 2$
- Sum is

$$0+4+2+8+6+4+5+2 = 31.$$

Odd Positions

- $9, 5, 3, 7, 4, 5, 1, 3$
- Doubling:
  $18, 10, 6, 14, 8, 10, 2, 6$
- Sum of digits is

$$1+8+1+0+6+1+4$$
$$+8+1+0+2+6 = 38.$$

Sum is $31 + 38 = 69$ which is not divisible by 10.

The number is not potentially valid.

Error Detection in Luhn's Algorithm (Does not detect the 90 swap).

Actual: 9054328746541532 Received: 0954328746541532

Error Detection in Luhn's Algorithm (Does not detect the 90 swap).

Actual: 9054328746541532 Received: 0954328746541532

Even Positions

- $9, 4, 2, 7, 6, 4, 5, 2$
- Sum is

$$9+4+2+7+6+4+5+2 = 39.$$

Odd Positions

- $0, 5, 3, 8, 4, 5, 1, 3$
- Doubling:
  $0, 10, 6, 16, 8, 10, 2, 6$
- Sum of digits is

$$0 + 1 + 0 + 6 + 1 + 6 + 8 + 1 + 0 + 2 + 6 = 31.$$

Sum is $39 + 31 = 70$ which is divisible by 10.

The number is potentially valid.

Luhn's Algorithm Sanity Check

Which of these numbers are potentially valid given that your credit card number is:

1234567899000123

1. 1234657899000123
2. 1234597899000123
3. 1234567890900123
4. 3214567899000123
5. 2134567899000213

Luhn's Algorithm Sanity Check

Which of these numbers are potentially valid given that your credit card number is:

1234567899000123

1. 1234657899000123 not potentially valid
2. 1234597899000123 not potentially valid
3. 1234567890900123 potentially valid
4. 3214567899000123 potentially valid
5. 2134567899000213 ???

Error Detection in Luhn's Algorithm (Does not detect the 90 swap).

2134567899000213

Even Positions

- $1, 4, 6, 8, 9, 0, 2, 3$
- Sum is

$$1+4+6+8+9+0+2+3 = 33.$$

Odd Positions

- $2, 3, 5, 7, 9, 0, 0, 1$
- Doubling:
  $4, 6, 10, 14, 18, 0, 0, 2$
- Sum of digits is

$$4 + 6 + 1 + 0 + 1 + 4 + 1 + 8 + 0 + 0 + 2 = 27.$$

Sum is $33 + 27 = 60$ which is divisible by 10.

The number is potentially valid.

Pros and Cons

What are some pros and cons to this method?

Summary of Luhn's Algorithm

Pros:

- Detects most common single errors.

Summary of Luhn's Algorithm

Pros:

- Detects most common single errors.

Cons:

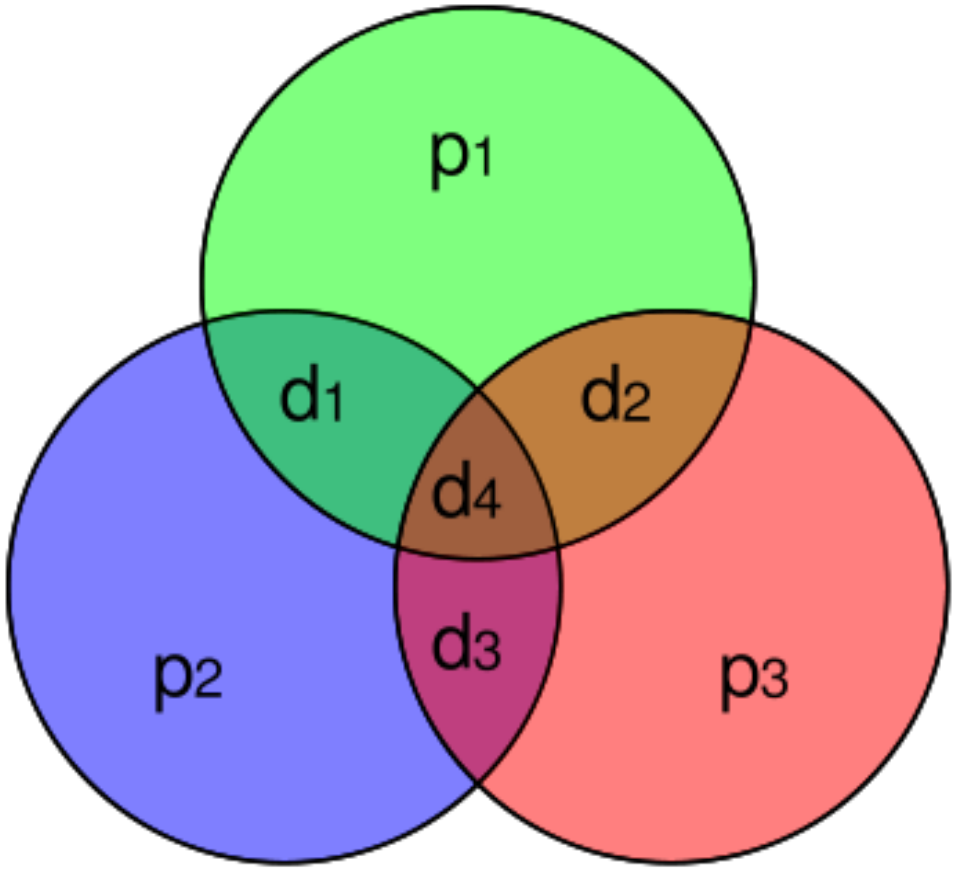
- Cannot detect multiple errors.
- Cannot correct errors, only detect.

Idea 2: $(7, 4)$ Hamming Code

The $(7, 4)$ Hamming Code is a redundancy error checking code that takes in four bits and produces a 7 bit encoding consisting of the original four bits and three parity bits $d_1 d_2 d_3 d_4 p_1 p_2 p_3$.

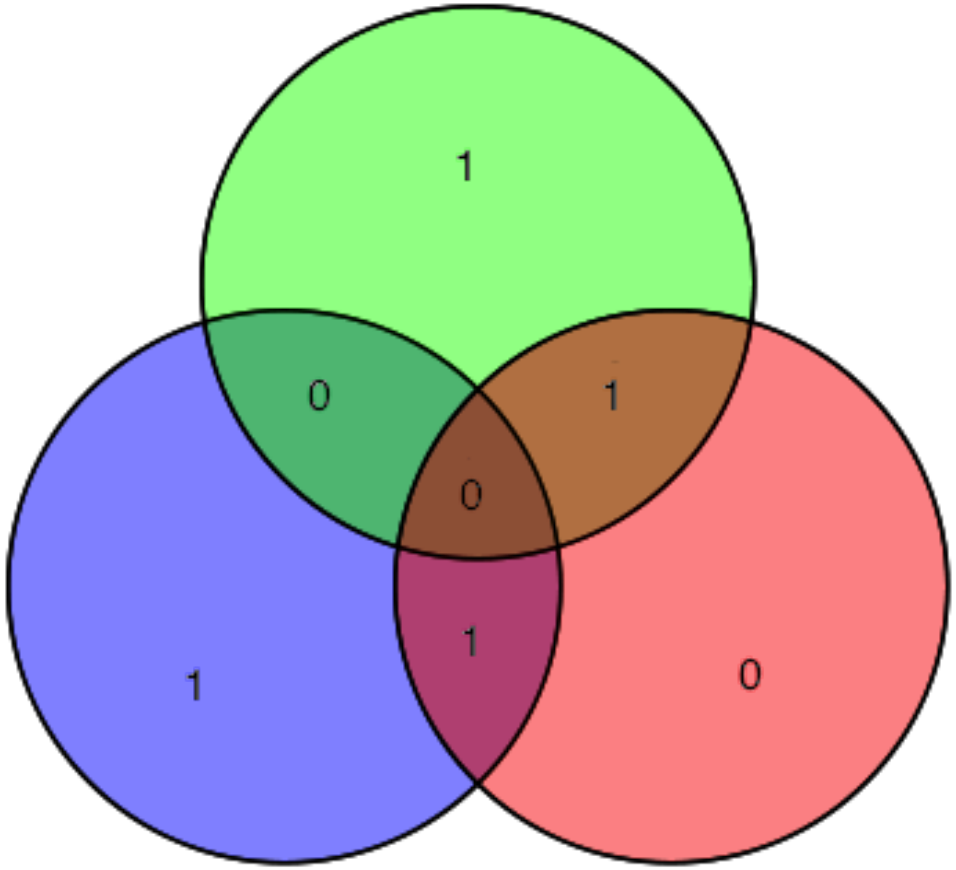
Idea 2: $(7, 4)$ Hamming Code

The $(7, 4)$ Hamming Code is a redundancy error checking code that takes in four bits and produces a 7 bit encoding consisting of the original four bits and three parity bits $d_1 d_2 d_3 d_4 p_1 p_2 p_3$.



Note: Ordering of digits differs in practice.

$(7, 4)$ Hamming Code (Courtesy Wikipedia)



Your Turn! $(7, 4)$ Hamming Code

Complete the table below

Original	Encoding	Original	Encoding
0000	0000000	1000	1000110
0001	0000111	1001	1001001
0010	0000011	1010	1010101
0011	0000100	1011	1011010
0100	0100101	1100	
0101	0101010	1101	
0110	0110110	1110	
0111	0111001	1111	

$(7, 4)$ Hamming Code

Original	Encoding	Original	Encoding
0000	0000000	1000	1000110
0001	0000111	1001	1001001
0010	0000011	1010	1010101
0011	0000100	1011	1011010
0100	0100101	1100	1100011
0101	0101010	1101	1101100
0110	0110110	1110	1110000
0111	0111001	1111	1111111

$(7, 4)$ Hamming Code Error Correction

With this Hamming Code, we can actually detect and correct single errors!

$(7, 4)$ Hamming Code Error Correction

With this Hamming Code, we can actually detect and correct single errors!

- Suppose we receive the message 0110101.

$(7, 4)$ Hamming Code Error Correction

With this Hamming Code, we can actually detect and correct single errors!

- Suppose we receive the message 0110101.
- Out parity bits tells us there is a problem. In digits 1, 3 and 4, there is a single one and so the second parity bit should be a 1. However it is a 0.

(7, 4) Hamming Code Error Correction

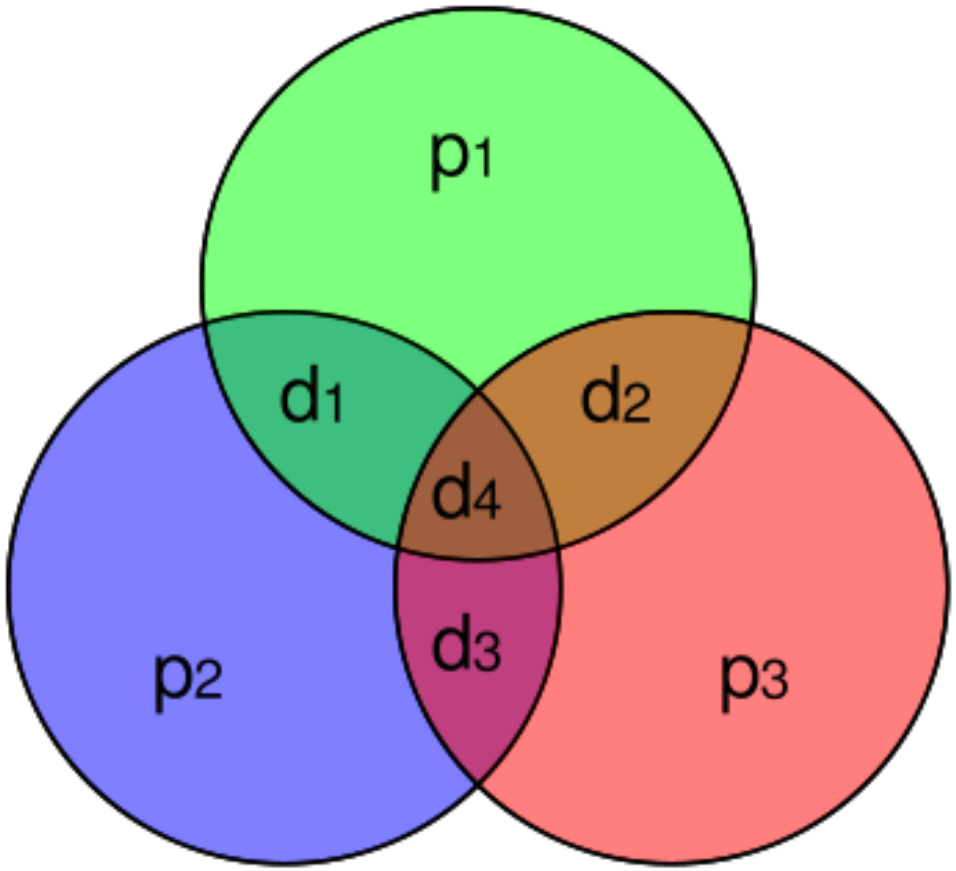
With this Hamming Code, we can actually detect and correct single errors!

- Suppose we receive the message 0110101.
- Out parity bits tells us there is a problem. In digits 1, 3 and 4, there is a single one and so the second parity bit should be a 1. However it is a 0.
- Continuing our check:
  - Digits 1,2,4 contain a single 1 so the first parity bit should be a 1.
  - Digits 2,3,4 contain 2 copies of 1 so the last parity bit should be a 0 but it is a 1.
- So the two check sums containing the digit 3 are incorrect. This means that digit 3 is incorrect and changing the 1 back to a 0 recovers the message.
- This assumes that there was only one message in the code!

(7, 4) Hamming Code Error Correction

The following have a single error. Find and correct the error

1. 0010110
2. 1000001
3. 1011011



$(7,4)$ Hamming Code Error Correction

The following have a single error. Find and correct the error

1. 0110110
2. 1001001
3. 1011010

Pros and Cons

What are some pros and cons to this method?

$(7, 4)$ Hamming Code

Pros:

- Detects and corrects single errors.

$(7, 4)$ Hamming Code

Pros:

- Detects and corrects single errors.

Cons:

- Might erroneously correct errors if multiple errors are made.
- Can detect errors up to 3 bits incorrect (but lose ability to correct single errors).

This is actually used in practice! For example DRAM still uses this. This was originally created by Richard Hamming while working for Bell Telephone Laboratories. He was frustrated by the error-prone punch card readers and wanted to create a system to double check the code.

Summary

- Error checking codes are vital to many current systems.
- Many complicated codes exist to help solve this problem. (Reed-Solomon Codes, QR Decompositions etc.)
- Error checking codes exist in many current objects like CDs, credit cards and SIN numbers.

Thank you!

Thank you!

Carmen Bruni

cbruni@uwaterloo.ca