CT111 Project

Presented by team 6 of group 6

Date: 19th July 2021.

Honor Code

We declare that

- The work that we are presenting is our own work. We have not copied the work (the code, the results, etc.) that someone else has done.
- Concepts, understanding and insights we will be describing are our own.
- We make this pledge truthfully. We know that violation of this solemn pledge can carry Grave consequences.

Signatures of Team members

Information is the resolution of uncertainty.

-Claude Shannon

Contribution of Members

Task	Name	ID	Contribution
BEC & BSC Channel Coding	Kunal Hotwani Smit Bhavsar Om Patel	202001468 202001464 202001462	Designed channel for generating random Message.
BEC Hard decision decoder	Om Patel Kalp Pandya Kunal Hotwani	202001462 202001466 202001468	Designed algorithm and implementation in c++
BSC Hard decision decoder	Om Patel Chintan Kanpariya Raj Shah	202001462 202001463 202001460	Designed algorithm and implementation in c++

Contribution of Members

Task	Name	ID	Contribution
BEC Soft decision decoder	Om Patel Jay Navadiya Jay Grover	202001462 202001465 202001467	Designed algorithm and implementation in c++
BSC soft decision decoder	Om Patel Jay Navadiya Kalp Pandya	202001462 202001465 202001466	Designed algorithm and implementation in c++
Presentation	Jay Navadiya Raj Shah Harsh Agheda	202001465 202001460 202001461	Content Creation, Designing and Animations.

Contribution of Members

Task	Name	ID	Contribution
Monte-Carlo Simulations	Jay Navadiya Chintan Kanpariya Harsh Agheda	202001465 202001463 202001461	Did most number of simulations and derived data for graphs
Graph Plotting & Matrix Loading	Kunal Hotwani	202001468	Plotted graphs in MATLAB with the data generated, Loaded H matrices in C++.

MVP of Our Team

Om Patel
(202001462)

Special Mentions

Jaykumar Navadiya (202001465)
Kunal Hotwani (202001468)
Kalp Pandya (202001466)

The Main Learnings

Main Learnings

01

LDPC Codes

02

Debugging skills

03

Patience

04

How to work in a Team?

Difficulties we faced

- We did not have direct functions for Matrix and XOR operations because we decided to use C++.
- Matrix computations were therefore harder to perform in C++.
- Running Monte Carlo simulations sometimes crashed our computers and sometimes terminal was forcefully stopped by the computer giving us partial result of the simulations.
  (showed zsh: killed)
- Apart from this we also faced many issues which we debugged later on.

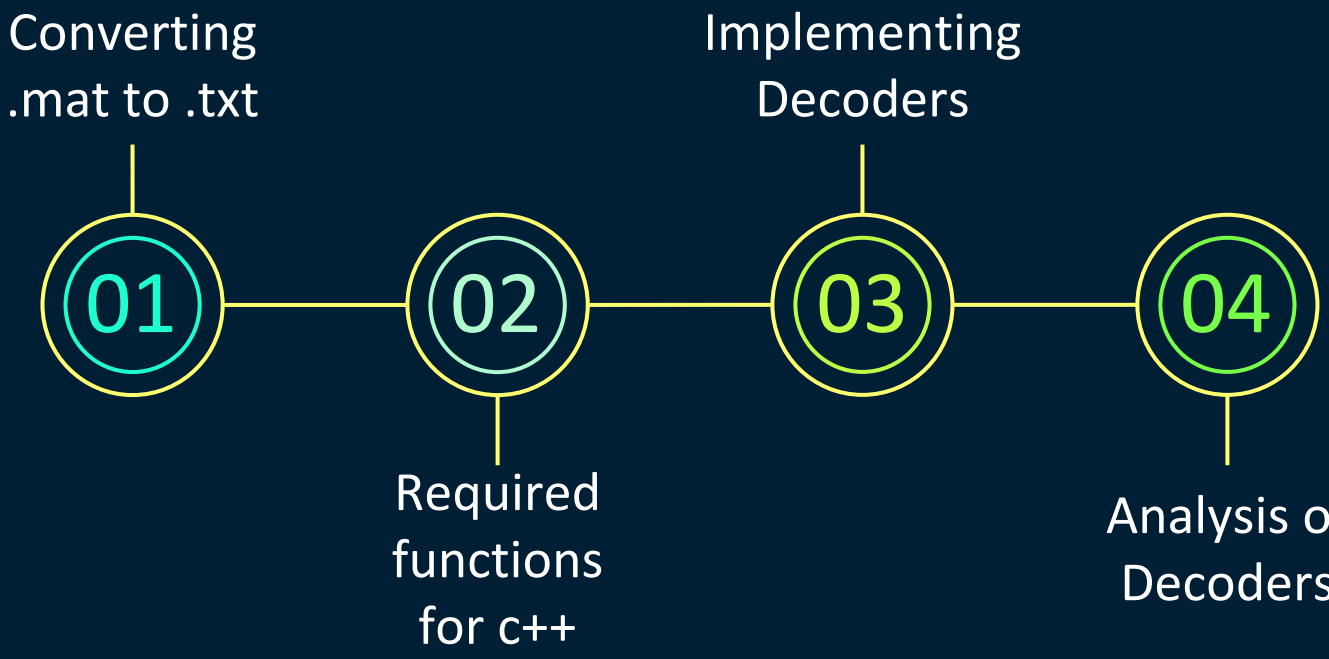
Solution to the problems

- We performed the matrix operations manually by using loops in C++.
- Divided Monte-Carlo simulations in parts for using time efficiently. (0-0.3, 0.31-0.6,0.61-1)

Major Accomplishments (Novelty of our approach)

- We were able to obtain great time efficiency because of our algorithms expect BSC hard decision decoding because we did not had to store messages of VN to CN or CN to VN anywhere else. We did that by only using h matrix itself. It reduced running time by quite a lot.

Roadmap for Project



Converting .mat to .txt

- We had to convert .mat files to .txt files as there is no way to load .mat into c++ directly.
- To do so, we opened matlab and used dlmwrite function.
- This function writes the data in the .mat file to a .txt file.
- We then used this .txt file to read the data in C++.

trail_codes > H.txt

```
1,0,0,0,0,1,0,1,0,1,0,0
1,0,0,1,1,0,0,0,0,0,1,0
0,1,0,0,1,0,1,0,1,0,0,0
0,0,1,0,0,1,0,0,0,0,1,1
0,0,1,0,0,0,1,1,0,0,0,1
0,1,0,0,1,0,0,0,1,0,1,0
1,0,0,1,0,0,1,0,0,1,0,0
0,1,0,0,0,1,0,1,0,1,0,0
0,0,1,1,0,0,0,0,1,0,0,1
```

Loading .txt file in C++

```
vector< vector<double> > h(row, vector<double>(column, -1));
ifstream file;

file.open(filename);
// H matrix will be now stored in h 2d array.
for (int i = 0; i < row; i++)
{
  int j = 0;
  char tempchar;
  for (j = 0; j < column - 1; j++)
  {
    file >> tempchar;
    h[i][j] = (tempchar - '0');
    file>> tempchar;
  }
  file >> tempchar;
  h[i][j] = (tempchar - '0');
}
file.close();
```

- We used the code given on the left for loading .txt files in C++.
- It was stored in a 2d vector (Matrix) named h.

Channels

BEC Channel (Soft)

```
double ax = q*100;
int tx = 100;

double *r = new double[m];
for(int i=0;i<m;i++)
{
  int bx = random(tx);
  if(bx<=ax)
  {
    r[i] = 0.5;
  }
  else
  {
    r[i] = c[i];
  }
}
```

BSC Channel

```
int *r = new int[m];
double ax = q*1000;
int tx = 1000;
for(int i=0;i<m;i++)
{
  int bx = random(tx);
  if(bx<=ax)
  {
    r[i] = (c[i]+1)%2;
  }
  else
  {
    r[i] = c[i];
  }
}
```

Random function

```
int random(int limit)
{
  random_device rd;
  mt19937 gen(rd());
  uniform_int_distribution<> dis(1,limit);
  return dis(gen);
}
```

Decoding Algorithms

BEC Hard Decision Decoding

- At the very first,we will load the variable nodes with message from the channel.
- After this, all the variable nodes will pass the message to their respective check nodes.
- Then we will start iteration through check nodes.
- If there is one erasure in the check node then,the value of that erased node will be **modulo two sum** of all other variable nodes values in the check nodes.
- If there is more than one erasure then the erasure is not solvable.
- We will return this value to the connected variable node.
- If there is no erasure present in all of the variable nodes or if the number of iterations exceeds the given maximum number of iterations then we will break the loop.
- After this, it is checked if all the erasures are removed. If there is an erasure present then we will consider that the message was not decoded successfully.

BSC Hard Decision Decoding

- We just load values of received message as the value of corresponding VN for the zeroth iteration.
- Further, we send message[1] from each VN to all connected CN and calculate decision bit for corresponding VN[2].
- Next, we send message[3] from CN to VN.
- Lastly, we check whether codeword gets updated from previous iteration. If codeword does not gets updated, we break the loop else we move to next iteration with updated values for VNs until we have reached iterations limit.
- When we have exited the loop either due to no updation or iteration limit reached, we check whether received and transmitted codewords are same or not. If they are same, we are assured that the codeword is successfully decoded.

[1] Majority vote of d_v – 1 messages and message received over channel

[2] Majority vote of d_v CNs and received message over channel

[3] Modulo 2 sum of d_c – 1

BSC Soft Decision Decoding

- We just load values of VNs with probabilities from received codeword from channel for zeroth iteration.
- Further, we send message[1] from each VN to all connected CN and calculate decision bit for corresponding VN[2]. Next, we send message[3] from CN to VN. The message will be calculated by the formulae:

$$r_{ji}(0) = \frac{1}{2} + \frac{1}{2} \prod_{i' \epsilon V_j \setminus i} (1 - 2q_{i'j}(1))$$

- Lastly, we check whether codeword gets updated from previous iteration. If codeword does not gets updated, we break the loop else we move to next iteration with updated beliefs for VNs until we have reached iterations limit.

$$q_{ij}(0) = K_{ij} (1 - P_i) \prod_{j' \epsilon C_i \setminus j} r_{j'i}(0)$$

- When we have exited the loop either due to no updation or iteration limit reached, we check whether received and transmitted codewords are same or not. If they are same, we are assured that codeword is successfully decoded.

BSC Soft Decision Decoding

[1] VN sends probability of itself happening 0 when value at d_v – 1 check nodes and receiving channel are all 0.

[2] Decision is set to 1 when λ_1 (product of likelihoods that VN is 1 for d_v CNs and receiving channel) is greater equal to 1.

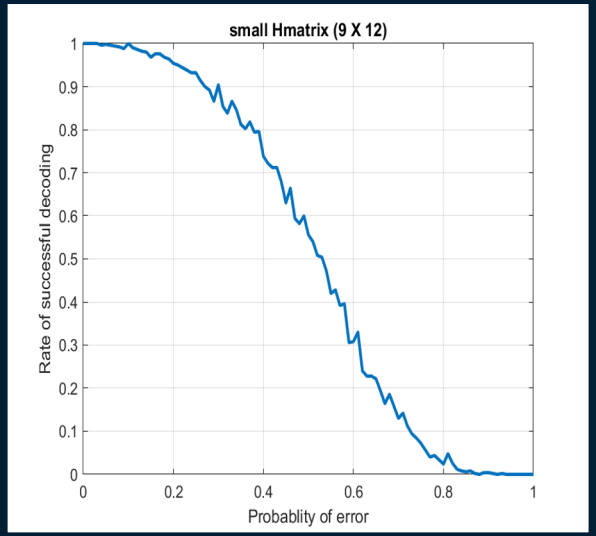
[3] CN sends probability of VN happening 0 that accounts for different cases of d_c – 1 VNs connected to it.

BEC Soft Decision Decoding

- At the very first,we will load the variable nodes in the array Respectively.if r_i 0 it will send 0 else if r_i 1 it will send 1 else it will send 0.5.
- Everything else is just same as BSC Soft decision decoding and we will remove erasure if λ is not equal to 1.
- Breaking conditions will be when there are no erasures or the iterations are completed.

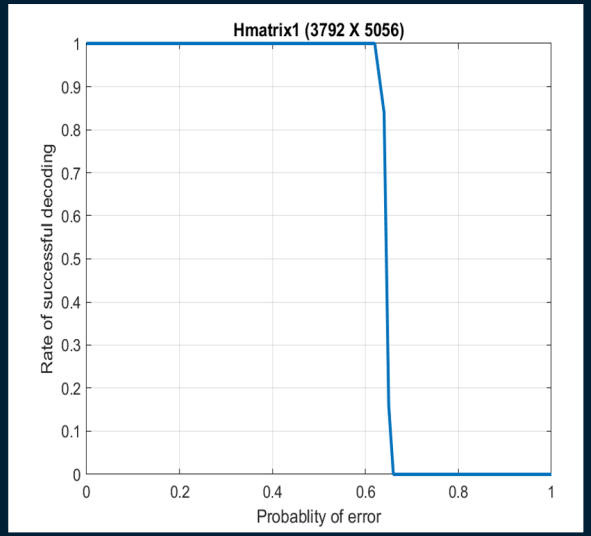
Numerical Results

BEC Hard Decision Decoding



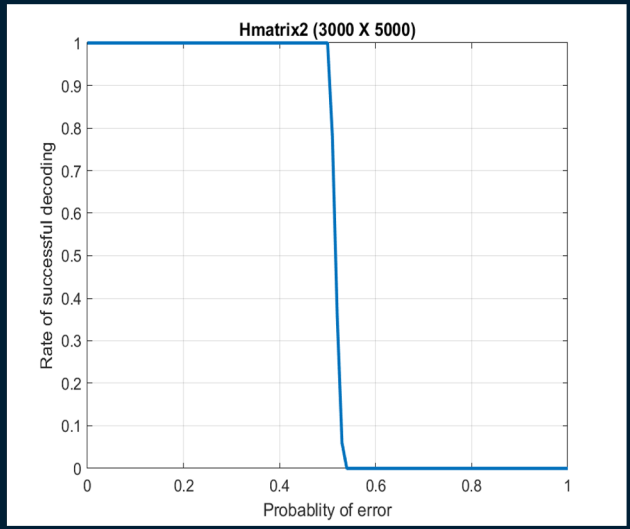
Small H matrix (9 x 12)(3,4)

Drops to 0 at P_{error}=1.



Hmatrix1 (3792 x 5056)(4,3)

Drops to 0 at P_{error}=0.62.

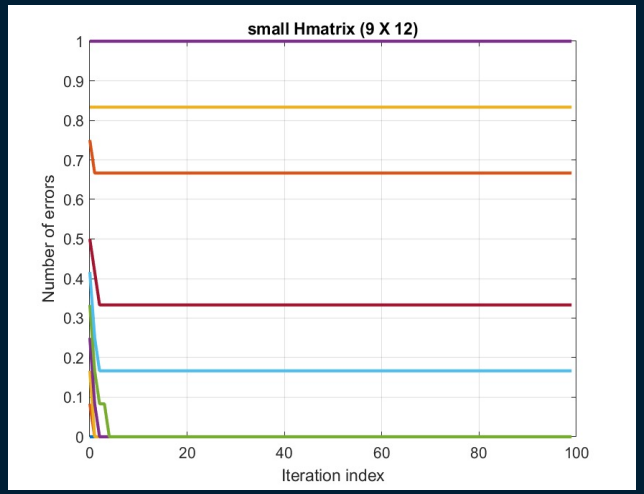


Hmatrix2 (3000 x 5000)(5,3)

Drops to 0 at P_{error}=0.51.

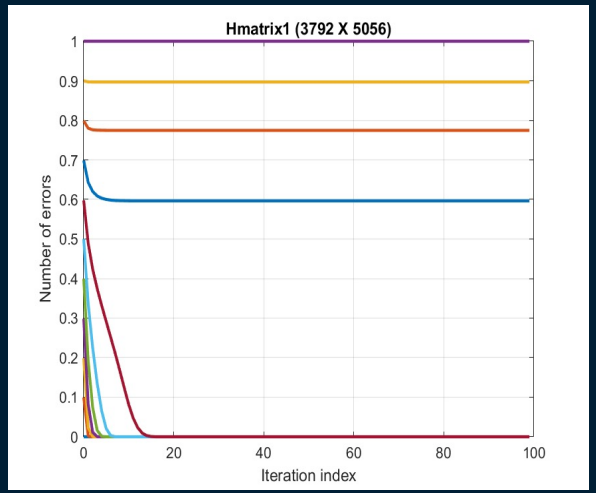
BEC Hard Decision Decoding

Algorithm Convergence Graphs



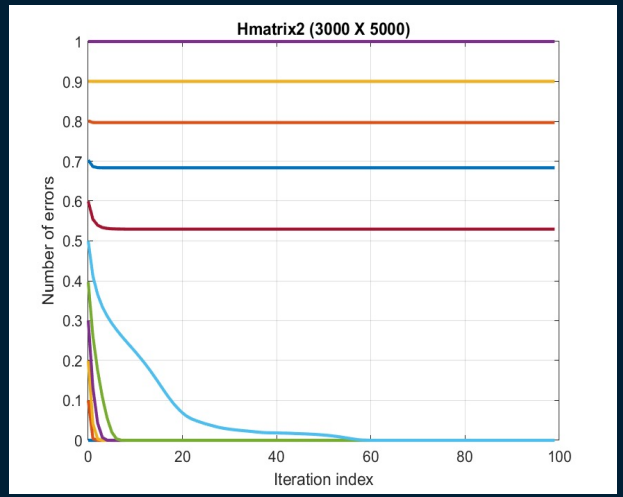
Small H matrix (9 x 12)(3,4)

Drops to 0 at P_{error}=1.



Hmatrix1 (3792 x 5056)(4,3)

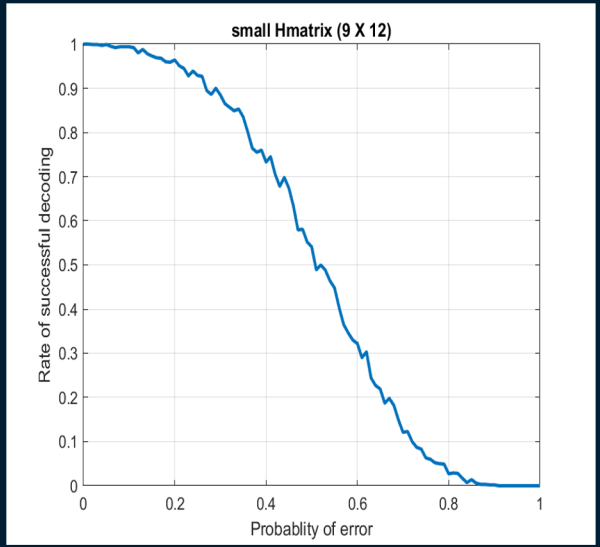
Drops to 0 at P_{error}=0.62.

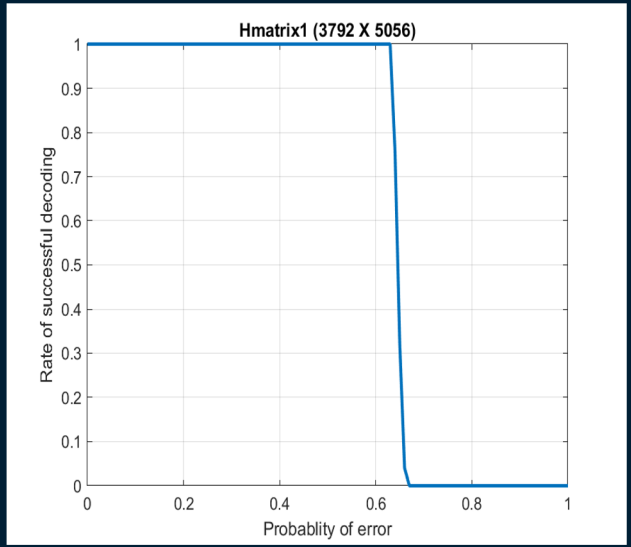


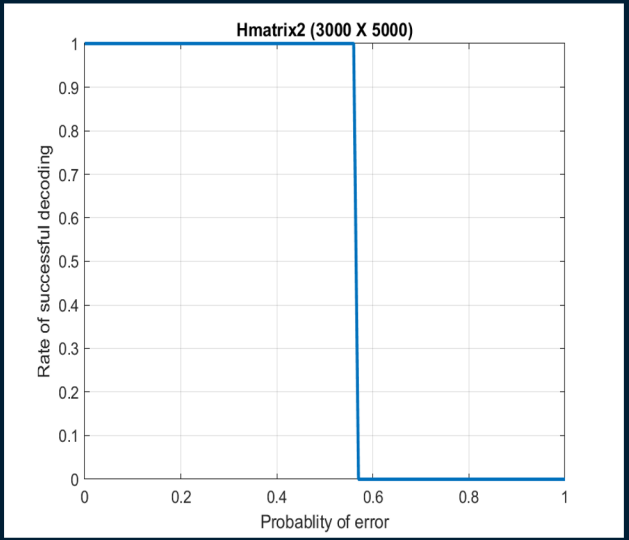
Hmatrix2 (3000 x 5000)(5,3)

Drops to 0 at P_{error}=0.51.

BEC Soft Decision Decoding







Small H matrix (9 x 12)(3,4)

Drops to 0 at P_{error}=1.

Hmatrix1 (3792 x 5056)(4,3)

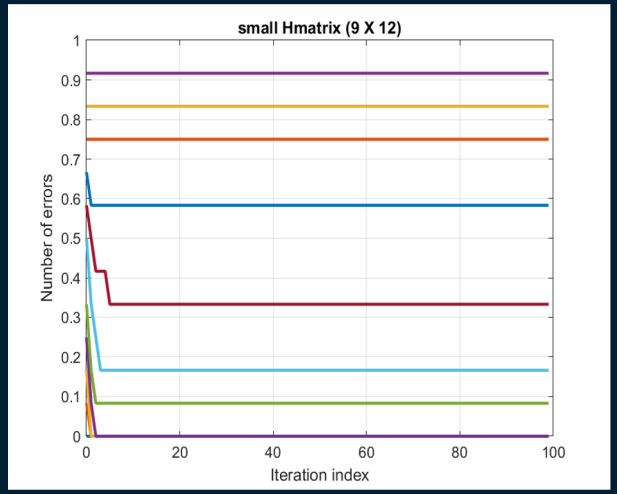
Drops to 0 at P_{error}=0.63.

Hmatrix2 (3000 x 5000)(5,3)

Drops to 0 at P_{error}=0.57.

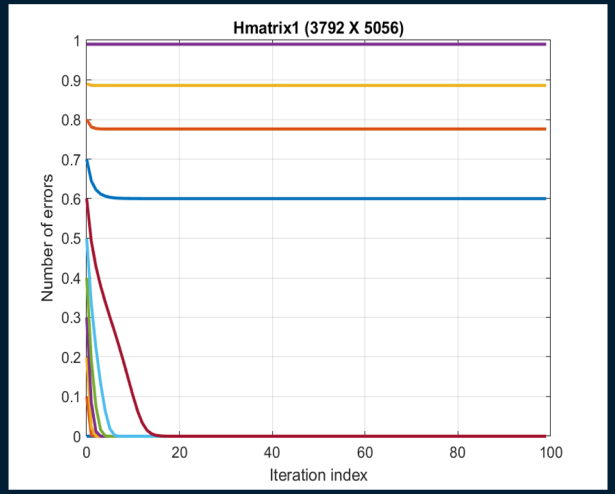
BEC Soft Decision Decoding

Algorithm Convergence Graphs



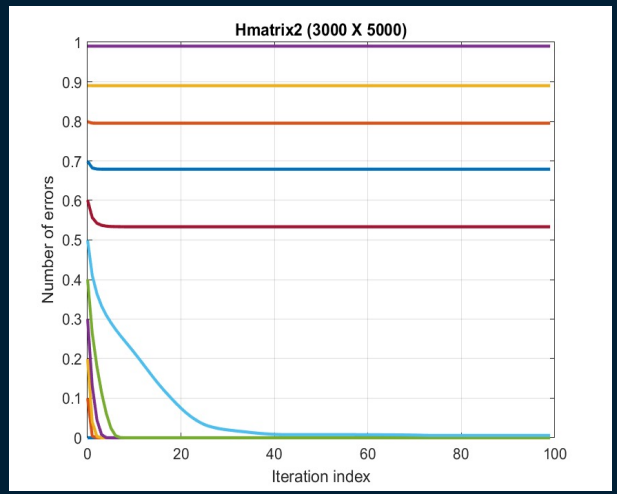
Small H matrix (9 x 12)(3,4)

Drops to 0 at P_{error}=1.



Hmatrix1 (3792 x 5056)(4,3)

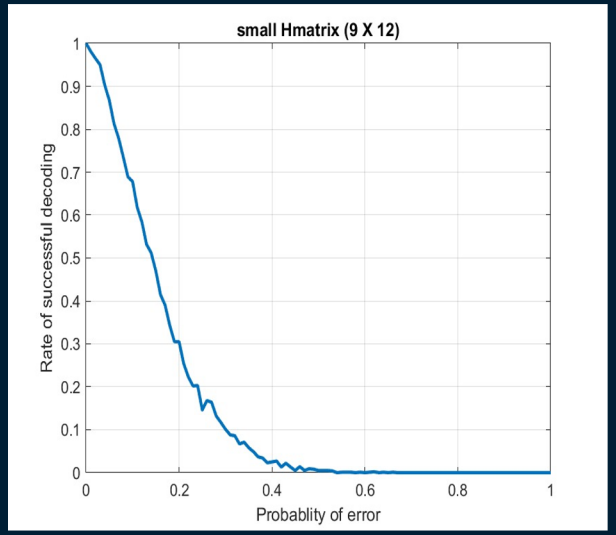
Drops to 0 at P_{error}=0.63.

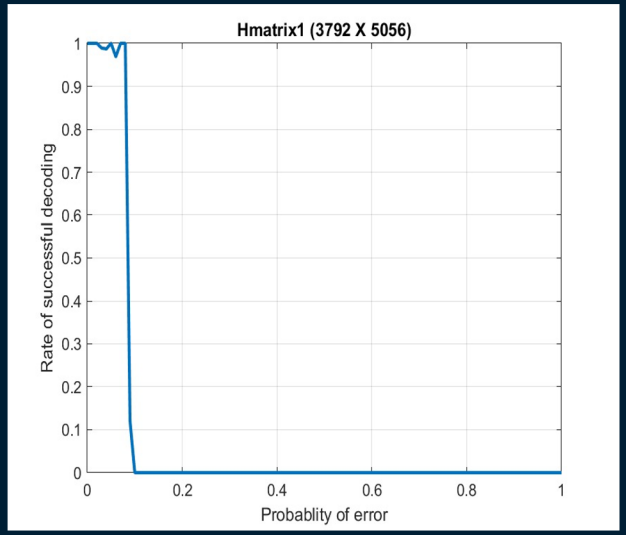


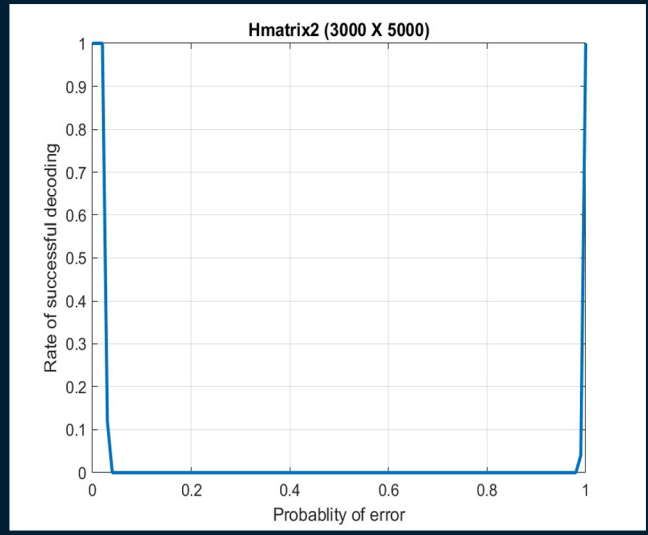
Hmatrix2 (3000 x 5000)(5,3)

Drops to 0 at P_{error}=0.57.

BSC Hard Decision Decoding







Small H matrix (9 x 12)(3,4)

Hmatrix1 (3792 x 5056)(4,3)

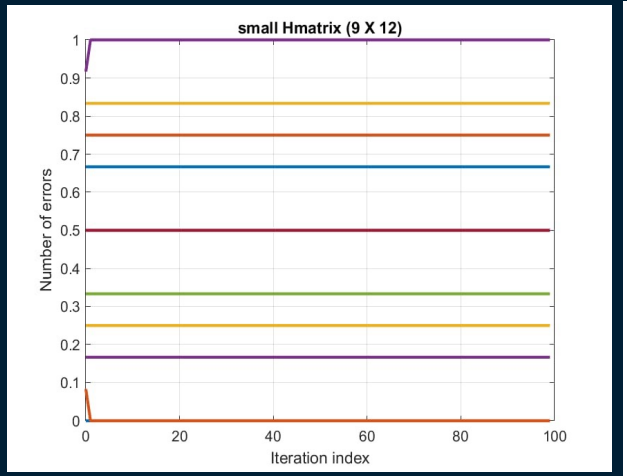
Hmatrix2 (3000 x 5000)(5,3)

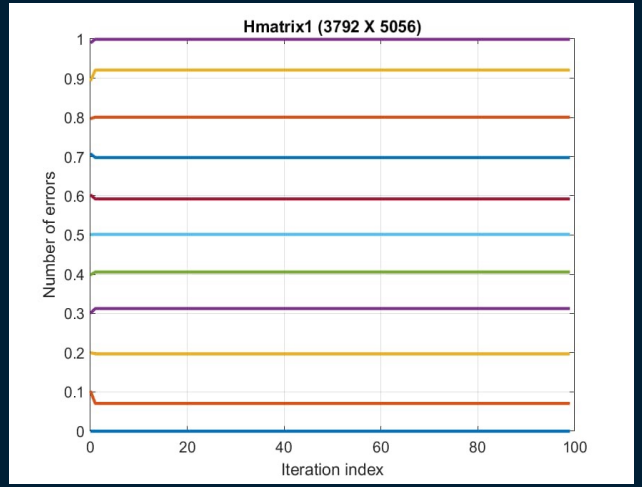
Here small H matrix and Hmatrix1's have their check node's degree even. That's why they will not be decoded at P_{error}=1.

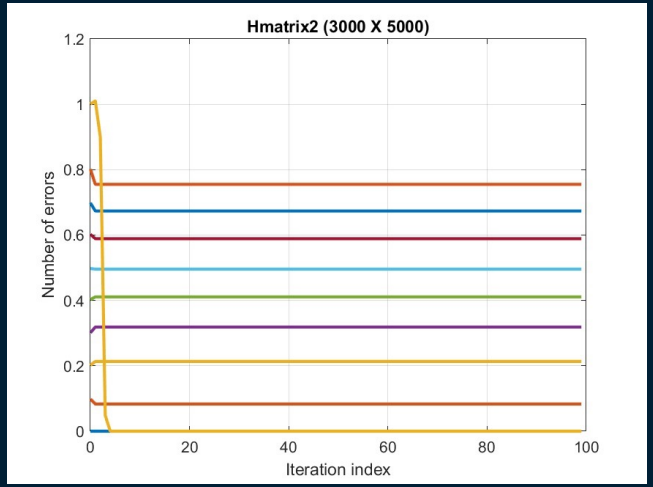
As we can see, BSC is giving much worse efficiency compared to BEC decoders

BSC Hard Decision Decoding

Algorithm Convergence Graphs







Small H matrix (9 x 12)(3,4)

Drops to 0 at P_{error}>0.5.

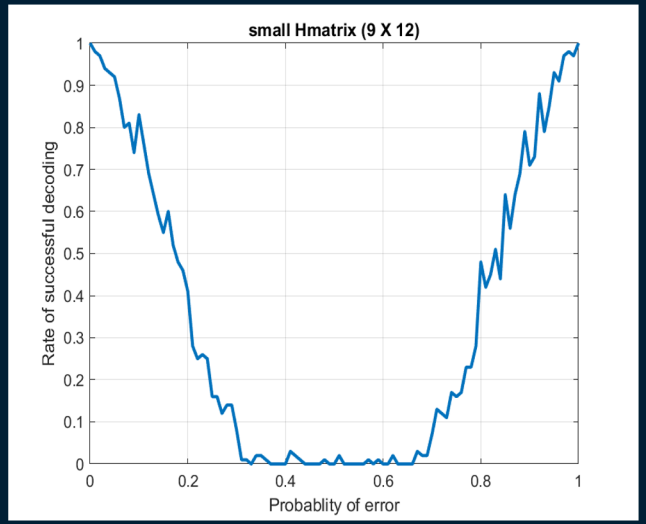
Hmatrix1 (3792 x 5056)(4,3)

Drops to 0 at P_{error}=0.09.

Hmatrix2 (3000 x 5000)(5,3)

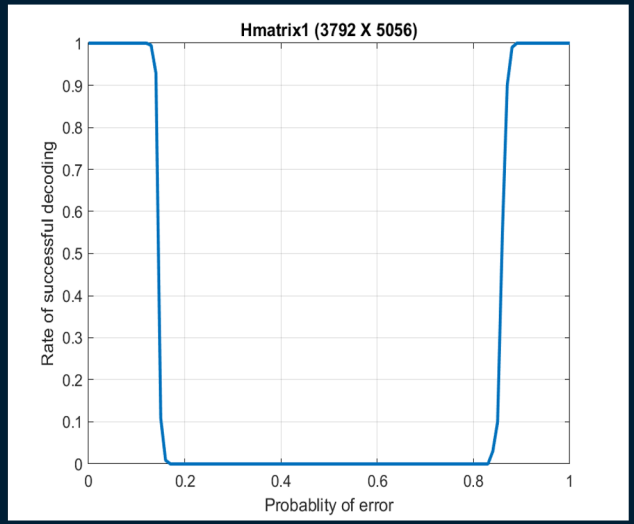
Drops to 0 at P_{error}=0.04.

BSC Soft Decision Decoding



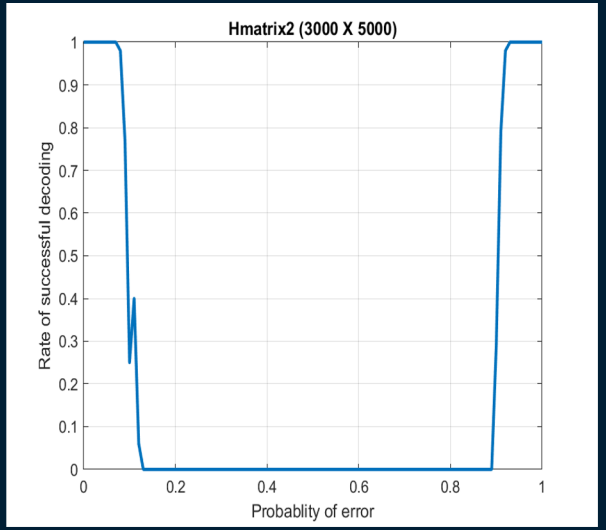
Small H matrix (9 x 12)(3,4)

Drops to 0 at $0.3<P_{error}<0.65$.



Hmatrix1 (3792 x 5056)(4,3)

Drops to 0 at $0.15<P_{error}<0.86$.

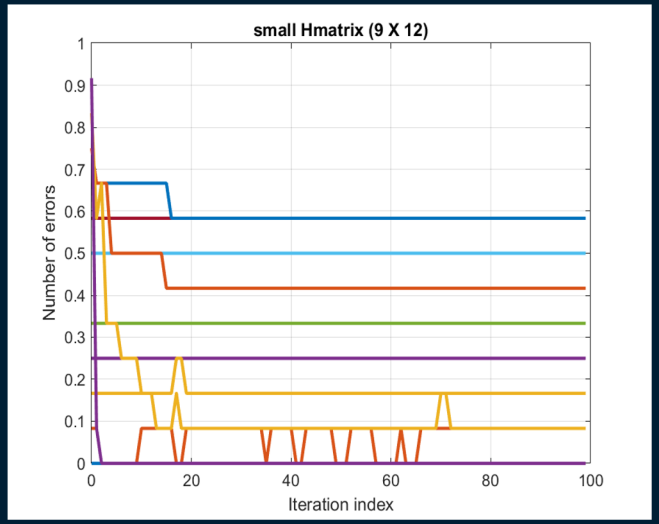


Hmatrix2 (3000 x 5000)(5,3)

Drops to 0 at $0.11<P_{error}<0.89$.

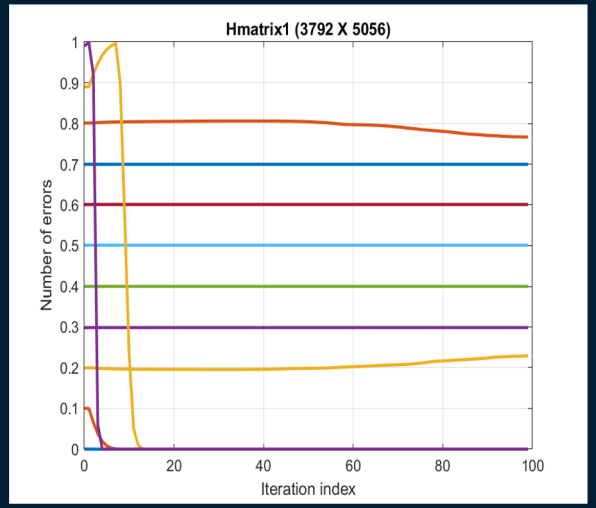
BSC Soft Decision Decoding

Algorithm Convergence Graphs



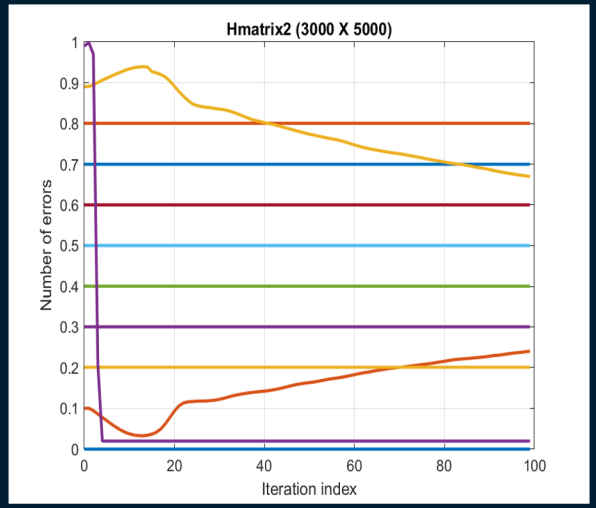
Small H matrix (9 x 12)(3,4)

Drops to 0 at $0.3<P_{error}<0.65$.



Hmatrix1 (3792 x 5056)(4,3)

Drops to 0 at $0.15<P_{error}<0.86$.

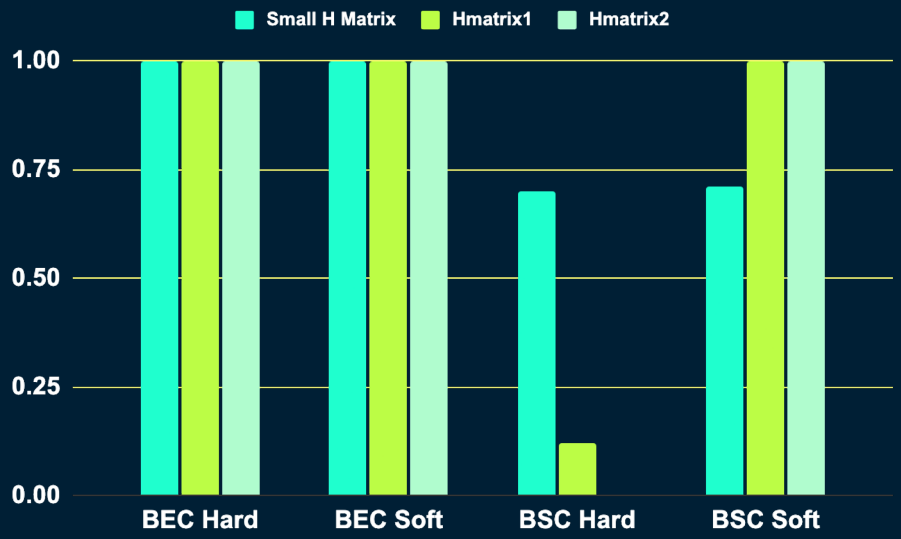


Hmatrix2 (3000 x 5000)(5,3)

Drops to 0 at $0.11<P_{error}<0.89$.

Comparison Of decoders

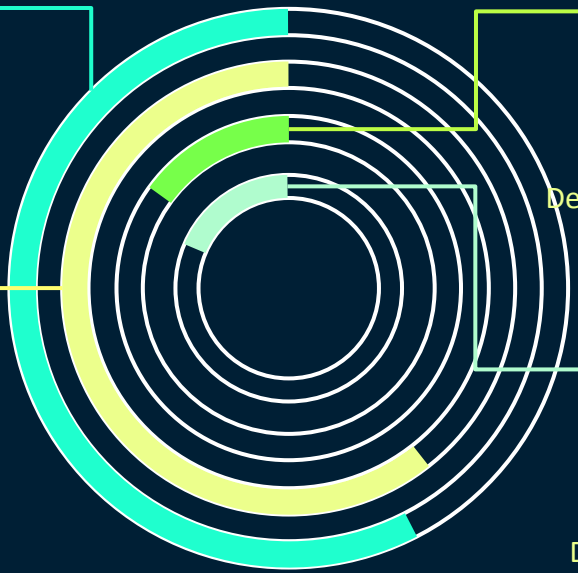
Compared at error probability = 0.1



As we can see BSC decoders are not efficient to use compared to the BEC decoders as BEC decoders successfully decodes till P_{error} =0.63.

Summary

Efficiency of various decoders



60%

BEC Hard

Decodes successfully till 0.6

9%

BSC Hard

Decodes successfully till 0.9

63%

BEC Soft

Decodes successfully till 0.63

14%

BSC Soft

Decodes successfully till 0.14 and 0.88-1

For Hmatrix1 (3792 x 5056)

Brief Discussion on Every Algorithm

- Our BEC algorithms were quite efficient as they were able to decode for error probabilities as high as 0.67.
- While BSC algorithms were not that efficient. So it's practical to use BEC algorithms in Real world applications.
- In BSC Hard decision decoding, Hmatrix1 and small H matrix will not be decoded at error probability = 1.
  (Reason: they have even degree of check Node)

Difficulty Level for implementing various decoders

BSC Hard < BEC Hard < BSC Soft < BEC Soft

References

01

Tutorial paper by Bernard Leiner

02

Gallager’s original LDPC Paper

03

GeeksforGeeks

Appendix

Why does the graph of Hmatrix1 and small H matrix does not go upwards at higher error probability = 1?

- Because degree of check node is even in Hmatrix1 and small H matrix. That's why the parity check equations will give value 1.
- Because the Variable Node receives 1, it's not said to be decoded.

What could have been changed in the soft decision decoding algorithms?

- We could have implement log likelihood ratio based algorithm instead of simple likelihood ratios because addition operations takes less time to be done compared to multiplication operations.
- Though we did not notice any slowing in the algorithm of the tutorial paper by Bernard too.
- Because of C++ our Multiplication operations were faster too.

Why did not we faced the issue of having 0 in the denominator in soft decision decoding schemes?

```
for (int i = 0; i < m; i++)
{
  double *temp = new double[(int)wc];
  for (int j = 0; j < wc; j++)
  {
    double x1 = 1;
    double x0 = 1;
    for (int k = 0; k < wc; k++)
    {
      if (k == j)
      {
        continue;
      }
      x1 *= h[var[k][i]][i];
      x0 *= 1 - h[var[k][i]][i];
    }
    x0 *= 1 - p[i];
    x1 *= p[i];
    temp[j] = x1 / (x1 + x0);
  }
  for (int j = 0; j < wc; j++)
  {
    h[var[j][i]][i] = temp[j];
  }
}
```

- At a time we do not divide 1's and 0's belief. We do multiply them.
- Then we find the constant and then we divide so we will never get 0 in denominator.
- As belief of 0+belief of 1 can not be 0 at a time.

Thank You

Guided By:

Prof. Yash Vasavada
Teaching Assistants:
Ritik Malaviya sir and Vivek Aswani sir

Project is also Available at...