Simulation of Turbo Codes

Kwame Ackah Bohulu

27-04-2017

1 Introduction

2 Block Interleavers and Linear Interleavers

A block interleaver is a deterministic interleaver definded by a rectangular matrix of size $N = m \times n$. The input bits are written row-wise and read column-wise. The index mapping function of a block interleaver is defined in (1) below[00868474]

$$\Pi_{\mathfrak{B}_N} \equiv ni + \lfloor 1/m \rfloor \mod N, 0 \leq i < N.$$

The index mapping function of a linear interleaver is defined by (2) below

$$\Pi_{\mathfrak{L}_N} \equiv ki + v \mod N, 0 \leq i < N.$$

where k is a fixed integer which is relatively prime to N and v is a fixed integer.

It is noted that the linear interleaver "linearizes" the $\lfloor \ \rfloor$ portion of (1). Block and Linear interleavers are both based on linear congruence and therefore essentially have the same properties. For example in block interleavers,two elements seperated by a constant $t \mod N$ in an input sequence almost always map to an output sequence with corresponding elements seperated by a fixed constant $tn \mod n$ whiles for linear interleavers, they always map to output sequences with corresponding elements separated by a fixed constant $tk \mod N$. The constant k is called the angular coefficient of the linear interleaver[00868474] and for the block interleaver this is equal to the interleaver depth. Due to the similarity in properties the analysis of the block interleaver can be done using the linear interleaver model.

The performance of the interleaver is affected by its angular coefficient (interleaver depth) as is established in [11]. When used in Turbo codes, linear interleavers are prone to "bad" input sequences of weight 2 and weight 4, especially those of input weight 2. A "bad input sequence" is one that outputs a low weight codeword in Turbo codes. In the case of "bad" weight 4 input sequences, changing the angular coefficient has no effect on improving the performance of the turbo code. However, In the case of "bad" weight 2 input sequences, changing the angular coefficient using the specified technique below helps to avoid such bad input sequences.

To guarantee the minimum value of the sum of the distances between the projection of two lattice points on the x-axis and the distance between the projection of the same points on the y-axis is approximately $\sqrt{2N}$, angular coefficient of $\sqrt{2N}$ or integer value close to it is required.[12] It is however shown that angular coefficient value of $\sqrt{N}$ is also a good choice meaning there is a large minimum distance between the points in the lattice which ensures a large minimum distance for the weight 2 input sequences of Turbo codes when this interleaver is used.

A graph of two linear interleavers are shown in Figs. 1 and 2 below

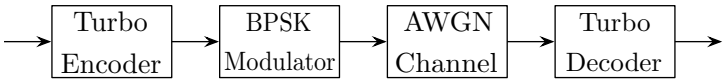

Figure 1: システムモデル

2.1 ターボ符号器

ターボ符号器、二つの畳み込み符号 (要素符号器) をインタリーバで並列連結することで作られている。入力はバイナリの情報系列であり、出力はターボ復号である。ターボ符号器の図は、図 2 に書かれている。要素符号器は、同じ

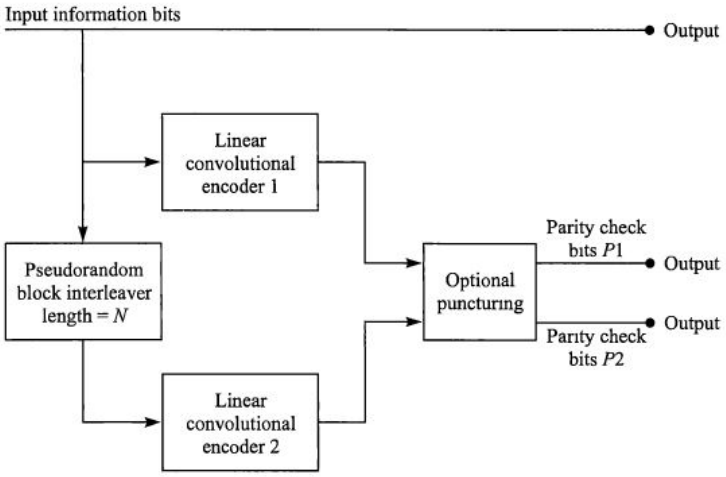

Figure 2: ターボ符号器

で、5/7, k=1,n=2,K=3 RSC 符号器である。利用したインタリーバは置換多項式に基づいてインタリーバ [5] とブロックインタリーバである。インタリーバの長さは、情報系列の長さと同じである。パンクチュアリングはしない

ので、ターボ符号器のレートは 1/3 である。ターボ符号器のプログラムは図 3 に書かれている。

```
[IP,cs,ns,op]=RSCencoder2(K,k,n,gen_func);
rng('default');
 msg=randi([0,1],1,interleaver_size-(K-1));

%encoder 1
[codeword1,
code1,code_parity1,padding1]=ConvoCodeGenerator2(msg,cs,ns,op,IP
,gen_func);
systematic=codeword1(:,1)';

%encoder2
%a. interleave
interleaved_msg=systematic(yy);
%interleaved_msg(end-(K)+1:end)=[];
[codeword2, code2,code_parity2,padding2]
=ConvoCodeGenerator2(interleaved_msg,cs,ns,op,IP,gen_func);

 if length(code_parity2)>interleaver_size
     code_parity2(interleaver_size+1:end)=[];
elseif length(code_parity2)<interleaver_size
code_parity2(end+1:interleaver_size)=zeros(1,interleaver_size-
length(code_parity2));
 end
code=[systematic;code_parity1;code_parity2];
fin_code=code(:)';
```

Figure 3: ターボ符号器プログラム

2.2 BPSK 変調器と AWGN チャネル

ターボ復号器の出力は、BPSK 変調器の入力になる。入力ビットが 0 の場合、出力が −1 になり、入力ビットが 1 の場合、出力が −1 になる。BPSK 変調器の出力は、AWGN チャネルで送信する。BPSK 変調器と AWGN チャネルのプログラムは図 4 書かれている。

2.3 ターボ復号器

ターボ復号器の図は図 5 に書かれている。SISO の復号器が利用されるので、BPSK 復調はしない。ターボ復号器の入力は、AWGN チャネルの出力であり、系統的ビットと二つのパリティービットに分別する。1 番目の SISO 復号器の入力は系統的ビットと 1 番目のパリティービットであり、2 番目の SISO 復号器は、インタリーバで並び替えされた系統的ビットと 2 番目のパリティービットである。両方の SISO 復号器は、Max-Log-MAP アルゴリズムを再帰的に利用して復号する。ターボ復号器の出力は、入力の LLR 値であり、最後の反復になったら、それをデインタリーブして、情報系列の値を評価する。ターボ復号器のプログラムは図 6 に書かれている。

```
%BPSK Modulation
u=find(fin_code==0);
mod_fin_code =fin_code;
mod_fin_code(u)=-1;
%AWGN Channel
y=awgn(mod_fin_code,EbNo,'measured');
```

Figure 4: BPSK 変調器と AWGN チャネルプログラム

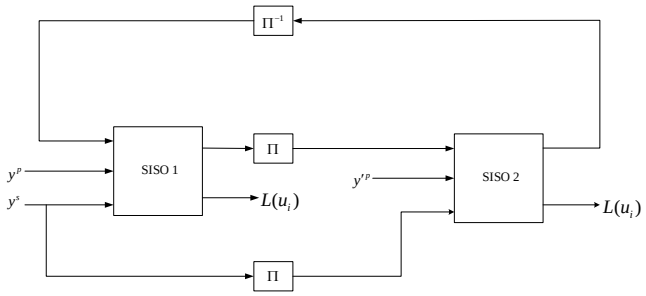

Figure 5: ターボ復号器

3 Max-log-MAP アルゴリズムの実現方法

Max-log-MAP アルゴリズムは、対数領域の BCJR アルゴリズムである。BPSK 変調と AWGN チャネルの場合は、式 1-3 を使用して復号する。

$$\widetilde{\gamma}(s',s) = \ln P(u_i) + (\frac{2y_i^s c_i^s}{N_o}) + (\frac{2y_i^p c_i^p}{N_o}) \tag{1}$$

$$\begin{aligned} \widetilde{\alpha}_i(s) &= \max_{s'}\{\widetilde{\alpha}_{i-1}(s') + \widetilde{\gamma}(s',s)\} \\ \widetilde{\beta}_{i-1}(s) &= \max_{s}\{\widetilde{\beta}_i(s') + \widetilde{\gamma}(s',s)\} \end{aligned} \tag{2}$$

```
%Turbo Decoder
%a. decompose y into systematic and parity bits
Y=reshape(y,n+1,length(y)/(n+1));
ys=Y(1,:);%systematic portion
yp=Y(2,:);%parity checkbit 1
ypp=Y(3,:);%parity checkbit 2
yss=ys(yy);
La1=zeros(1,length(yp));%a priori LLR

Y1=[ys;yp];
y1=Y1(:)';
Y2=[yss;ypp];
y2=Y2(:)';

Lc=(4)/(10^(-EbNo(r)*0.1));
LcYs=Lc*ys;
LcYss=Lc*yss;

for w=1:iterations
% %first SISO decoder
[LLR1] = max_log_map(gen_func,EbNo(r),y1,La1);
  Le1=LLR1-La1-LcYs;
 %[LE1;Le1]
La2=LE1(yy);%feed Le from previous decoder as o priori LLR

% %second decoder

[LLR2] = max_log_map(gen_func,EbNo(r),y2,La2);
 Le2=LLR2-LaZ-LcYss;

% %feedback
 La1=LE2(deint);

 LLR=LLR2(deint);
decoded_op=zeros(1,length(LLR));
end
%decode output
plus1=find(LLR>=0);
minus1=find(LLR<0);
decoded_op(plus1)=1;
decoded_op(minus1)=0;
```

Figure 6: ターボ復号器のプログラム

$$
\begin{aligned}
L(u_i) = &\max_{R_1}\left\{\widetilde{\alpha}_{i-1}(s') + \widetilde{\gamma}(s', s) + (\widetilde{\beta}_i(s')\right\} - \\
&\max_{R_0}\left\{\widetilde{\alpha}_{i-1}(s') + \widetilde{\gamma}(s', s) + (\widetilde{\beta}_i(s')\right\}
\end{aligned} \tag{3}
$$

$$
L^{(e)}(u_i) = L(u_i) - \left(L_c y_i^s + L^{(a)}(u_i)\right) \tag{4}
$$

$L_c y_i^s = \frac{4\sqrt{\varepsilon_c} y_i^s}{N_o}$ はチャネルの $L(u_i)$ 値を示し、組織的なビットの影響に関するチャネルの出力を表れる。
$L^{(a)}(u_i) = \ln \frac{P(u_i)=1}{P(u_i)=0}$ は先天的な (a priori)$L(u_i)$ 値である。
$L^{(e)}(u_i)$ は、外因性 $L(u_i)$ の値を示し、パリティビットに関する $L(u_i)$ の値である。

両方の場合は、以下の初期条を利用する。

$$
\widetilde{\alpha}_0(s) = \begin{cases} 0, & s = 0 \\ -\infty, & s \neq 0 \end{cases}
$$

$$
\widetilde{\beta}_N(s) = \begin{cases} 0, & s = 0 \\ -\infty, & s \neq 0 \end{cases}
$$

$\widetilde{\alpha}_i(s)$ は、時間が i-1 のとき、状態が s' で今まで受信された系列は ($\boldsymbol{y}_{<i}$ の共同確率を表す。
$\widetilde{\gamma}(s', s)$ は、過去の状態が s' の条件で未来の状態が s であり、受信された系列

が $\boldsymbol{y}_i$ である確率を表す。
$\widetilde{\beta}_{i-1}(s)$ は、現在の状態が s の条件で未来の系列は $\boldsymbol{y}_{>i}$ になるの条件付確率を表す。

Max-Log-MAP アルゴリズムを実現するため、図 7 に書かれている　5/7, k=1,n=2,K=3 RSC 符号器 trellis を使って説明する。

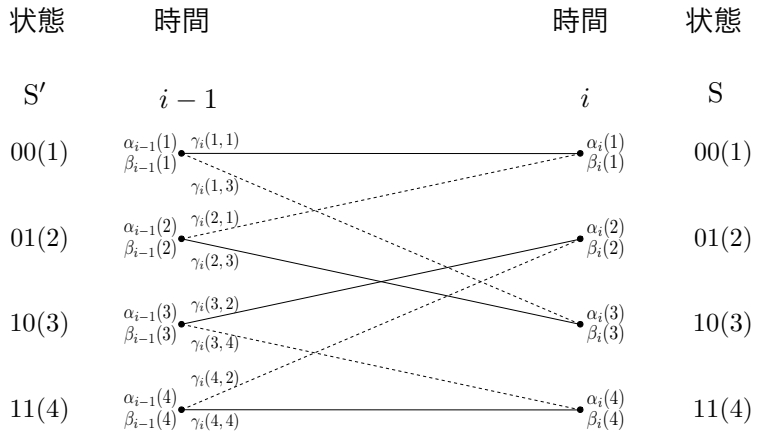

Figure 7: 5/7, k=1,n=2,K=3 RSC 符号器の trellis

3.1 $\widetilde{\gamma}(s',s)$ の計算方法

$\widetilde{\gamma}(s',s)$ を計算するために、trellis の状態遷移を保存する必要があります。$\widetilde{\alpha}_i(s)$ と $\widetilde{\beta}_{i-1}(s)$ の計算にも必要である。$\widetilde{\gamma}(s',s)$ のプログラムは図 8 に書かれている。

3.2 $\widetilde{\alpha}_i(s)$ と $\widetilde{\beta}_{i-1}(s)$ の計算方法

式 2 で $\widetilde{\alpha}_i(s)$ と $\widetilde{\beta}_{i-1}(s)$ の計算ができる。計算するのに、$\widetilde{\gamma}(s',s)$ の値が必要である。例えば、保存された trellis の状態遷移は表 1 に書かれている。

左	右
1	1
1	3
2	3
2	1
3	4
3	2
4	2
4	4

$\widetilde{\alpha}_i(s)$ を計算する場合、右側の値は、次の $\widetilde{\alpha}_i(s)$ を計算すのに、どちらの $\widetilde{\gamma}(s',s)$ を使用すればいいのかを示す。左側の値は、どちらの前の $\widetilde{\alpha}_i(s)$ の値

```
CS=binaryVectorToDecimal(cs)'+1;
NS=binaryVectorToDecimal(ns)'+1;
CS_NS=[CS:NS] ;
trans_states =repmat(CS_NS(:)',depth,1);
trans_states_alpha=repmat(1:2^(K-1),depth+1,1);
trans_states_beta=trans_states_alpha;


while ii<length(y1)
      YY=y1(1+ii:ii+divisor);
     c_it=op;
     qq=find(c_it==0);
     c_it(qq)=-1;

    uvw_yy=repmat(YY,length(c_it),1);
  uvw=(2*uvw_yy.*c_it)';
     Ip=(IP)';
     ab=find(Ip==0);
     bc=find(Ip==1);
prob=1./(1+exp(La));%P(Ui=0)
ff=zeros(1,length(Ip));
     ff(ab)=prob(iii);
     ff(bc)=1-prob(iii);
     vx=(sum(uvw)./(10^(-EbNo/10)));
     gammas(iii,1:length(uvw))=log(ff)+(vx);

    x_g(iii,1:length(uvw))=(uvw(2:end,:))./(10^(-EbNo/10));

    ii=ii+divisor;
    iii=iii+1;

end
```

Figure 8: $\widetilde{\gamma}(s', s)$ のプログラム

を使用すればいいのかを示す。

$\widetilde{\beta}_{i-1}(s)$ を計算する場合、左側の値は、次の $\widetilde{\beta}_{i-1}(s)$ を計算すのに、どちらの $\widetilde{\gamma}(s', s)$ を使用すればいいのかを示す。右側の値は、どちらの前の $\widetilde{\beta}_{i-1}(s)$ の値を使用すればいいのかを示す。図 9 の考え方で、$\widetilde{\alpha}_i(s)$ と $\widetilde{\beta}_{i-1}(s)$ の計算が簡単にできる。

3.3 $L^{(e)}(u_i)$ と $L(u_i)$ の計算方法

$L(u_i)$ は、式 3 で計算される。最初に時間が i のときの $\widetilde{\gamma}(s', s)$ の値は、情報ビット 0 に関する値 (分母) と情報ビット 1 に関する値 (分子) に分ける。次にそれぞれの $\widetilde{\gamma}(s', s)$ どちらの $\widetilde{\alpha}_i(s)$ と $\widetilde{\beta}_{i-1}(s)$ と足し算すればいいのかを決める。次に、分子の和から分母の和を引く。最後に、式 4 を使って、$L^{(e)}(u_i)$ を計算する。$L^{(e)}(u_i)$ と $L(u_i)$ のプログラムは図 10 に書かれている。注意**:** 2 番目の SISO 復号器の $L^{(a)}(u_i)$ はインタリーブされた $L^{(e)}(u_i)$ になるので、それを使って、$\widetilde{\gamma}(s', s)$、$\widetilde{\alpha}_i(s)$ と $\widetilde{\beta}_{i-1}(s)$ をまた計算しなくてはいけない。

```
%-----------------calculate alpha--------------------
Ts=trans_states(1,:);
tS=vec2mat(Ts,2);
 odd_tS=tS(:,1);
kk=1;
while kk<size(alpha,1)
AA=alpha(kk,:);
GG=gammas(kk,:);

     aalpha=[];
    for i=1:2^(K-1)
        sum_pos=find(tS(:,2)==(i));
        alpha_to_use=odd_tS(sum_pos);
        for aa=1:length(alpha_to_use)
            bb(aa)=find(trans_states_alpha(kk,:)==alpha_to_use(aa));
        end
       temp_alpha= AA(bb)+GG(sum_pos);
        aalpha(i)=max(temp_alpha);

    end
alpha(kk+1,:)=aalpha;
kk=kk+1;
end


%-----------------calculate beta--------------------

kk=depth;
Ts=trans_states(kk,1:length(trans_states(kk,:)));
tS=vec2mat(Ts,2);

 even_tS=tS(:,2);
while kk>0
BB=beta(kk+1,:);
GG=gammas(kk,:);

     bbetaa=[];
    for i=1:2^(K-1)
        sum_pos=find(tS(:,1)==(i));
        beta_to_use=even_tS(sum_pos);
        for aa=1:length(beta_to_use)
            bb(aa)=find(trans_states_alpha(kk,:)==beta_to_use(aa));
        end
       temp_beta= BB(bb)+GG(sum_pos);
        bbeta(i)=max(temp_beta);

    end
beta(kk,:)=bbeta;
```

Figure 9: $\widetilde{\alpha}_i(s)$ と $\widetilde{\beta}_{i-1}(s)$ のプログラム

4 References

[**1**] John G. Proakis, Masoud Salehi. ”Digital Communications”, Fifth Edition,Chapter 8, McGraw-Hill

.

[**2**] Oscar Y. Takeshita, Member, IEEE, and Daniel J. Costello ,”New Deterministic Interleaver Designs for Turbo Codes”,IEEE Trans. Inform. Theory, vol. 46,pp. 1988-2006,Nov. 2000

[**3**] L. C. Perez, J. Seghers, D. J. Costello, Jr., ”A distance spectrum interpretation of turbo codes”, IEEE Trans. Inform. Theory, vol. 42, pp. 1698-1709, Nov. 1996.

```
 %---------------------LLR-------------------------%

for kk=1:depth


TSA=trans_states_alpha(kk,:);
TSB=trans_states_beta(kk+1,:);

GG=gammas(kk,:);
AA=alpha(kk,:);
BB=beta(kk+1,:);
TS=vec2mat(trans_states(kk,:),2);

ip_z=1:2:length(GG);
ip_o=2:2:length(GG);

top_GG=GG(ip_o);
bot_GG =GG(ip_z);

%top alpha and beta
a_b_top=TS(ip_o,:);

%bottom alpha and beta
a_b_bot =TS(ip_z,:);


top_AA=AA(a_b_top(:,1)) ;
bot_AA=AA(a_b_bot(:,1));
top_BB=BB(a_b_top(:,2));
bot_BB=BB(a_b_bot(:,2));


LLR_top=max(top_AA+top_BB+top_GG);

LLR_bot=max(bot_AA+bot_BB+bot_GG);

LLR(kk)=(LLR_top-LLR_bot);
end
```

Figure 10: $L(u_i)$ ののプログラム

[**4**] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes", Proc. Intern. Conf. Communications (ICC), Geneva, Switzerland, pp. 1064- 1070, May 1993.

[**5**] Jing Sun, Oscar Y. Takeshita "Interleavers for Turbo Codes Using Permutation Polynomials over Integer Rings", IEEE Trans. Inform. Theory, vol. 51, pp. 101 - 119 Jan. 2005