Simulation Results of Channel Capacity using Parity Bits for Error Correction

Austin Minor

November 24, 2016

1   Introduction

Claude Shannon was a major figure in creating the mathematical discipline of information theory. This paper analyzes how empirical results line up with the statements made by this mathematical model of information. More formally, we will be analyzing the effect of using parity bits for error correction and how well they are at approaching the formal limit on the conveyance of data.

We will start with a mathematical description of the concepts related to the problem. From there, we will give the problem statement. After running the test, we will analyze the results and give our conclusion.

2   Mathematical Description

The official description of conveyable information is the mutual information between to system of events. Let A be the source alphabet. Let B be the reception alphabet. Let Q be the transmission matrix (IE the probability of receiving $b_j \in B$ given a transmission of $a_i \in A$. Let these entries be represented as $q_{i,j}$ respectively. For our example we will only consider the simpler channel known as the binary symmetric channel. Thus

$$Q = \begin{pmatrix} q_{0,0} & q_{0,1} \\ q_{0,1} & q_{1,1} \end{pmatrix} = \begin{pmatrix} p & 1-p \\ 1-p & p \end{pmatrix}$$

Given such a channel, the probability of bit errors can be represented by a Bernoulli Trial.

The mutual information between two system of events (definition given elsewhere) is defined as following:

$$I(A, B) = \sum_{i=1}^{m} \sum_{j=1}^{n} Pr(A_i \cup B_j) \log \left(\frac{Pr(A_i \cup B_j)}{Pr(A_i)Pr(B_j)} \right)$$

$$... = \sum_{i=1}^{m} Pr(A_i) \sum_{j=1}^{n} q_{i,j} \log \left(\frac{q_{i,j}}{\sum_{t=1}^{m} Pr(A_t)q_{t,j}} \right)$$

It has been proved elsewhere that the maximal channel capacity (maximum I(A,B) based on $Pr(A_i)$ – probability that a source letter is transmitted) for a BSC is when $Pr(A_1 = 0) = Pr(A_2 = 1) = \frac{1}{2}$. Furthermore for these values,

$$I(A, B) = p * log(2 * p) + (1 - p) * log(2 * (1 - p))$$

For example, if $p = .99$ then $I(A, B) = .919$.

3   Problem Statement

To test this model of channel capacity and mutual information, a Matlab program was written that would satisfy the important requirements listed above $(Pr(A_i) = \frac{1}{2}$, Q as described above). The necessary error correction was implemented using parity bits. The Matlab code is attached below.

```matlab
1   PRECISION = 100;
2   RETRY_COUNT = 10000;
3
4   %first avg transmission
5   %second error rate of transmission
6   %third avg channel capacity
7   A = zeros(3, 100);
8   A = zeros(2, PRECISION);
9
10  temp = 0;
11  for i = 1:PRECISION
12    for j = 1:RETRY_COUNT
13      temp = transmit(generate_random_msg(10,4), i/PRECISION, 20);
14      if temp == -1
15        A(2,i) = A(2,i) + 1;
16      else
17        A(1,i) = A(1,i) + temp;
18      end
19    end
20    %compute avg ignoring failed transmissions
21    temp = RETRY_COUNT - A(2,i);
22    if temp == 0
23    %do nothing because A(1,i) will be zero also
24    else
25      A(1,i) = A(1,i)/(RETRY_COUNT-A(2,i));
26    end
27  end
28
29  %compute channel cap graphics_toolkit
30  %min bytes computed by testing for zero error
31  min_bytes = transmit(generate_random_msg(10,4), 0, 20);
32  %remember to subtract parity bytes because not in orginal msg
33  A(3,:) = (ones(1,PRECISION)*(min_bytes-10))./A(1,:);
34
35  figure
36  %bits of info needed for transmission
37  a = subplot(3,1,1);
38  plot(A(1,:))
39  ylabel(a,'Number of Bits Needed for Transmission')
40  xlabel(a,'Error Chance (1-Success Percentage)')
41  %number of errors in transmission
42  b = subplot(3,1,2);
43  plot(A(2,:))
44  ylabel(b,'Number of Errant Messages')
45  xlabel(b,'Error Chance (1-Success Percentage)')
46  %channel capacity
47  c = subplot(3,1,3);
48  plot(A(3,:))
49  ylabel(c,'Channel Capacity')      3
50  xlabel(c,'Error Chance (1-Success Percentage)')
```

Listing 1: Main Simulator

```matlab
function bit_count = transmit(msg, err_prob, retry_lim)
  msg = insert_parity_bit(msg);

  num_transmissions = 0;
  transmitted_msg = 0;
  for i = 1:size(msg)(1)
    transmitted_msg = transmit_msg(msg(i,:),err_prob);
    %check rough equality with parity bit
    %guarentees proper transmission for one or less errors
    temp_c = 0;
    while transmitted_msg(1) != parity(transmitted_msg(2:end)) && temp_c < retry_lim
      num_transmissions = num_transmissions + 1;
      temp_c = temp_c + 1;
      transmitted_msg = transmit_msg(msg(i,:),err_prob);
    end
    %msg would fail using given parity scheme and error probality
    if isequal(transmitted_msg, msg(i,:)) == false
      bit_count = -1;
      return
    end
    %always one transmission
    num_transmissions = num_transmissions + 1;
  end

  %num trans multiplied by the size of packet transmitted
  bit_count = num_transmissions * size(msg)(2);
end
```

Listing 2: Transmission Simulator

```matlab
function msg = transmit_msg(msg_in, err_prob)
  msg = zeros(size(msg_in));
  rnd_num = 0;
  for i = 1:size(msg_in)(2)
    rnd_num = rand();

    if rnd_num < err_prob
      if msg_in(i) == 0
        msg(i) = 1;
      else
        msg(i) = 0;
      end
    else
      msg(i) = msg_in(i);
    end
  end
end
```

Listing 3: Per-packet Transmission Simulator

```matlab
function msg = insert_parity_bit(msg_in)
  c = size(msg_in)(1);
  msg = zeros(size(msg_in)(1),size(msg_in)(2) + 1);
  for i = 1:c
    %https://www.mathworks.com/matlabcentral/newsreader/view_thread/331396
    msg(i,:) = [parity(msg_in(i,:)) msg_in(i,:)];
  end
end
```

Listing 4: Per-packet Parity Bit Inserter

```
1  function par = parity(packet)
2    num_ones = 0;
3    for i = 1:size(packet)(2)
4      if packet(i) == 1
5        num_ones = num_ones + 1;
6      end
7    end
8    if (mod(num_ones,2) == 0)
9      par = 0;
10   else
11     par = 1;
12   end
13 end
```

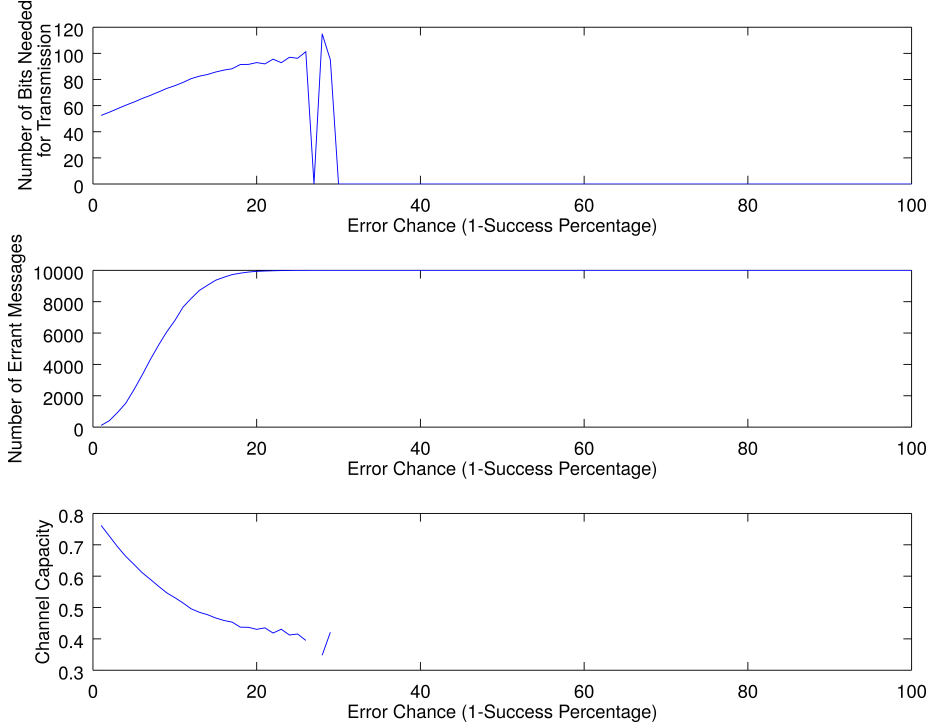Listing 5: Parity Bit Message Inserter

```
1  function msg = generate_random_msg(num_packets, packet_len)
2    msg = zeros(num_packets,packet_len);
3    for i = 1:num_packets
4      for j = 1:packet_len
5        msg(i,j) = floor(2*rand());
6      end
7    end
8  end
```

Listing 6: Generator of Random Messages with Packets

4   Analysis

Because the error correction was implemented using parity bits, the maximum channel capacity possible is $\frac{4}{5}$. This is since one new bit is added to each four bit transmission package thus lengthening the transmitted message at a $\frac{4}{5}$ rate.

The average number of of bits needed follows a roughly linear pattern. This is of course till around 30 when the number of errors invalidates the scheme. It is of notice that around 25 the number of bits needed goes to 0. From analysing the second graph on number of errant messages, we theorize that this tail end is due to the few times that message transmits successfully. Since this rarely happens after 20 according to the second graph, the tail end of the graph is less smooth and sporadic as the random events are smoothed over less and less successful counts.

Since the goal of an error correction scheme is to reduce the number of errors to essentially zero, the second graph reveals that after 1% error rate the number of errors is accountable and is thus too much. Before 1% the number of errors are not accountable and the message is transmitted successfully.

Analyzing the first and third graph, shows the channel is relatively inefficient. Since the parity bit adds 1 bit of information for every 4 bits in our case, the channel capacity is low because we do not need this much information to detect and find errors. I.E. the max information at perfect transmission is 80%. The channel capacity after 16 is less than 50%. However, even before this as mentioned in the second graph, the encoding schemes fails.

5   Conclusion

Parity bits are an easy to implement and fast method for detecting bits in a byte packet. However, they are not very efficient as noted. Not only that, they are not applicable in high noise environments since outside of 1% errors the parity scheme starts to fail to transmit properly. However in low noise environments such as CPU to RAM, parity bits are a good scheme for quick, easily to implement error correction.

Thanks should go to Pete Johnson, my teacher for Information Theory. Also to https://en.wikipedia.org/wiki/Parity_bit for a description of parity bits and how they are calculated.