1. In the basic\_function.cc

Follow function was added.

void create\_trellis(const int a, const int b, const int memory\_order, Node state\_diagram[], const int nb\_states){

bool output\_temp[b];

bool temp;

bool init\_data[(memory\_order + 1) \* a \* b];

int n = 1;

//Generate the inner trellis code(This must not the destroy the Property of the Marcov chain)

double rate\_code = (double)a / (double)b;

int max = (memory\_order + 1) \* a \* b;

int first\_num = round((double)max \* rate\_code);

for(int i = 0; i < first\_num; i++){

init\_data[i] = 1;

}

for(int j = first\_num; j < max; j++){

init\_data[j] = 0;

}

gsl\_matrix\_int \* matrix\_trellis = gsl\_matrix\_int\_alloc(max + 1, max + 1);

for(int n = 0; n < max + 1; n++){

for(int m = 0; m < max + 1; m ++){

gsl\_matrix\_int\_set(matrix\_trellis, n, m, init\_data[m]);

}

}

gsl\_matrix\_int\_transpose(matrix\_trellis);

for(int i = 0; i < max; i++){

init\_data[i] = gsl\_matrix\_int\_get(matrix\_trellis, 1, i);

}

int input\_plus\_state; // concatenation of the input its and the state of the registers

if(1 << (a \* memory\_order) != nb\_states)

{

cout << endl

<< "ERROR: The memory order and the number of states in the state "

<< "diagram do not match." << endl;

exit(1);

}

// loop on all the possible states of the registers

for(int cur\_state = 0; cur\_state < nb\_states; cur\_state++)

{

// Loop on all the possible transtions from the current state

for(int cur\_tran = 0; cur\_tran < (1 << a); cur\_tran++)

{

// next state of the transition

state\_diagram[cur\_state].transitions[cur\_tran] = (cur\_state >> a) | (cur\_tran << (a \* (memory\_order - 1)));

input\_plus\_state = cur\_state | cur\_tran << (a \* memory\_order);

// Initianlize temp outputs

for (int i = 0; i < b; i++) output\_temp[i] = 0;

for(int i = 0; i < memory\_order + 1; i++)

for(int j = 0; j < a; j++)

{

temp = (input\_plus\_state) & (1 << ((a \* (memory\_order + 1)) - 1 - i \* a - j));

for (int k = 0; k < b; k++)

{

output\_temp[k] = output\_temp[k] ^ (temp & init\_data[(j \* b + k) \* (memory\_order + 1) + i]);

}

}

for(int i = 0; i < b; i ++){

state\_diagram[cur\_state].bits[cur\_tran][i] = output\_temp[i];

}

} // Loop on all the possible transtions from the current state

}

return;

}

1. In the rect\_code function of the basic\_functions.cc

//Inner trellis codes

else if(option == 3){

// Generate the first column (code bits and transitions) of the trellis

create\_trellis(a, b, memory\_order, state\_diagram, vpc);

// Change the transitions of the first column so that they point to the second column

for(int v = 0; v < vpc; v++)

for(int i = 0; i < k; i++)

state\_diagram[v].transitions[i] += vpc;

// Copy the first column (bits and transitions) into the other columns

for(int c = 1 ; c < input\_length\_symb; c++)

for(int v = 0; v < vpc; v++)

for(int i = 0; i < k; i++)

{

state\_diagram[c \* vpc + v].transitions[i] = (state\_diagram[v].transitions[i] + c \* vpc) % nb\_states;

for(int j = 0; j < b; j++){

state\_diagram[c \* vpc + v].bits[i][j] = state\_diagram[v].bits[i][j];

}

}

}

//Outter ldpc codes

else if(option == 4){

// File containing the trellis encoder

// Its name has the form r\_k\_vpc\_$(input\_length\_symb).txt

stringstream sstemp;

sstemp << "r\_" << k << "\_" << vpc << "\_" << input\_length\_symb << ".txt";

string fname = sstemp.str();

ifstream tempf(fname.c\_str(), ios::in);

// If the graph does not exist, it is created.

if (!tempf) gen\_rect(k, vpc, input\_length\_symb);

else tempf.close();

// File containing the rectangular graph

ifstream graph\_file(fname.c\_str(), ios::in);

if(!graph\_file)

{

cout << "The file containing the state diagram was not found." << endl;

exit(1);

}

//Generate the outter ldpc code file(FileName : outter\_ldpc\_code.txt)

stringstream gen\_outter\_ldpc;

gen\_outter\_ldpc << "./gen\_outter\_ldpc " << nb\_states \* k << " " << b << " " << gsl\_rng\_default\_seed;

cout << gen\_outter\_ldpc.str().c\_str() << endl;

// Execute the make-ldpc for the generate the outter ldpc code.

system((const char\*)gen\_outter\_ldpc.str().c\_str());

// This is the outter ldpc code file name.

ifstream outter\_ldpc\_file("./outter\_ldpc\_code.txt", ios::in);

if(!outter\_ldpc\_file)

{

cout << "The file containing the outter ldpc bits was not found." << endl;

exit(1);

}

// Read the graph info at the beginning of the file.

// It allows the buffer to go directly to the graph itself

int temp;

graph\_file >> temp;

graph\_file >> temp;

graph\_file >> temp;

// Get structure of the encoder

for(int i = 0; i < nb\_states; i++)

{

for(int j = 0; j < k; j++)

{

graph\_file >> temp;

graph\_file >> state\_diagram[i].transitions[j];

}

}

graph\_file.close();

for(int c = 0; c < nb\_states; c++)

for(int i = 0; i < k; i++)

for(int j = 0; j < b; j++)

outter\_ldpc\_file >> state\_diagram[c].bits[i][j];

outter\_ldpc\_file.close();

}

3. In the main function of the perf\_rect\_code.cc

Follow part was added.

// Rectangle graph with inner trellis codes

else if(option == 3){

if(argc != 7){

cout << "Performance of inner trellis codes over the NSC" << endl

<< "The input arguments are:" << endl

<< "1 : Option 3" << endl

<< "2 : Number of outgoing edges per vertex" << endl

<< "3 : Number of vertices per column" << endl

<< "4 : Number of nonbinary input symbols" << endl

<< "5 : b (encoder of rate a:b (IN BITS))" << endl

<< "6 : Output file name" << endl

<< endl;

exit(1);

}

// Input the initial parmeters.

k = atoi(argv[2]);

vpc = atoi(argv[3]);

input\_length\_symb = atoi(argv[4]);

b = atoi(argv[5]);

output\_file\_name = argv[6];

}

// Rectangle graph with outter ldpc codes

else if(option == 4){

if(argc != 7){

cout << "Performance of outter ldpc codes over the NSC" << endl

<< "The input arguments are:" << endl

<< "1 : Option 4" << endl

<< "2 : Number of outgoing edges per vertex" << endl

<< "3 : Number of vertices per column" << endl

<< "4 : Number of nonbinary input symbols" << endl

<< "5 : b (encoder of rate a:b (IN BITS))" << endl

<< "6 : Output file name" << endl

<< endl;

exit(1);

}

//Input the initiaj parameters.

k = atoi(argv[2]);

vpc = atoi(argv[3]);

input\_length\_symb = atoi(argv[4]);

b = atoi(argv[5]);

output\_file\_name = argv[6];

}