Contribution:

- BSC channel model -> Malav , Kandarp
- BEC channel model -> Pranav, Kandarp
- Tanner graph -> Kandarp
- Soft BSC -> Kandarp
- Soft BEC -> Kandarp , Krupal
- Hard BSC -> 2 codes:
    1. By Kandarp
    2. By Om , Vushil
- Hard BEC -> Kandarp , try by Krish
- Monte Carlo Simulation -> Kandarp , Krupal
- Algorithm convergence -> Krupal
- Presentation layout , edit -> All except Kandarp , Krupal mainly Krish , Malav
- Presentation content -> Krupal , Kandarp
- Graph plotting in matlab-> Krupal , Kandarp

MVP

- Kandarp , Krupal

Main Learnings

- Accomplishments:
    1. Project related: Given a all zero codeword , passing it through various BSC and BEC channel ; and doing soft and hard decision ; and to measure the performance of the 4

decoders by plotting p_success values vs p (for 3 different H matrices)

2. File handling: .mat(H matrix) -> .csv ; then loading this .csv in C++ ; and then storing the p_success values(output of the decoder) in a separate .csv file ; and then loading this in to matlab and plotting graphs.

- Problems faced:
    1. Even though seemingly correct code had been written from a long time, due to some error it just didn't give the expected output , the problem was actually in the srand() function (used in the BSC and BEC channels) which was used to generate flips or erasures randomly according to the p values.(took almost a week to find the error and correct it)

       Solution: srand() (unsigned int seed) sets the starting for producing a series of pseudo-random integers. I used the srand() in the channel implementation function itself so srand() was running multiple times(and the seed pseudo random variables came out similar and thus gave wrong output) whereas this seed srand() function should be run only one time , so I shifted the line to just below int main()

    2. A general formula or way to calculate to the message passing for CN(j) to VN(i) for SDD(soft decision decoding) , as a CN is a SPC code and it has to send the conditional prob. of VN(i) = 1 for this we have to consider cases that CN has odd

no. of VN's sending prob. corresponding to being 1 . This was difficult to implement
Solution: Using recursive functions prob_odd and prob_even OR using a method mentioned in one of the LDPC papers sir (both used by Kandarp in soft BSC) send OR using for loops(by Krupal in soft BEC) (exact details will be mentioned later in the respective sections of the decoders)

- Novelty of the approach: The whole of the message passing in all the 4 decoders can be understood as happening between just some CN nodes, VN nodes and their connections (or double ended edges as I have used in the codes, depending on the H matrix) ; and we are just passing(or storing) either probs. or values on these 2 sets of edges (CNtoVN and VNtoCN)

Add diagram of Tanner Graph implementation (left?????)

CODE snippets of various parts of tanner graph implementation:

```
struct CN_node
{
    vector<int> edges;
};// CN node contains a vector of index of edges (i.e all VN's it is connected to)

struct VN_node
{
    int r_bit;
    vector<int> edges;
};// VN node contains a vector of index of edges (i.e all CN's it is connected to)
  // and r_bit or r_prob according to whether it is for HDD or SDD

 vector<CN_node> v_CN; // vector points to n_CN CN nodes
 vector<VN_node> v_VN; // vector points to n_VN vN nodes

 for (int i = 0; i < n_CN; i++) {
     v_CN.push_back(CN_node());
 }
 for (int i = 0; i < n_VN; i++) {
     v_VN.push_back(VN_node());
 }
```

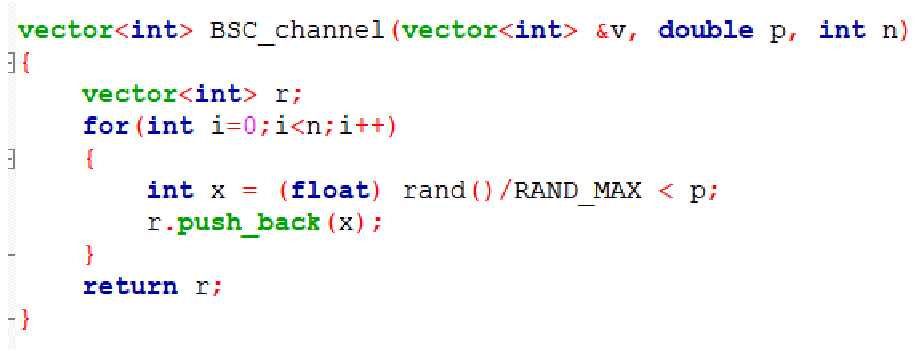
```
vector<int> edge_CnToVn;
vector<int> edge_VnToCn;
int edgeCount = 0;

for(int i=0;i<n_CN;i++) // Creating edge connection according to the 1's in H matrix
{
    for(int j=0;j<n_VN;j++)
    {
        if(H[i][j]==1)
        {
            edge_CnToVn.push_back(-1);
            edge_VnToCn.push_back(-1);
            v_CN[i].edges.push_back(edgeCount);
            v_VN[j].edges.push_back(edgeCount);
            edgeCount++;
        }
    }
}
```

- Some insights not known earlier
    1. File handling, and using matlab and c++ side by side
    2. The CN to VN message passing for soft using the method described in the LDPC paper

- Alternate ways of decoding: Probably doing box decoding like in the product code (still doubt)???

Implementation of Decoder and the Channel:

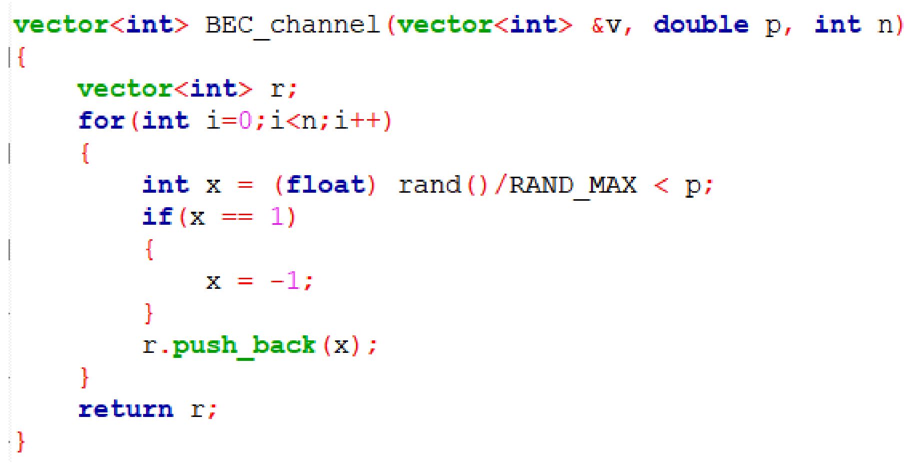
- BSC model:



```
vector<int> BSC_channel(vector<int> &v, double p, int n)
{
    vector<int> r;
    for(int i=0;i<n;i++)
    {
        int x = (float) rand()/RAND_MAX < p;
        r.push_back(x);
    }
    return r;
}
```

Here, a vector m of all 0’s (all zero codeword is only passed through the BSC channel) is given to the BSC channel; p->prob. of bit flip in the BSC channel and n -> size of transmitted message vector m (= no. of VN’s = no. of columns of the H matrix) According to the values of p the value r’s(received bits) can be either 0 or 1.

- BEC model:



```
vector<int> BEC_channel(vector<int> &v, double p, int n)
{
    vector<int> r;
    for(int i=0;i<n;i++)
    {
        int x = (float) rand()/RAND_MAX < p;
        if(x == 1)
        {
            x = -1;
        }
        r.push_back(x);
    }
    return r;
}
```

Here, p -> prob. of erasure in the BEC channel. According to the values of p the values of r’s(received bits) can be either 0 or -1(erasure) as we have assumed that only an all zero codeword is passed

- LDPC hard BSC:

1. First by the tanner graph implementation as shown previously, the connections(edges) between CN’s and VN’s is made according to the given H matrices.

2. And in the decoding, first the transmitted message vector m (of all 0’s) is passed through the BSC channel and we get the received bits vector (according to the noise controlled by p).

3. Then the first VN to CN message passing (iteration -> 0) occurs in which simply the bits received at the VN’s (after passing through the BSC channel) are passed on the corresponding edges VNtoCN for all the edges connected to the VN.

4. To avoid repetitive code, for all the coming iterations (iterations from 1 -> itr_MAX) first CN to VN, then VN to CN and then c_hat checking is done (to keep it in immediate continuation after the iteration = 0, VN to CN message passing)

5. In VN to CN message passing, we do majority voting, if more no. of bits (from the CN’s other than the one VN is sending the message to) -> received bit + CN bits are 1’s then send 1 else send 0.

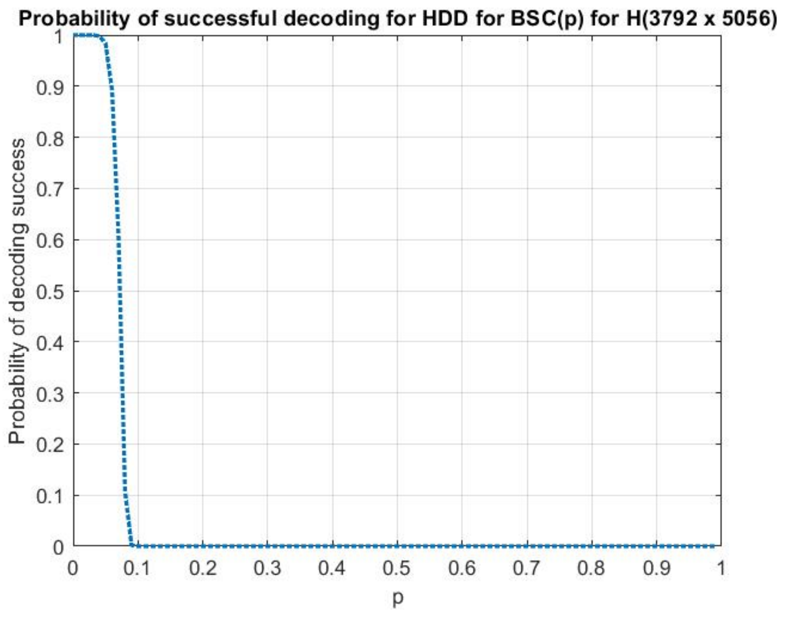
6. In CN to VN message passing we simply do SPC(single parity check) decoding which is equivalent to just finding the modulo 2 sum of the bits (of the VN’s other than the one CN is sending the message to).

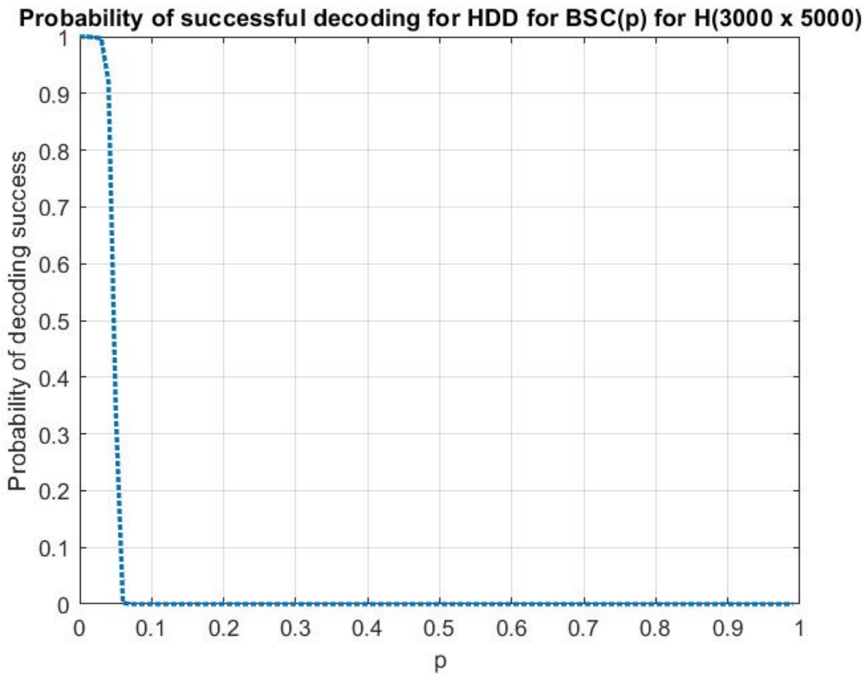
7. Then we make the estimate the codeword, by finding the c_hat vector where each of its bits is just the majority voting of bits sent by all the CN's the corresponding VN is connected to.

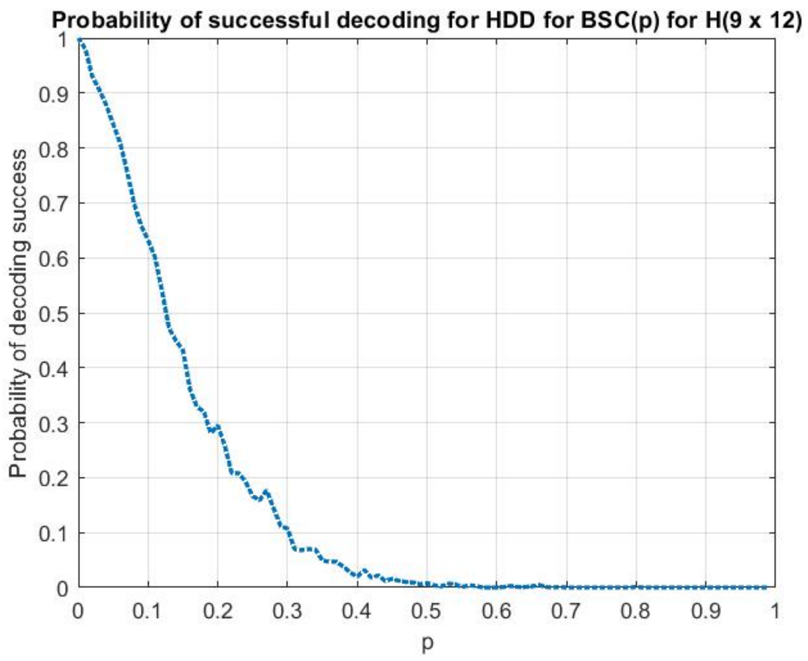
8. If the estimate of the c_hat codeword vector comes same in 2 consecutive iterations then we terminate.

9. Finally, after termination we compare the estimated codeword(c_hat) and the transmitted message vector m; if they are same Ncorr++ else Nerr++

10. The above steps are done for Nsim simulations (for a particular value of p) ; then the p_success = Ncorr/Nsim is found.




Probability of successful decoding for HDD for BSC(p) for H(3000 x 5000)
Probability of decoding success
p
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Probability of successful decoding for HDD for BSC(p) for H(9 x 12)
Probability of decoding success
p
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

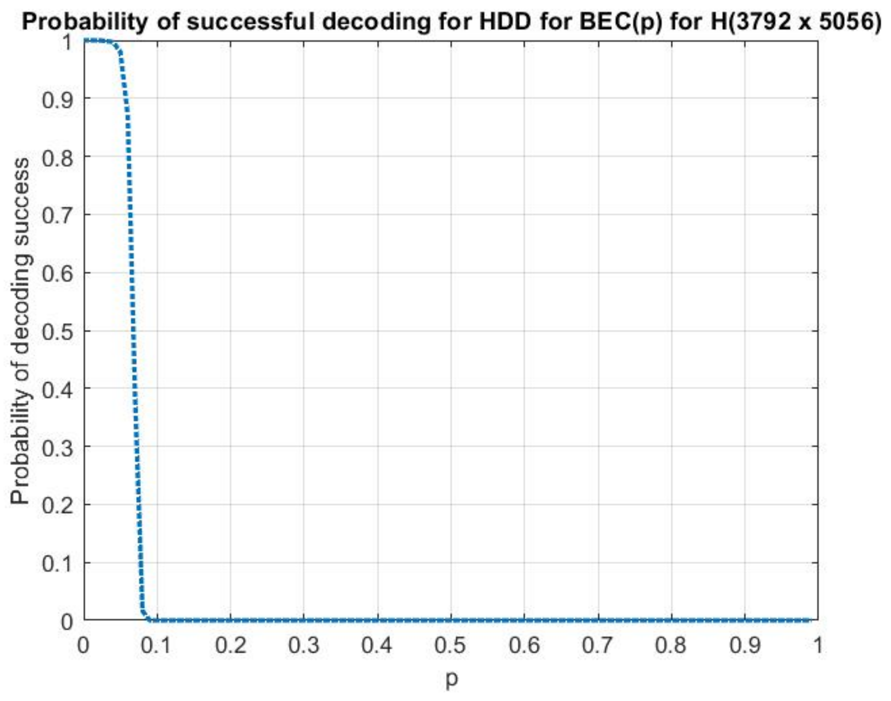
- LDPC hard BEC: (erasure -> -1)

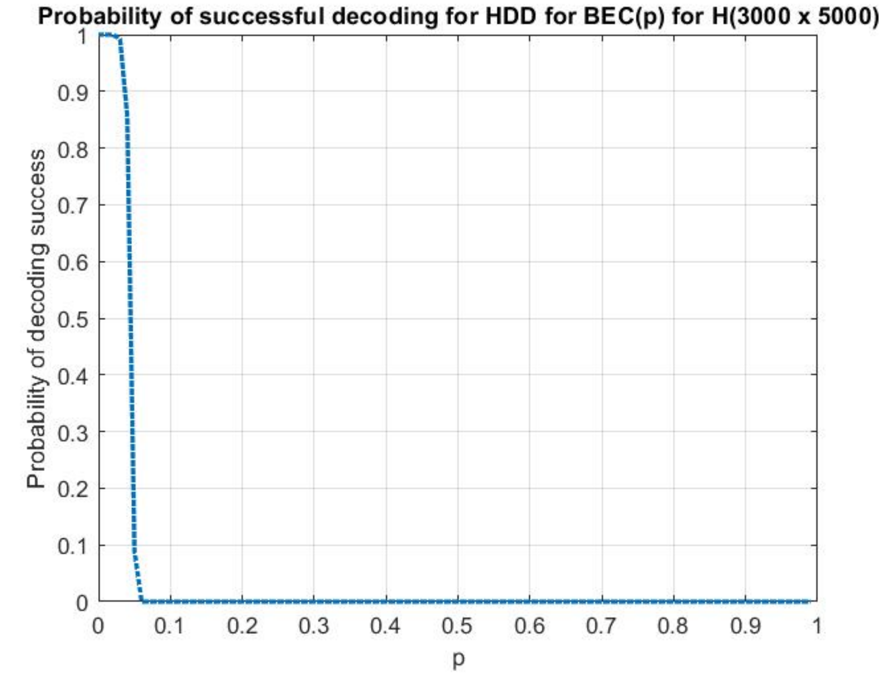
1. The first 4 steps are the same as in LDPC hard BSC except here the all zero codeword is passed through the BEC channel.

2. In VN to CN message passing, we do majority voting, if more no. of bits (from the CN's other than the one VN is sending the message to) -> received bit + CN bits are 0's then send 0 else send erasure (-1).

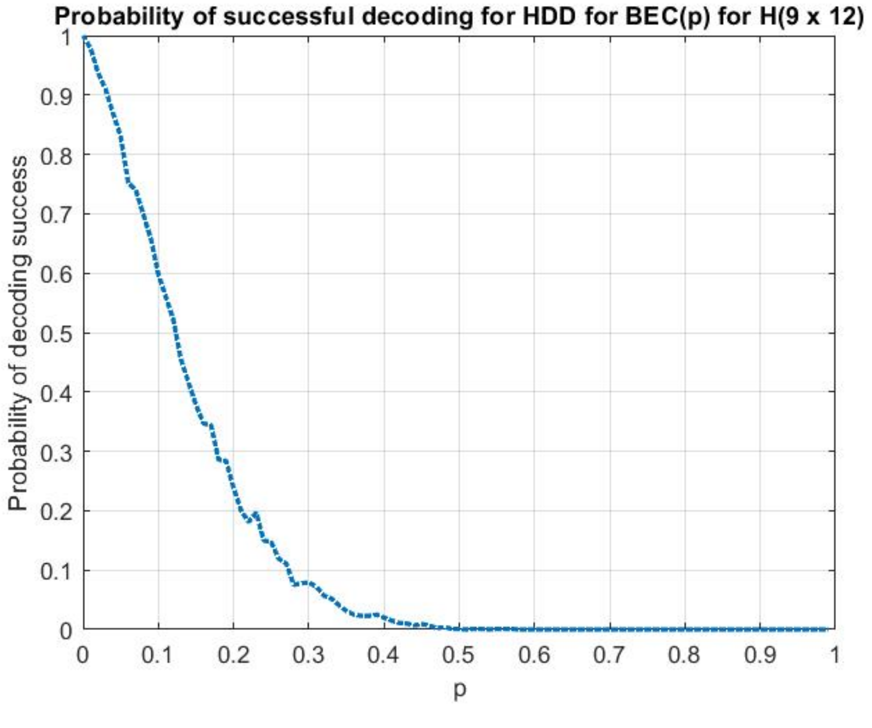
3. In CN to VN message passing if even any of the VN sends a erasure (-1) then send erasure (-1) from the CN to VN else we simply do SPC decoding (of the VN's other than the one CN is sending the message to).

4. The rest of the steps are same as the steps 7 to 10 as in LDPC hard BSC.





Probability of successful decoding for HDD for BEC(p) for H(3000 x 5000)
Probability of decoding success
p
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



Probability of successful decoding for HDD for BEC(p) for H(9 x 12)
Probability of decoding success
p
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

- LDPC soft BSC:

1. The first 2 steps are the same as in LDPC hard BSC

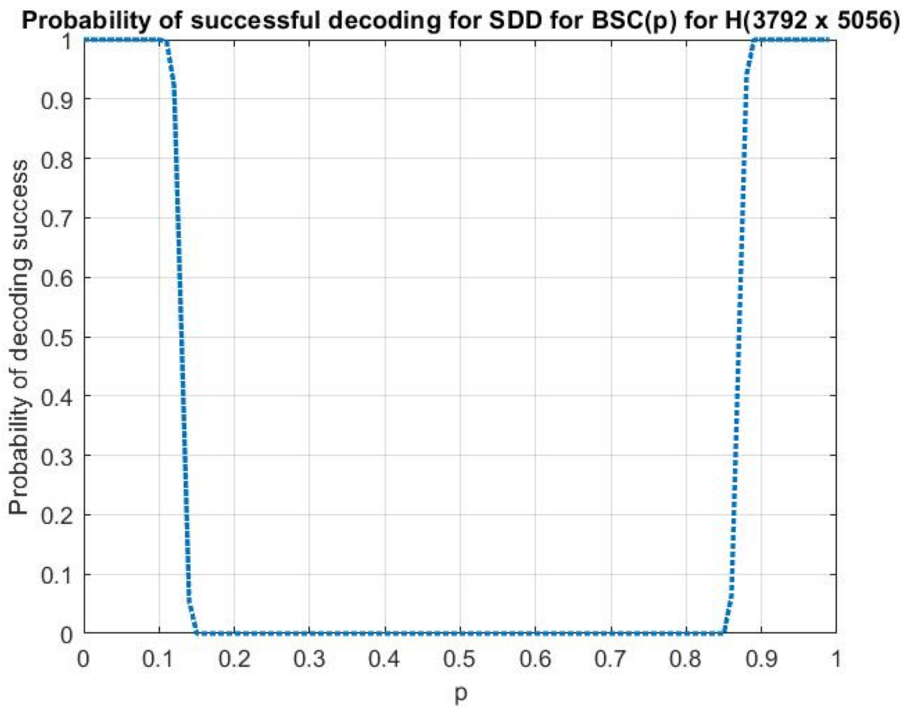
2. Then the first VN to CN message passing (iteration -> 0) occurs in which we simply pass the probabilities received at the VN's to the corresponding edges VNtoCN for all the edges connected to that VN. The probabilities loaded in each VN is according to the corresponding bits received after passing the message vector through the BSC channel.
   a) If received bit = 1, then p_i = 1-p is loaded
   b) Else p_i = p is loaded
      Where p_i denotes the conditional prob. that ith VN is 1 given the received bit r_i

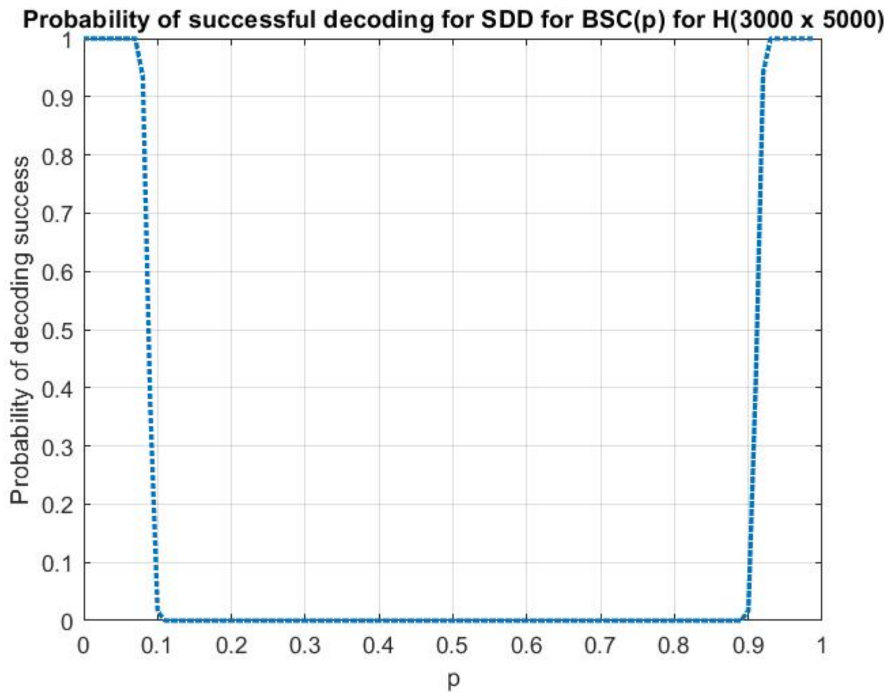
3. In CN to VN message passing jth CN sends the probability that ith VN (to which it is sending the message to) is equal to 1. For this we need to find the probability that jth CN sends 1 , and since CN do SPC decoding , for it to send a 1 it should receive probabilities corresponding to odd 1's from the remaining $d_c – 1$ VN's

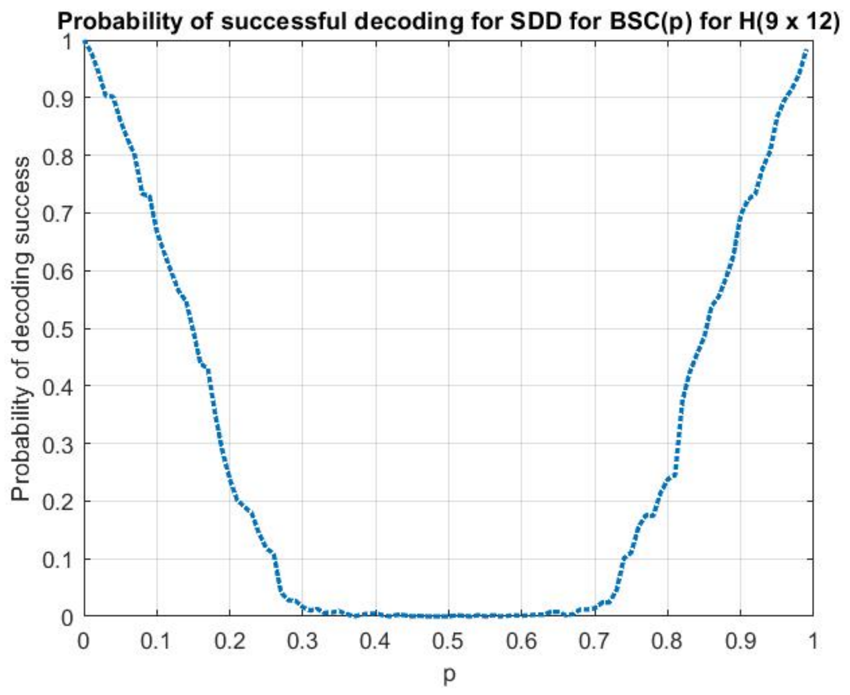
4. For a general iteration (i.e., after 1st iteration) of VN to CN message passing we send the probability that ith VN is 1 to the jth CN by taking probability messages from the other $d_v – 1$ CN's.

5. Then we estimate the codeword, by finding the c_hat vector where each of its ith bit found by finding the odds in the favor of the ith VN being equal to 1. If this odds is >= 1 then estimated ith c_hat bit is 1 else 0.

6. The rest of the steps are same as the steps 8 to 10 as in LDPC hard BSC.







Monte Carlo Simulations:

How you performed Monte Carlo simulation of decoder + channel.

First, we run a for loop for different values of p from 0 to 1 in steps of 0.01, inside this for loop we run Nsim = 1000 simulations, then for each simulation, the channel (BSC or BEC) is run once and then the decoding (soft or hard) is carried out. For Nsim simulations we find the no. of instances where the decoder is able to correctly decode the codeword, and get the value of p_success = Ncorr/Nsim for each value of p.

What was the speed of your simulator?

- For all
    1. For all the 4 schemes:
        a) Nsim = 1000, and p ranges from 0 to 1 in steps of 0.01

    b) and also, the decoding of 9 x 12 matrix only took a couple of seconds

2. For soft BSC each of the 2 H matrices (3792 x 5056 & 3000 x 5000) took approx. about 1031 seconds = 17.13 mins each.

3. For soft BEC each of the 2 H matrices took approx. about 2288 seconds = 38.13 mins each.

4. For hard BEC each of the 2 H matrices took approx. about 621.5 seconds = 10.35 mins each.

- ideas to make the simulator more efficient – both in the memory usage and the execution time?
    1. At first, I thought of tanner graph implementation by maintaining 2 matrices (no. of CN's x no. of VN's) to store bit or probability values for CN to VN and VN to CN message passing by going through the H matrix every time, this would have increased the time and space complexity by a large amount.

    2. So, I used the current implementation of tanner graph in the codes. Since these LDPC matrices are very sparse (but still of large size) with 1's (i.e., very less no. of CN's and VN's as compared to H matrix size) it is better to go through the H matrix only once and then making empty CN's and VN's and making connections just once; and maintaining the 2 edge vectors VN to CN and CN to VN, so next time we could directly access the required bits or

probabilities for the message passing from these relatively less no. of edges than having to go through the H matrix each time.

3. Also, we are overwriting the bit or probability values in the edges as we don't need to store their old values.

4. Another thing is that in soft BSC I had initially used recursive functions for CN to VN message passing, but then I used method or rather the formula illustrated in the LDPC paper by Bernhard M.J Leiner:

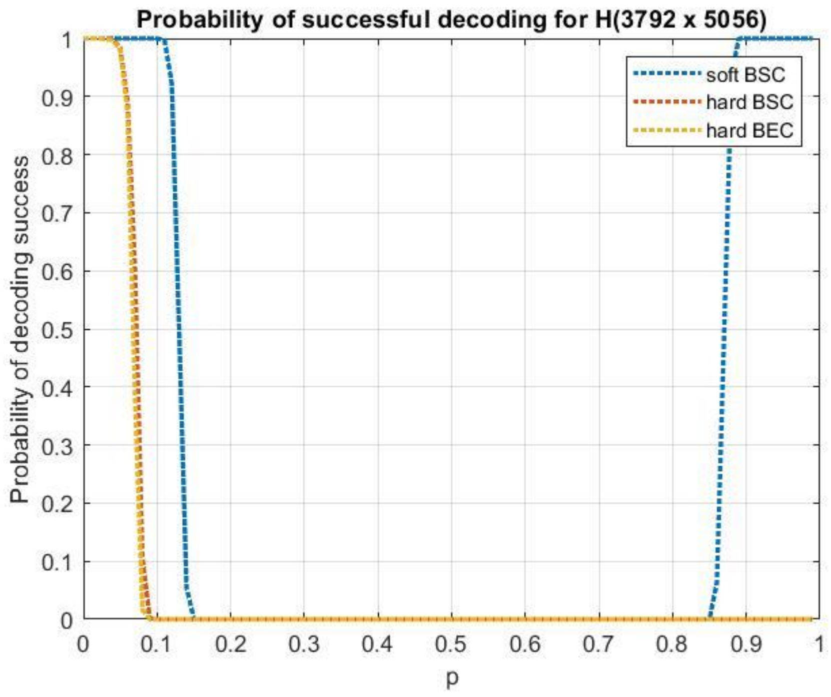
$$r_{ij}(0) = 1/2 + 1/2 \prod (1- 2q_{i'j}(1))$$

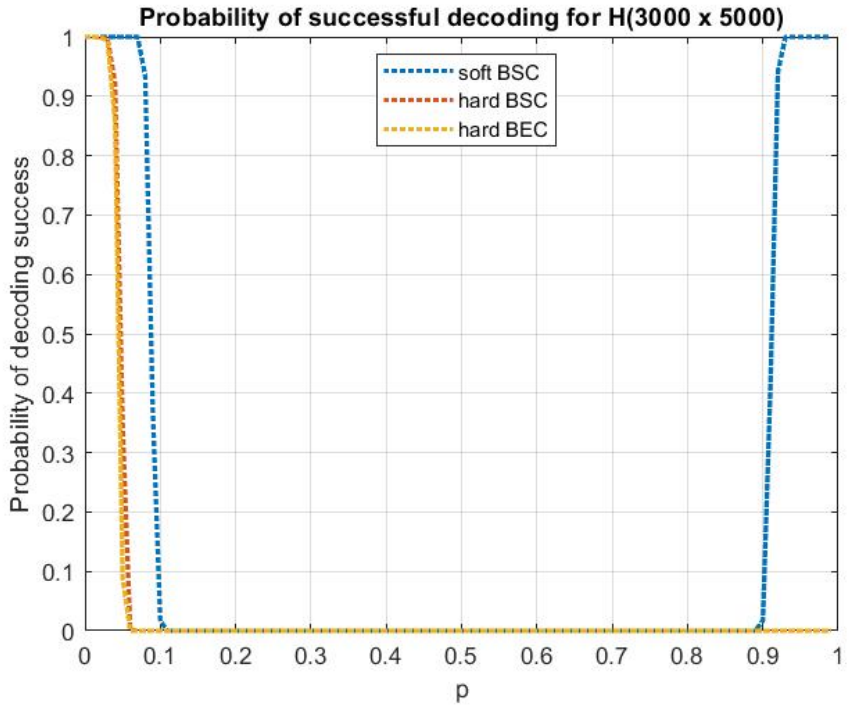
and

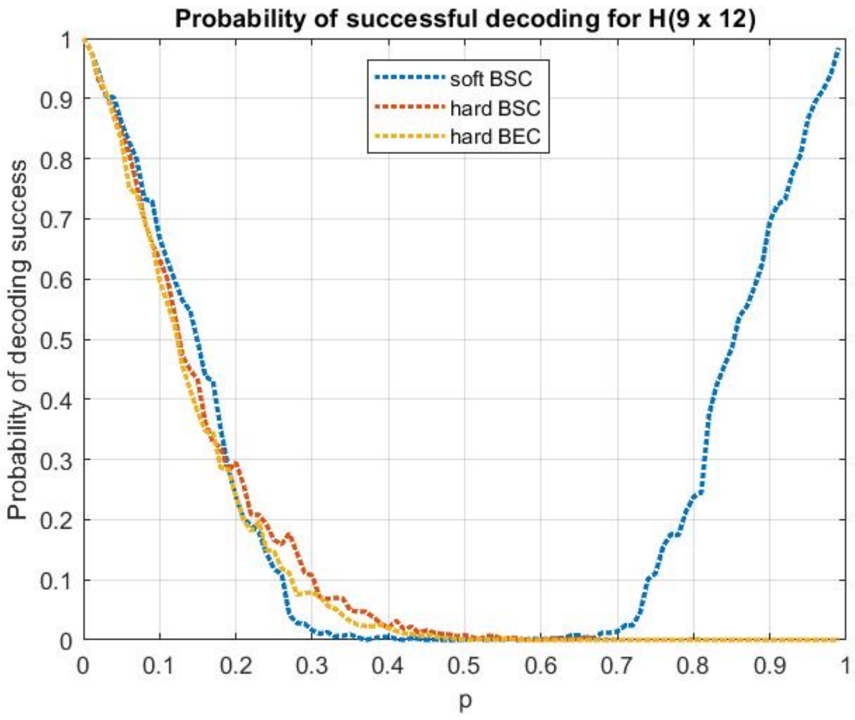
$$r_{ij}(1) = 1 - r_{ij}(0)$$

This doesn't change the time required to run simulations for soft BSC drastically but it stills take lesser time than when the recursive functions are used.

Numerical Results:






Probability of successful decoding for H(9 x 12)
Probability of decoding success
p
soft BSC
hard BSC
hard BEC
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1