5.4. ALGORITMO DE VITERBI.

Debido a la necesidad que existe en los sistemas de comunicaciones móviles de aprovechar el reducido espectro disponible obliga a dotar a estos sistemas de esquemas de codificación que reduzcan la en cierta manera la tasa de bits en la que es convertida la voz o datos que son necesarios mandar en comunicaciones inalámbricas. Sin embargo, la codificación de voz y datos cumple otra misión. Debido a lo áspero que resulta, como ya hemos podido comprobar, el canal de radio para comunicaciones móviles, es necesario tener la posibilidad de poder recuperar información que en el transcurso de la comunicación hayan podido perderse. Esta técnica de protección se ve cumplimentada junto con otras que hacen al sistema más robusto frente a caídas repentinas de la señal. No es intención en este capitulo el presentar de forma precisa este proceso, sino mostrar como se ha implementado un canal de comunicaciones con una señal GSM, como se ha protegido a esta señal con los mecanismos que especifica el sistema y como empleando la codificador de Viterbi es posible recuperar la señal en aquellos casos en los que la perdida de información no ha sido grave. Ponemos de relieve que el canal de comunicaciones utilizado es, en lugar del modelo de bloques que habría que utilizar en cualquier simulación, un modelo oculto de Markov, el cual presenta una gran diferencia con respecto a la anterior opción desde el aspecto computacional.

5.4.1. Esquema de codificación en el sistema GSM.

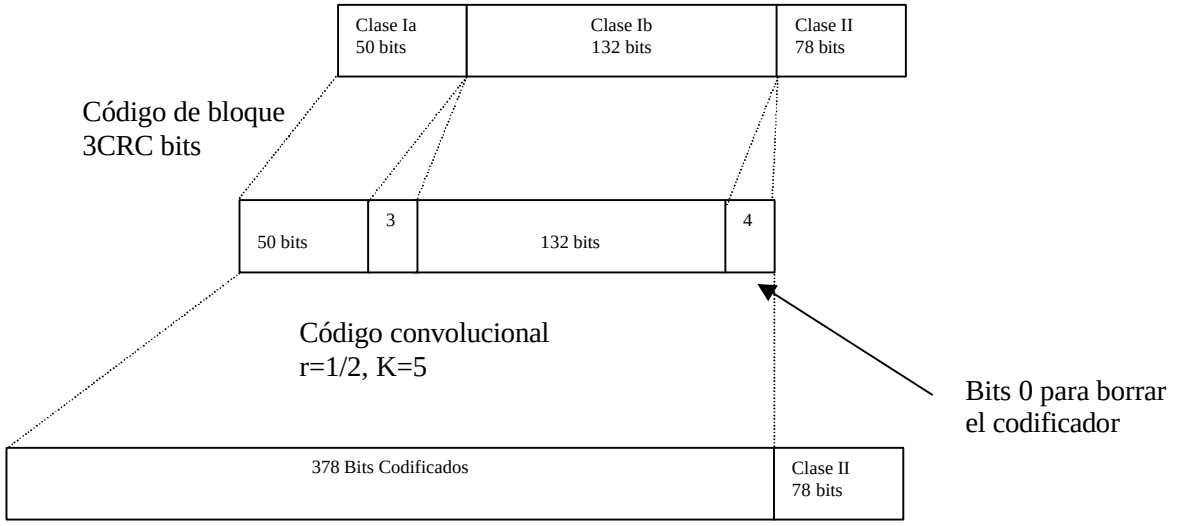
La unidad básica en el sistema de codec de la señal de voz en GSM es un bloque de 20ms que sobre una total de 160 muestras y obtiene una señal a 13kbps obteniendo de esta forma 260bits en un periodo de 20 ms. NOsotros trabajaremos con la señal que deja este codec, señal que se robustece tras el proceso de codificación.

En este codec se produce una concatenación de dos códigos:

- Código tipo bloque, utilizado como detector de errores.
- Código de tipo convolucional, empleado como corrector de errores.

Existen distintos tipos de codificación en función de la información que se este mandando existiendo diferente trato para los datos, voz o señalización. Nosotros veremos la codificación que sufre la voz.

El esquema en el que sintetizamos los procesos de codificación de la voz se muestran en la siguiente gráfica.



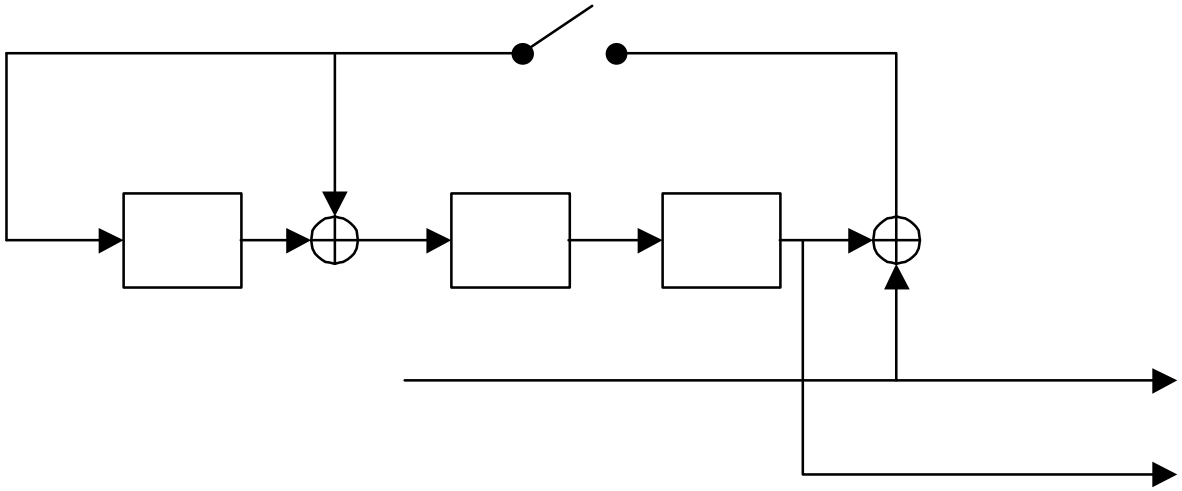
De esta forma conseguimos un total de 456 bits sobre un periodo de 20 ms, lo cual nos da una tasa final de 23.250 kbps.

La codificación esta implementada en la función codif.m

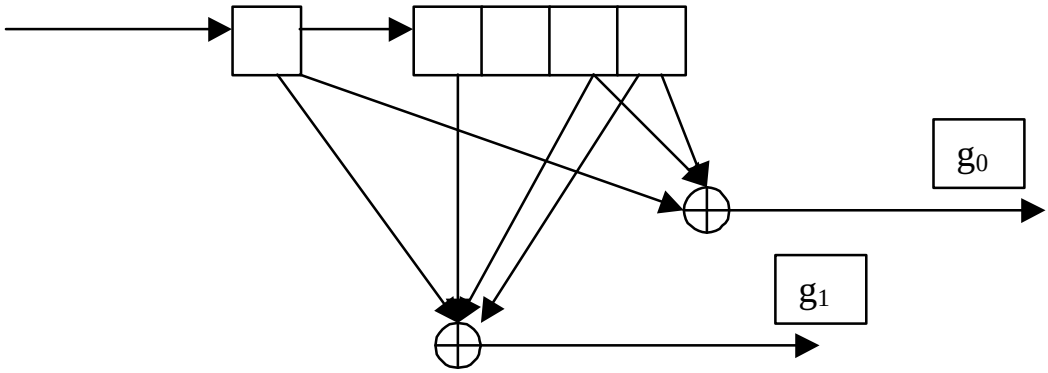
```
function [bitc]=codif(bit)

% Funcion a la que se le pasan 260 bits resultado del codec vocal de GSM
% del tipo RPE-LPT
% ocupando un intervalo de 20ms y las convierte en 456 bits siguiendo las
% normas de codificacion
% para tramas con datos de voz en el estandar GSM.
%
%            [bitc]=codif(bit)
%
% bitc: Secuencia de bits codificados. Tendra una longitud de 456 bits.
% bit: Secuencia de bits que contienen la señal vocal de interes. Son 260
%      bits.


% los 50 primeros bits (clase 1a) se codifican segun bloque (53,3). Es
% del tipo cilico con el generador polinomial
% G4(D)=D^3+D+1
cod1a=zeros(0,0);
reg=zeros(1,3);
for(i=1:50)
   aux=reg;
   reg(1)=xor(aux(3),bit(i));
   reg(2)=xor(reg(1),aux(1));
   reg(3)=aux(2);
end
cod1a=[bit(1:50) fliplr(reg)];
% Los bits codificados de la clase 1a junto con los 132 bits siguientes y
% 4 bits de cola se codifican con un codigo
% convolucional de rendimiento 1/2 y longitud obligada 5. CC(2,1,5).
% Se usan como polinomios generadores g0=1+D^3+D^ y g1=1+D+D^3+D^4.
```

```
b=[cod1a bit(51:182) zeros(1,4)];
l=length(b);
cod1a1b=zeros(0,0);
reg=zeros(1,5);
for(j=1:l)
   for(k=5:-1:2)
      reg(k)=reg(k-1);
   end
   reg(1)=b(l-j+1);
   g0=xor(xor(reg(1),reg(4)),reg(5));
   g1=xor(xor(xor(reg(1),reg(2)),reg(4)),reg(5));
   cod1a1b=[cod1a1b g0 g1];
end
bitc=[cod1a1b bit(183:260)];
return
```

La primera parte de la función esta dedicada a implementar la codificación de bloque esquematizada en la función



Mientras que la segunda parte de la función se encarga de realizar la codificación convolucional mediante la siguiente función.



La información codificada en un periodo de 20 ms es posteriormente entrelazado hasta en 8 ráfagas distintas consiguiendo de esta forma que la perdida de una trama no provoque una perdida que luego el código convolucional no sea capaz de subsanar. El proceso de entrelazado y desentrelazado se realiza en las funciones entrel.m y entreinv.m

```
function [bitent]=entrel(bitcod)
% Funcion en la que se introducen 6 tramas codificadas de 456 bits cada
% una
% y de la que se obtiene una multitrama que dura 120ms y en la cual los
% bits se encuentran
% entrelazados siguiendo las recomendaciones del sistema GSM.
%
%      [bitent]=entrel(bitcod)
%
% bitent: Matriz  donde cada columna sera el bloque adecuado para
% introducir en una rafaga normal.
%         en cada rafaga se cogen dos filas de bitent.
% bitcod: Secuencia de bits codificados.

% Si tenemos 6 tramas A B C D E F la forma del entrelazado sera:

% Campo1:   AAAABBBBCCCCDDDDEEEEFFFF
% Campo2:   FFFFAAAABBBBCCCCDDDDEEEE

% Esto es lo mismo que decir AF-AF-AF-AF-BA-BA-BA-BA-CB-CB-CB-CB-DC-DC-
% DC-DC-ED-ED-ED-ED-FE-FE-FE-FE

% donde el bloque de A,B,...,F se obtiene de sacar filas de una matriz
% donde la trama ha sido ordenada por columnas.
A=zeros(0,0,0);

bitent=zeros(57,48);          % los bits de señalizacion se añaden en la
                              % funcion que genera la rafaga normal.
for(k=1:6)
   Aux=zeros(0,0);
   for(i=1:57)
      Aux=[Aux bitcod((i-1)*8+1+(k-1)*456:i*8+(k-1)*456)'];
   end
   A(:,:,k)=Aux;
end
j=1;
k=1;
for(i=1:2:48)
   a=rem(k,6);
   b=rem(k+5,6);
   if(a==0)
      a=6;
   end
   if(b==0)
      b=6;
   end
   bitent(:,i)=A(j,:,a)';
   bitent(:,i+1)=A(j,:,b)';
   if(j==4)
      k=k+1;
```

```
      j=j+1;
   elseif(j==8)
      j=1;
      k=k+1;
   else
      j=j+1;
   end
end
return


function [bitcod]=entreinv(bitent)
% Funcion en la que se introducen una matriz cuyas filas son bloques
% entrelazados
% siguiendo las recomendaciones del sistema GSM. Estos bloques pertenecen
% a 6 tramas codificadas de 456 bits cada una
% duran 120ms. De esta funcion se obtiene las tramas una vez eliminado el
% entrelazado.
%
%      [bitcod]=entreinv(bitent)
%
% bitcod: Secuencia de bits codificados.
% bitent: Matriz  donde cada columna sera el bloque adecuado para
% introducir en una rafaga normal.
%         en cada rafaga se cogen dos columnas de bitent.


% Si tenemos 6 tramas A B C D E F la forma del entrelazado sera:
% Campo1:   AAAABBBBCCCCDDDDEEEEFFFF
% Campo2:   FFFFAAAABBBBCCCCDDDDEEEE

% Esto es lo mismo que decir AF-AF-AF-AF-BA-BA-BA-BA-CB-CB-CB-CB-DC-DC-
% DC-DC-ED-ED-ED-ED-FE-FE-FE-FE

% donde el bloque de A,B,...,F se obtiene de sacar filas de una matriz
% donde la trama ha sido ordenada por columnas.

j=1;
k=1;
A=zeros(8,57,6);
for(i=1:2:48)
   a=rem(k,6);
   b=rem(k+5,6);
   if(a==0)
      a=6;
   end
   if(b==0)
      b=6;
   end
   A(j,:,a)=bitent(:,i)';
   A(j,:,b)=bitent(:,i+1)';
   if(j==4)
      k=k+1;
      j=j+1;
   elseif(j==8)
      j=1;
```

```
      k=k+1;
   else
      j=j+1;
   end
end
bitcod=zeros(0,0);
for(k=1:6)
   for(i=1:57)
      bitcod=[bitcod  A(:,i,k)'];
   end
end
return
```

Para trasladar la información se utilizan ráfagas normales donde la información codificada y entrelazada se encuentra al principio y final de la ráfaga. Las funciones que implementan este hecho son rafnormal.m y rafinv.m

```
function [bitnormal]=rafnormal(bitent)
% Funcion que genera a partir de las tramas entrelezadas que forman una
% multitrama las
% correspondientes rafagas normales con los correspondientes bits de
% señalizacion que estaran
% puestos a uno mientras que la secuencia de entrenamiento sera
% aleatoria. Ademoas la rafagas de bits tienen
% los bits de cola (Tail Bits)
%
%            bitnormal=rafnormal(bitent)
%
% bitnormal: Secuencia de bits de salida y que forman un conjunto de
%           rafagas normales.
% bitent: Matriz  donde cada columna sera el bloque adecuado para
%         introducir en una rafaga normal.
%         en cada rafaga se cogen dos columnas de bitent.


l=size(bitent);
len=l(2);
bitnormal=zeros(0,0);
for(i=1:2:len)
   rand('seed',i*2);
   bitnormal=[bitnormal zeros(1,3) bitent(:,i)' zeros(1,1)
round(rand(1,26)) bitent(:,i+1)' zeros(1,1) zeros(1,3)];
end
return



function [bitent]=rafinv(sal)
% Funcion que recibe rafagas normales y saca de ellas los datos de voz
% que estan entrelazada.
%
%            bitent=rafinv(sal)
%
% bitent: Matriz  donde cada fila sera el bloque adecuado para introducir
% en una rafaga normal.
%         en cada rafaga se cogen dos filas de bitent.
```

```
% sal: Secuencia de bits de salida y que forman un conjunto de rafagas
%      normales.

l=length(sal);
bitent=zeros(57,2*l/148);
j=1;
for(i=1:148:l)
   bitent(:,j)=sal(i+3:i+59)';
   bitent(:,j+1)=sal(i+87:i+143)';
   j=j+2;
end
return
```

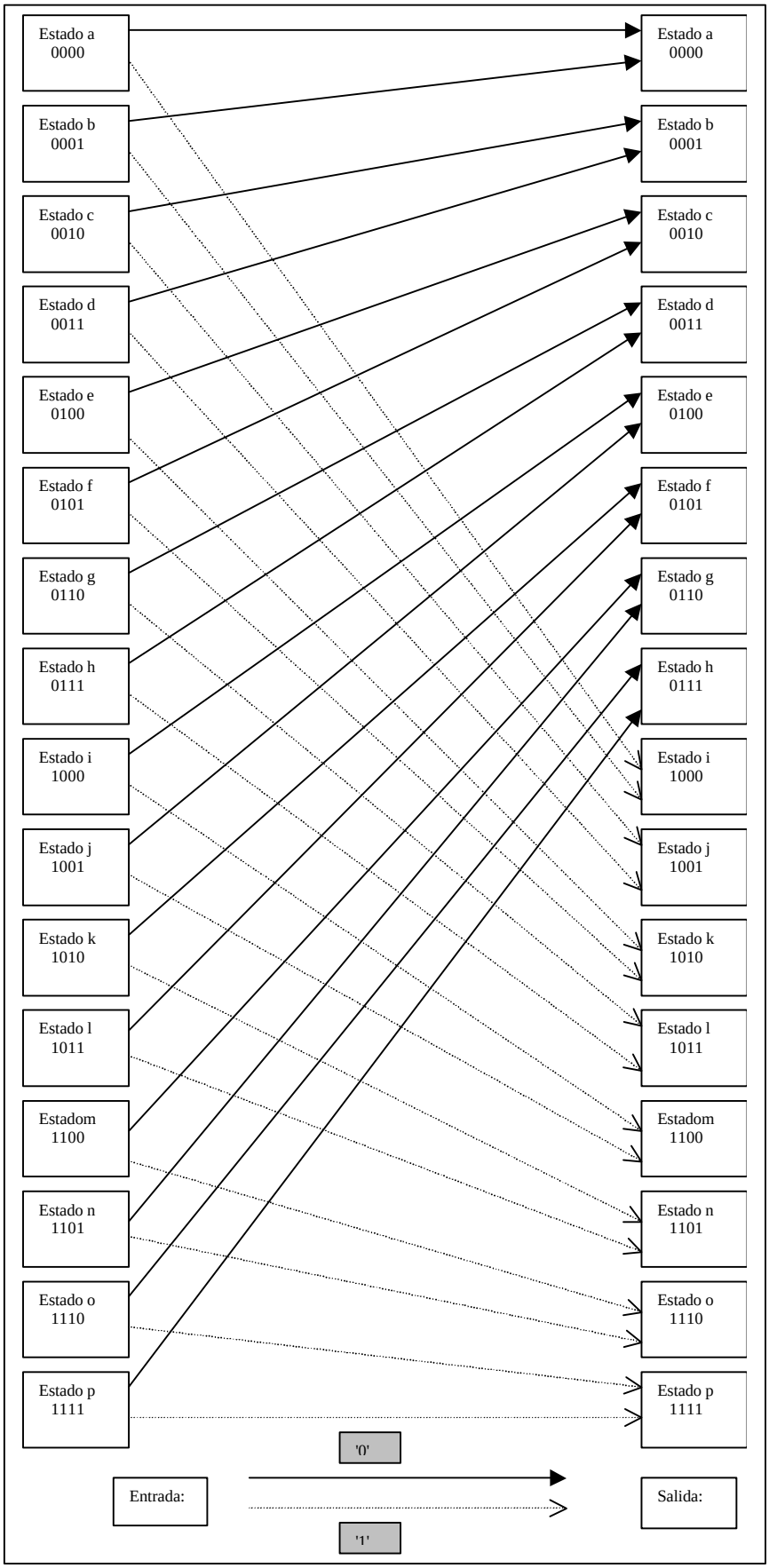
Una vez hemos compuesto la señal la hacemos pasar por el canal modelado con el modelo oculto de Markov mediante la función prueba.m que es la que arranca toda la simulación presentada en este capitulo.

```
function [ber,berc]=prueba(nmf,X,Y)

% Funcion que realiza la simulacion del sistema definido por una cadena
% de Markov
% utilizando para esto la estructura de codificacion del sistema GSM.
%
%          function [ber,berc]=prueba(nmf)
%
% ber: Probabilidad de error de bit de los datos generados recuperados
%      por el decodificador.
% berc: Probabilidad de error de bit de los datos a la salida del modelo
%      del canal.
% nmf: Numero de multitrama que queremos que tenga la simulacion. Las
%      tramas seran insertadas en rafagas normales
%      las cuales se daran de forma consecutiva. En las rafagas normales
%      no utilizaremos la secuencia de entrenamiento
%      ni los bits de señalizacion adjunto a cada bloque insertado en la
%      rafaga normal.


% multitrama= 6 tramas de 120ms.

sem=1;
biti=zeros(0,0);
bitf=zeros(0,0);
A=X;
B=Y;
berc=0;
for(i=1:nmf)
   bitcod=zeros(0,0);
   for(j=1:6)
      rand('seed',sem*i);
      v=round(rand(1,260));
      biti=[biti v];
      bitc=codif(v);
      bitcod=[bitcod bitc];
```

```
   end
   len=length(bitc);
   bitent=entrel(bitcod);            %Entrelaza los bits
                                     %correspondientes a una multitrama.


   bitnormal=rafnormal(bitent);     %Crea rafagas normales a partir de la
                                    %secuencia de bits entrelazados.


   [sal,epattern,b]=sdiscr(A,B,bitnormal',sum(100*clock));
   epattern
   berc=berc+b/nmf;

   bitent=rafinv(sal');
   bitcod=entreinv(bitent);
   for(k=1:6)

      f=decodif(bitcod((k-1)*len+1:k*len));
      bitf=[bitf f];
   end
end
ber= sum(xor(biti,bitf))/length(biti);

return
```

5.4.2. DECODIFICACIÓN CON ALGORITMO DE VITERBI DE CÓDIGO CONVOLUCIONAL EN GSM.

En la decodificación de códigos convolucionales, así como en los de bloque utilizados en GSM, es posible la utilización de diagramas de trellis. Para que esto sea viable, debemos tener la función que se encarga de generar los datos codificados a partir de la secuencia de información. En GSM el código convolucional utilizado es el CC(2,1,5), es decir un código de rendimiento 1/2 y longitud obligada 5. De esta forma conseguimos obtener 389 bits de un conjunto original de 189 bits, tal como se muestra en la figura anterior sobre la codificación de datos en GSM. De esta forma y con la función de generación podemos construir el diagrama de trellis correspondiente.


Estado a
0000
Estado b
0001
Estado c
0010
Estado d
0011
Estado e
0100
Estado f
0101
Estado g
0110
Estado h
0111
Estado i
1000
Estado j
1001
Estado k
1010
Estado l
1011
Estadom
1100
Estado n
1101
Estado o
1110
Estado p
1111
Estado a
0000
Estado b
0001
Estado c
0010
Estado d
0011
Estado e
0100
Estado f
0101
Estado g
0110
Estado h
0111
Estado i
1000
Estado j
1001
Estado k
1010
Estado l
1011
Estadom
1100
Estado n
1101
Estado o
1110
Estado p
1111
'0'
Entrada:
Salida:
'1'

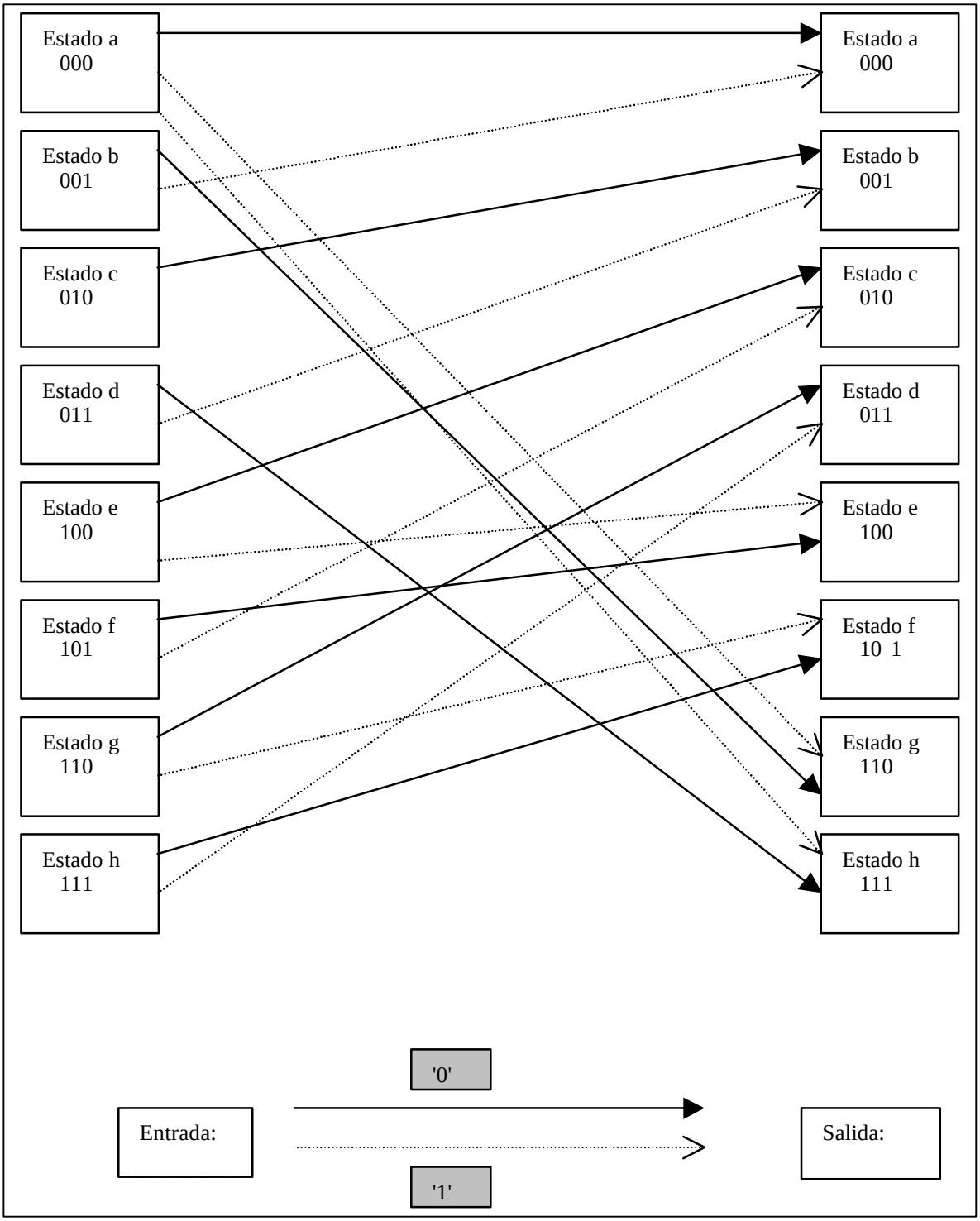
Este diagrama de trellis es aplicado en la función viterb.m.

```
function [deco]=viterb(codi)

% Funcion que aplica el algoritmo de Viterbi a la secuecnia codificada de
% forma
% convolucional utilizando CC(2,1,5) determinado por el estandar para el
% sistema GSM.
%
%         function [deco]=viterb(codi)
%
% deco: Secuencia decodificada cuya longitud es de 189 que representan el
% grupo1.
% codi: Secuencia codificada cuya longitud es de 378.


 M(1,:)=[ 0 0 1 1 1 9 ];          % M(1,:)=[ 0 0  |1|  1 1  |9| ]
 M(2,:)=[ 1 1 1 0 0 9 ];
 M(3,:)=[ 1 1 2 0 0 10 ];
 M(4,:)=[ 0 0 2 1 1 10 ];
 M(5,:)=[ 0 0 3 1 1 11 ];
 M(6,:)=[ 1 1 3 0 0 11 ];
 M(7,:)=[ 1 1 4 0 0 12 ];
 M(8,:)=[ 0 0 4 1 1 12 ];
 M(9,:)=[ 0 1 5 1 0 13 ];
M(10,:)=[ 1 0 5 0 1 13 ];
M(11,:)=[ 1 0 6 0 1 14 ];
M(12,:)=[ 0 1 6 1 0 14 ];
M(13,:)=[ 0 1 7 1 0 15 ];
M(14,:)=[ 1 0 7 0 1 15 ];
M(15,:)=[ 1 0 8 0 1 16 ];
M(16,:)=[ 0 1 8 1 0 16 ];

% M: Matriz de informacion de estados producidos por el codigo CC(2,1,5)y
% los generadores
% g0=1+D^3+D^ y g1=1+D+D^3+D^4.

T=zeros(16,189)+nan;
peso=zeros(16,1);
estado=1;
activado=zeros(16,1);

activado(estado)=1;
%pesos(estado)=0;

for(i=1:189)

   aux1=find(activado==1);
   Taux=T;
   laux1=length(aux1);
   pesoaux=peso;
   activado=zeros(16,1);
   for(j=1:laux1)
     if(activado(M(aux1(j),3))==0)
         activado(M(aux1(j),3))=1;          % Activo los dos estados a
                                            % los que llega.
```

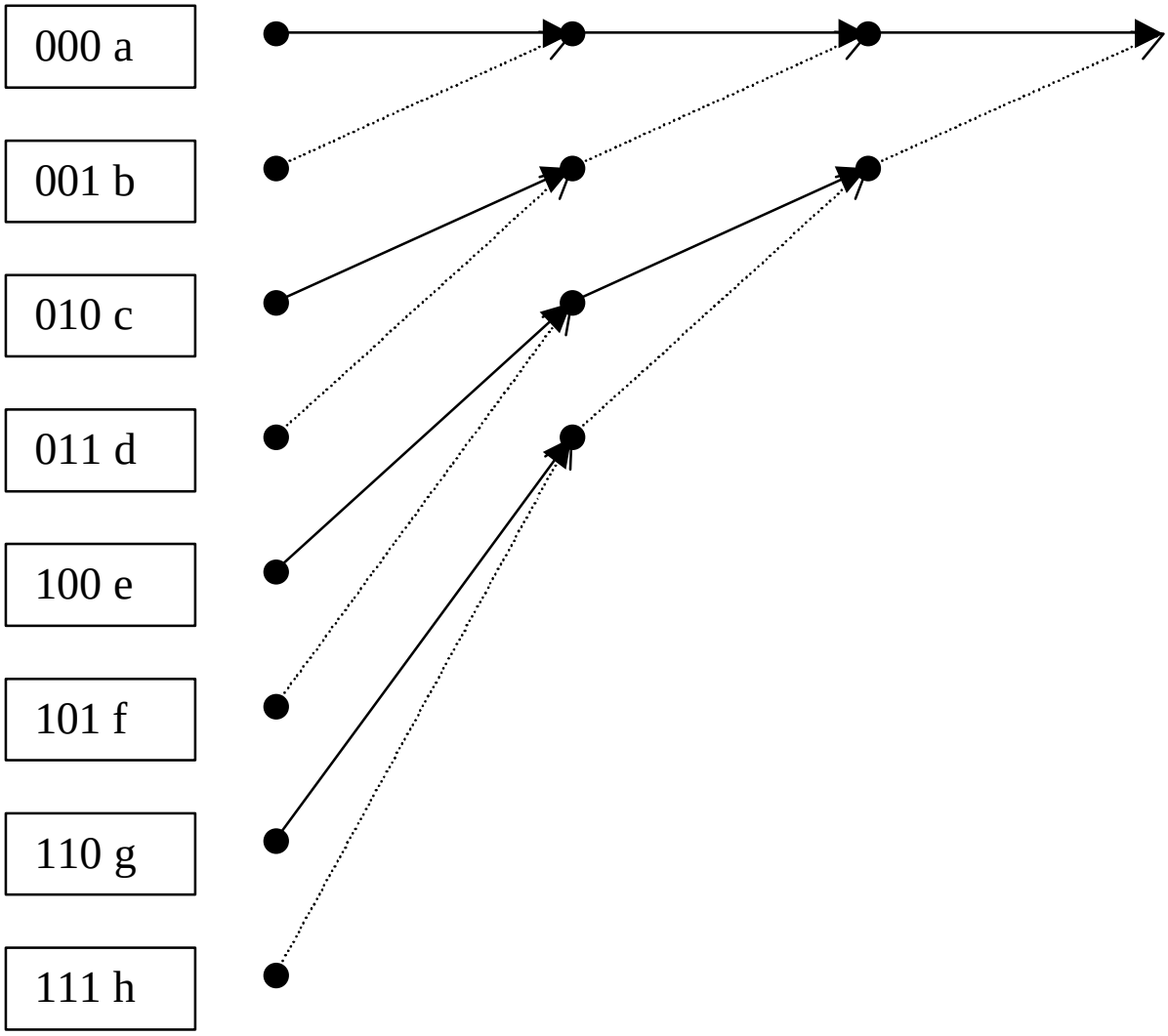
```
peso(M(aux1(j),3))=pesoaux(aux1(j))+sum(xor(M(aux1(j),1:2),codi(2*i-
                                                                1:2*i)));
         T(M(aux1(j),3),:)=Taux(aux1(j),:);
         T(M(aux1(j),3),i)=0;


elseif(peso(M(aux1(j),3))>pesoaux(aux1(j))+sum(xor(M(aux1(j),1:2),codi(2*
                                                              i-1:2*i))))

peso(M(aux1(j),3))=pesoaux(aux1(j))+sum(xor(M(aux1(j),1:2),codi(2*i-
                                                                1:2*i)));
         T(M(aux1(j),3),:)=Taux(aux1(j),:);
         T(M(aux1(j),3),i)=0;

      end
      if(activado(M(aux1(j),6))==0)
         activado(M(aux1(j),6))=1;            % Activo los dos estados a
                                              % los que llega.

peso(M(aux1(j),6))=pesoaux(aux1(j))+sum(xor(M(aux1(j),4:5),codi(2*i-
                                                                1:2*i)));
         T(M(aux1(j),6),:)=Taux(aux1(j),:);
         T(M(aux1(j),6),i)=1;


elseif(peso(M(aux1(j),6))>pesoaux(aux1(j))+sum(xor(M(aux1(j),4:5),codi(2*
                                                              i-1:2*i))))

peso(M(aux1(j),6))=pesoaux(aux1(j))+sum(xor(M(aux1(j),4:5),codi(2*i-
                                                                1:2*i)));
         T(M(aux1(j),6),:)=Taux(aux1(j),:);
         T(M(aux1(j),6),i)=1;
      end
   end
end

aux2=find(peso==min(peso));
deco=fliplr(T(aux2,:));   % porque se introducen en orden inverso al
                          % codificador convolucional
                          % para aprovechar los 4 Tail Bits.
return
```

5.4.3. DECODIFICACIÓN CON ALGORITMO DE VITERBI DE CÓDIGO DE BLOQUE EN GSM.

De igual forma a como hemos presentado la decodificación del código convolucional para el código de bloque también generamos un árbol de trellis a partir de la función que lo genera.


Estado a
000
Estado b
001
Estado c
010
Estado d
011
Estado e
100
Estado f
101
Estado g
110
Estado h
111
Estado a
000
Estado b
001
Estado c
010
Estado d
011
Estado e
100
Estado f
10 1
Estado g
110
Estado h
111
'0'
Entrada:
Salida:
'1'

Si vemos unos cuantos pasos de este diagrama de trellis observamos como los 3 bits nulos que se añaden tras la información codificada con el código de bloque en la decodificación sirven para inicializar el contador. Lo mismo pasa con los cuatro bits nulos que se añaden antes de codificar convolucionalmente y que tras la codificación permanecen con valor '0'.



La función utilizada para decodificar la señal modificada por el código bloque se encuentra en decbloque.m.

```
function [deco]=decbloque(codi)

% Funcion que decodifica la secuencia que se le pasa lacual
% esta codificada segun un codigo ciclico con polinomio G4(D)=D^3+D+1.
```

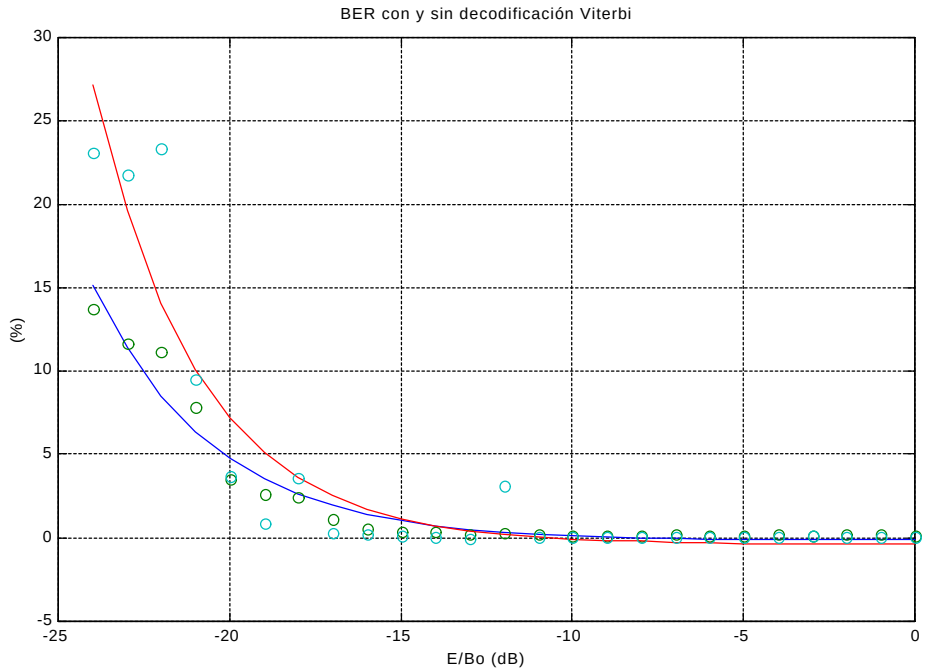
```
%
%      [deco]=decbloque(codi)
%
% deco: Secuencia de bits decodificados con longitud 50.
% codi: Secuencia de bits codificados por un codificador (53,50). Como
%        consecuancia su longitud es de 53.



 M(1,:)=[ 0 1 1 7 ];          % M(1,:)=[ 0   |1|  1   |7| ]
 M(2,:)=[ 0 7 1 1 ];
 M(3,:)=[ 0 2 1 8 ];
 M(4,:)=[ 0 8 1 2 ];
 M(5,:)=[ 0 3 1 5 ];
 M(6,:)=[ 0 5 1 3 ];
 M(7,:)=[ 0 4 1 6 ];
 M(8,:)=[ 0 6 1 4 ];

% M: Matriz de informacion de estados producidos por el codigo bch(53,50)
%    y el generador es
%    G4(D)=D^3+D+1

T=zeros(8,53)+nan;
peso=zeros(8,1);
estado=1;
activado=zeros(8,1);

activado(estado)=1;
%pesos(estado)=0;

for(i=1:50)

   aux1=find(activado==1);
   Taux=T;
   laux1=length(aux1);
   pesoaux=peso;
   activado=zeros(8,1);
   for(j=1:laux1)
     if(activado(M(aux1(j),2))==0)
         activado(M(aux1(j),2))=1;           % Activo los dos estados a
                                             % los que llega.
         peso(M(aux1(j),2))=pesoaux(aux1(j))+xor(M(aux1(j),1),codi(i));
         T(M(aux1(j),2),:)=Taux(aux1(j),:);
         T(M(aux1(j),2),i)=0;


elseif(peso(M(aux1(j),2))>pesoaux(aux1(j))+xor(M(aux1(j),1),codi(i)))
         peso(M(aux1(j),2))=pesoaux(aux1(j))+xor(M(aux1(j),1),codi(i));
         T(M(aux1(j),2),:)=Taux(aux1(j),:);
         T(M(aux1(j),2),i)=0;

      end
      if(activado(M(aux1(j),4))==0)
         activado(M(aux1(j),4))=1;            % Activo los dos estados a
                                              % los que llega.
         peso(M(aux1(j),4))=pesoaux(aux1(j))+xor(M(aux1(j),3),codi(i));
         T(M(aux1(j),4),:)=Taux(aux1(j),:);
```

```
         T(M(aux1(j),4),i)=1;


elseif(peso(M(aux1(j),4))>pesoaux(aux1(j))+xor(M(aux1(j),3),codi(i)))
         peso(M(aux1(j),4))=pesoaux(aux1(j))+xor(M(aux1(j),3),codi(i));
         T(M(aux1(j),4),:)=Taux(aux1(j),:);
         T(M(aux1(j),4),i)=1;
      end
   end
end
clear M;
M(1,:)=[ 0 1 ];          % M(1,:)=[ 0   |1|  1   |7| ]
M(2,:)=[ 1 1 ];
M(3,:)=[ 0 2 ];
M(4,:)=[ 1 2 ];
M(5,:)=[ 0 3 ];
M(6,:)=[ 1 3 ];
M(7,:)=[ 0 4 ];
M(8,:)=[ 1 4 ];



for(i=51:53)
   aux1=find(activado==1);
   Taux=T;
   laux1=length(aux1);
   pesoaux=peso;
   activado=zeros(8,1);
   for(j=1:laux1)
     if(activado(M(aux1(j),2))==0)
         activado(M(aux1(j),2))=1;           % Activo los dos estados a
                                             % los que llega.
         peso(M(aux1(j),2))=pesoaux(aux1(j))+xor(M(aux1(j),1),codi(i));
         T(M(aux1(j),2),:)=Taux(aux1(j),:);
         T(M(aux1(j),2),i)=M(aux1(j),1);


elseif(peso(M(aux1(j),2))>pesoaux(aux1(j))+xor(M(aux1(j),1),codi(i)))
         peso(M(aux1(j),2))=pesoaux(aux1(j))+xor(M(aux1(j),1),codi(i));
         T(M(aux1(j),2),:)=Taux(aux1(j),:);
         T(M(aux1(j),2),i)=M(aux1(j),1);
      end
   end
end

% aux2=find(peso==min(peso));

aux3=T(1,:);
deco=aux3(1:50);
return
```

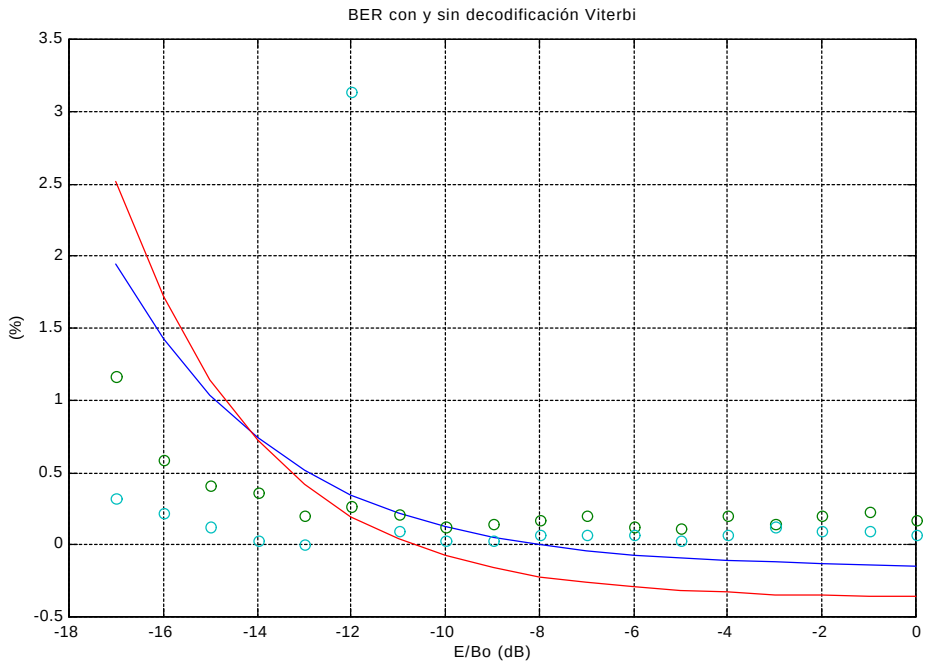
La función que engloba todas estas funciones es decodif.m.

```
function [f]=decodif(bitcod)

% Funcion a la que se le pasan 465 codificados siguiendo las normas de
% codificacion para tramas con datos de voz en el estandar GSM mediante
% por la accion de dos tipos de codificaciones que actuan sobre una trama
% de 260 bits resultado del codec vocal de GSM del tipo RPE-LPT ocupando
% un intervalo de 20ms.
%
%            [f]=decodif(bitcod)
%
% f: Secuencia de bitsresultado del decodificador. Tendra una longitud de
%    260 bits.
% bitcod: Secuencia de bits codificados. Tendra una longitud de 456 bits.

% Los ultimos 78 bits no sufren ningun tipo de codificacion, sino que son
% añadidos al final.
length(bitcod);
grupo2=bitcod(379:456);
cod1a1b=bitcod(1:378);        % Este grupo de bits han sufrido un codigo
                              % covolucional.

grupo1=viterb(cod1a1b);

gp1b=grupo1(54:185);

gp1a=decbloque(grupo1(1:53));

f=[gp1a gp1b grupo2];

      return
```

Repitiendo el proceso para diferentes entornos caracterizados por la relación señal a ruido existente obtenemos un patrón de error presentado en anteriores capítulos así como la evolución de la tasa de error tras pasar por el decodificador Viterbi. Podemos apreciar como para niveles de ruido tolerables existe una mejora en la BER.



La línea roja nos muestra la aproximación a los datos que obtenemos en la simulación tras decodificar la señal. Estos representa una BER inferior, reduciendo los fallos en la información, cuando el entrono presenta menos de 13 dB en la realción señal a ruido. Si observamos esta zona con más detalle vemos lo que decimos.



Ambas líneas están representadas por debajo de 0 dB por tratarse de una aproximación polinómica. Sin embargo podemos observar como se consigue bajar la tasa de error de bit.