Viterbi algorithm

برای توضیح روند کار، ابتدا فایل viterbi\_main تشریح داده میشود و سپس با توجه به آن توضیحات دیگر توابع میجود داده میشود.

clear  
clc  
close all  
% generate random bits  
len = 200;  
bits = randi([0 1], 1, len);  
  
% viterbi encoder with 6 bits memory  
% viterbi\_config([1,1,1,1,0,0,1;1,0,1,1,0,1,1]);  
% out = viterbi\_encoder(bits,0);  
%  
memory = viterbi\_config([1,1,0;1,1,1]);  
out = viterbi\_encoder(bits,0);  
%  
  
% % viterbi decoder  
rec = viterbi\_decoder( out );  
%  
% check : dismatch  
sum(rec(1:len - 5\*memory) ~= bits(1:len - 5\*memory) )

ابتدا در خط 5و6 ام ، بردار تصادفی باینتری bits به طول 200 ایجاد میکند.

در تابع بعدی کافیست یک بار در کل، این الگوریتم برای encoder , decoder آن config شود.

در حقیقت این تابع ماتریس conf را میگیرد و و آن را در Viterbi\_conf.mat ذخیره میکند.

نکته: در صورت عدم توانایی در عوض کردن کانفیگ، می بایست این فایل را در صورت وجود برای اولین بار، پاک نمود و در هر مرحله نیز، نیاز به ذخیره کردن آن ندارد.

ورودی این تابع، یک ماتریس m\*(k+1) می باشد که ذخیره خواهد شد.k تعداد بیت حافظه کانوونوشنال انکودر ماست که همین مقدار k بعنوان خروجی در memory ذخیره شده است( خط 12). m نیز تعداد خروجی هاست بنابراین می باشد.

هر سطر متعلق به یک خروجی ماست. همچنین یک های موجود در هر سطر، به معنای وصل بودن ورودی به خروجی و صفر بودن آن نیز مخالف این است. اندیس هر ستون نیز به معنای تاخیر آن است یعنی مثلا سطر معادل می باشد.

مثال دیگر:

Memory = 2

D

D

با همین ماتریس خروجی Viterbi\_encoder تولید میشود.

function out = viterbi\_encoder (bits,state)  
 load('viterbi\_conf.mat');  
 Size = size(conf);  
 memory\_encoder = encoder(state,Size(2)-1);  
 out = zeros(Size(1), length(bits));  
 for i = 1 : length(bits)  
 [out(:,i), memory\_encoder] = viterbi\_terlis( bits(i) , memory\_encoder );  
 end  
end

تابع encoder مورد استفاده در بالا یک عدد را به یک رشته بیت صفر و یک متناظر با آن عدد تبدل میشود. عدد دوم ورودی، سایز این بردار را تعیین میکند. مثلا :

encoder(5,8) = [ 0 0 0 0 0 1 0 1 ]

مقابل این تابع، تابع decoder([ 0 0 0 0 0 1 0 1 ]) = 5 می باشد.

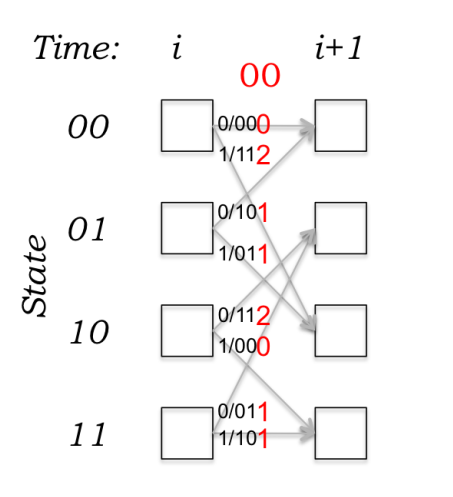
در واقع ورودی bits یک بردار و خروجی آن می باشد . همچنین ورودی state ، حالت اولیه را مشخص میکند که معمولا صفر می باشد.

تابع مورد استفاده ی دیگر در آن تابع viterbi\_terlis می باشد که یک بردار را به عنوان حالت اولیه و یک بیت را بعنوان ورودی میگیرد و خروجی متناظر با آن را که یک ماتریس می باشد. همچنین حالت بعدی را نیز میدهد.

این تابع نقشی حیاتی در هر دو viterbi\_encoder و viterbi\_decoder ایفا میکند و به نوعی نقش کلید مسئله را دارد که در زیر آمده است:

function [out , q] = viterbi\_terlis ( x , q )  
 load('viterbi\_conf.mat');  
 q = [ x q ];  
 out = conf \* q';  
 out = mod(out , 2);  
 q(end) = [];  
end

به صورت گرافیکی این کلید به صورت زیر می باشد.



حال به همان viterbi\_main باز گردیم:

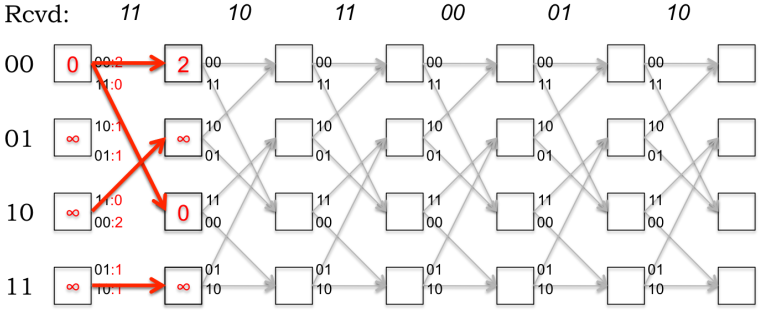
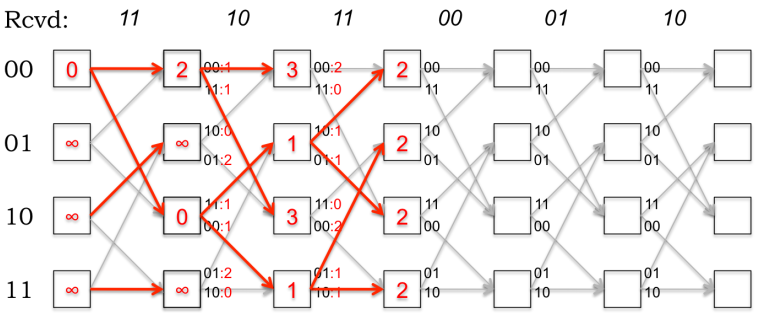
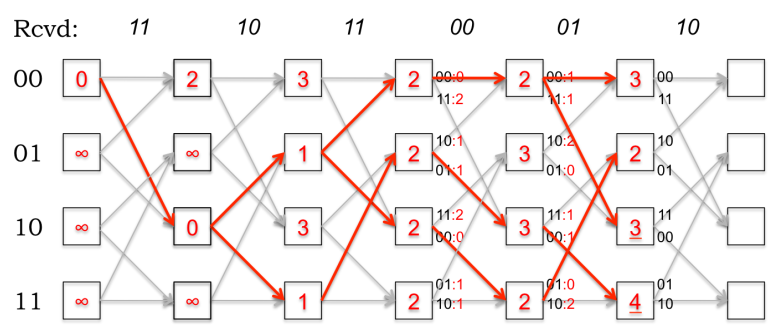
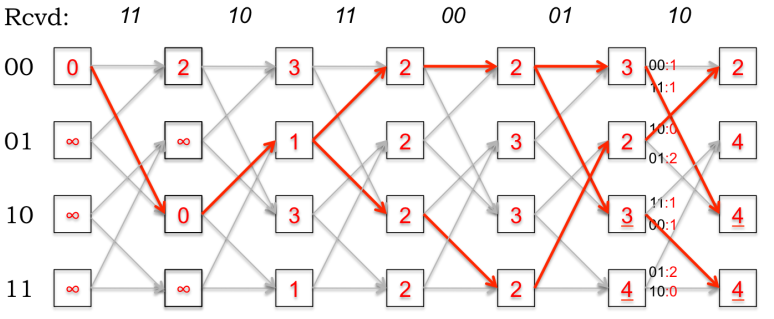
clear  
clc  
close all  
% generate random bits  
len = 200;  
bits = randi([0 1], 1, len);  
  
% viterbi encoder with 6 bits memory  
% viterbi\_config([1,1,1,1,0,0,1;1,0,1,1,0,1,1]);  
% out = viterbi\_encoder(bits,0);  
%  
memory = viterbi\_config([1,1,0;1,1,1]);  
out = viterbi\_encoder(bits,0);  
%  
  
% % viterbi decoder  
rec = viterbi\_decoder( out );  
%  
% check : dismatch  
sum(rec(1:len - 5\*memory) ~= bits(1:len - 5\*memory) )

بنابر این در این مثال out برابر یک بردار است.

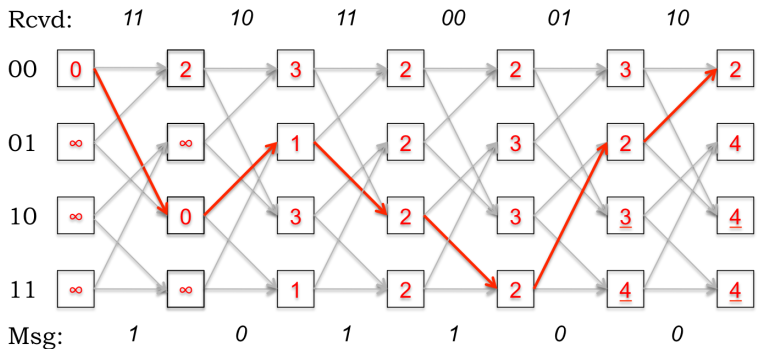
حال می خواهیم این خروجی را decode کنیم و با ورودی مقایسه کنیم. این خروجی از تابع viterbi\_decoder خارج میشود که به صورت زیر می باشد:

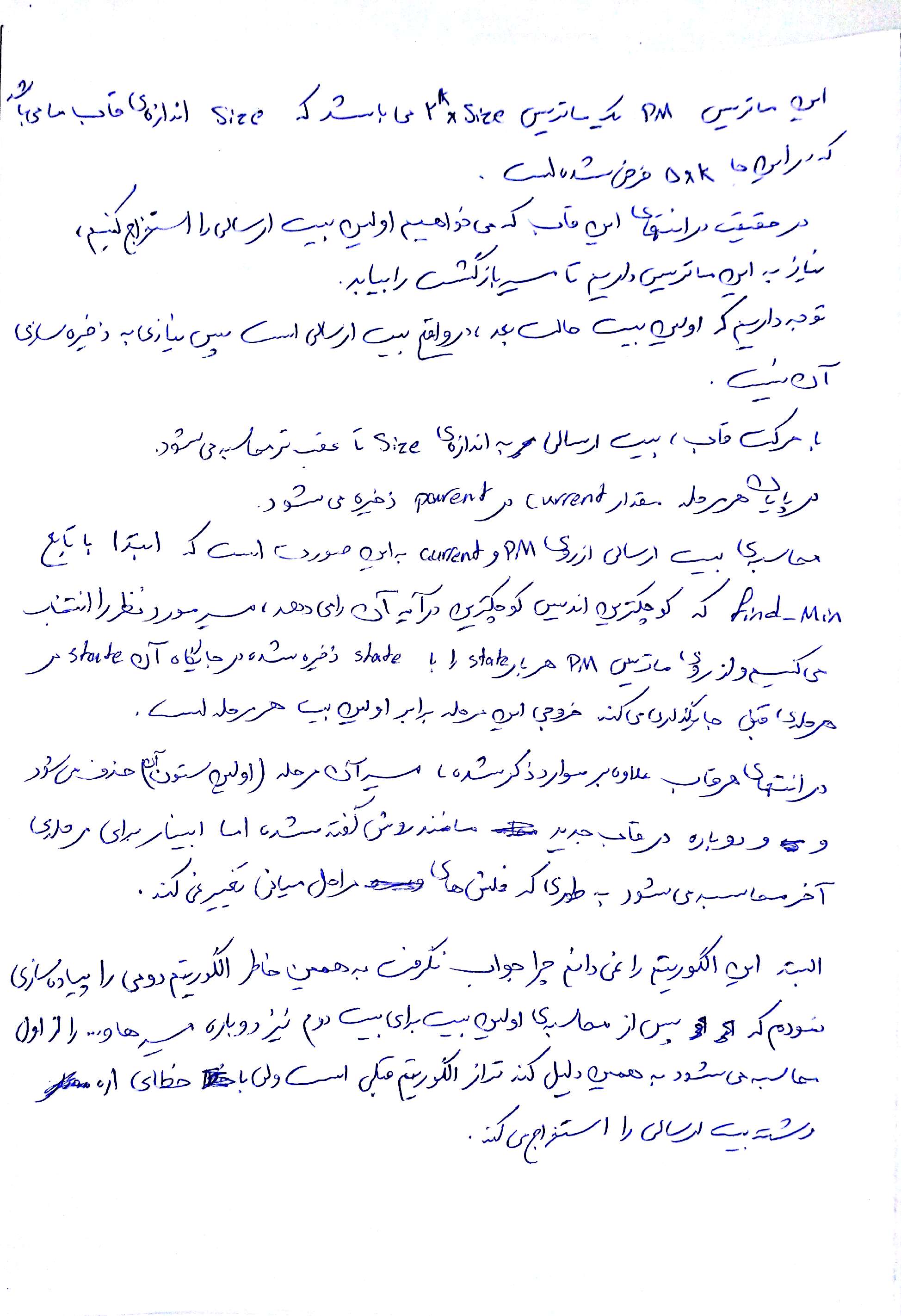
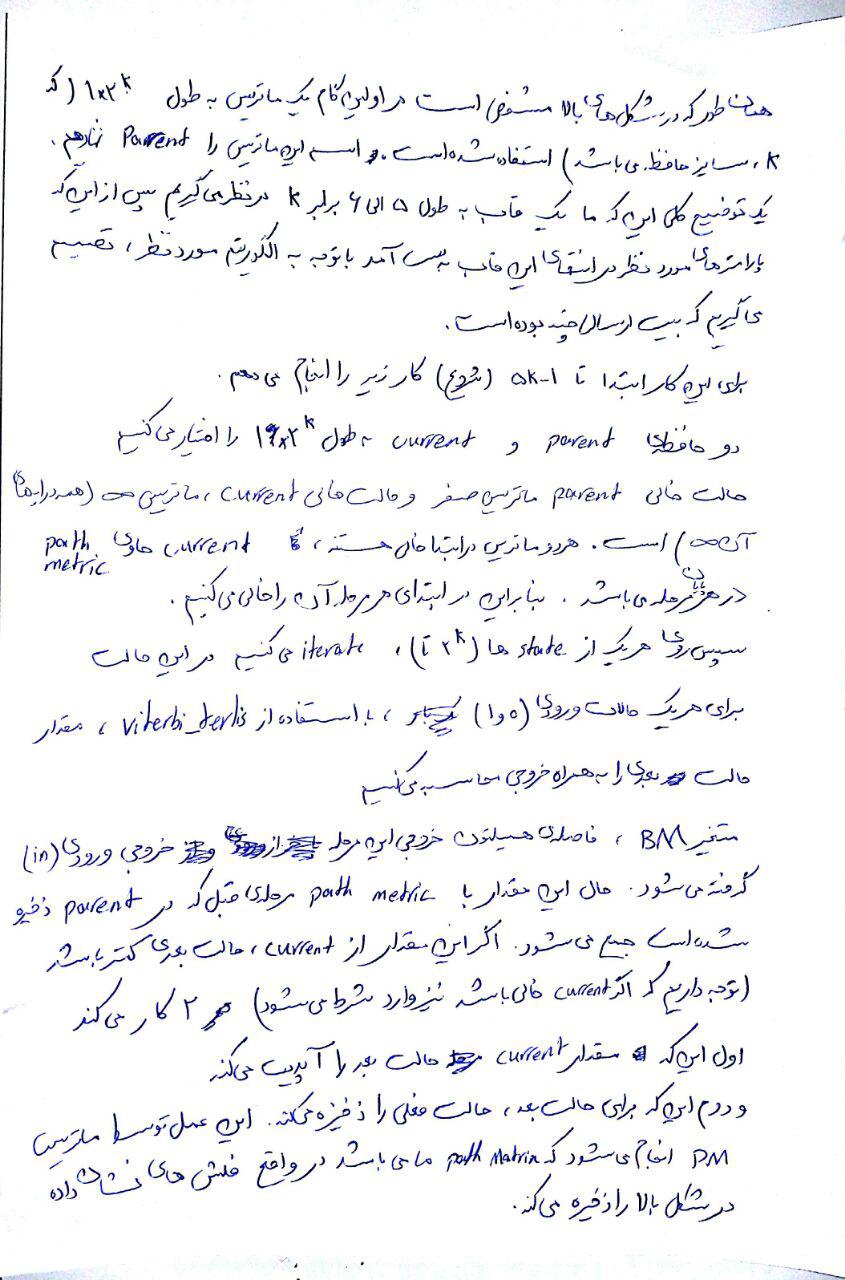
function out = viterbi\_decoder( in )  
 load('viterbi\_conf.mat');%,'conf\_size');  
 k = size(conf,2) -1; % k = length memory  
 parent = zeros(1, 2 ^ k);  
 Size = (k)\*5;  
 PM = zeros( 2^k, Size);  
 out = zeros(1,length(in));  
 for j = ( 1 : Size-1 )  
 current = Inf\*ones(1, 2^k);  
 for i = 1:2^k  
 stat = encoder(i-1,k);  
 for x = [0 1]  
  
 [y,mem] = viterbi\_terlis(x, stat);  
 BM = sum(y ~= in(:,j) );  
  
 temp = parent(i) + BM;  
 next = decoder(mem)+1;  
 if temp <= current(next)  
 PM(next,j) = i;  
 current(next) = temp;  
 end  
 end  
 end  
 parent = current;  
 end  
  
 for offset = 1:(length(in)-Size)  
 current = Inf\*ones(1, 2^k);  
  
 for i = 1:2^k  
 stat = encoder(i-1,k);  
 for x = [0 1]  
  
 [y,mem] = viterbi\_terlis(x, stat);  
 BM = sum(y ~= in(:,j) );  
  
 temp = parent(i) + BM;  
 next = decoder(mem)+1;  
 if temp <= current(next)  
 PM(next,Size) = i;  
 current(next) = temp;  
 end  
 end  
 end  
 parent = current;  
  
  
 Min = find\_min(current);  
  
 for i = (Size):-1:1  
 path = PM(:,i);  
 temp = encoder(Min-1,k);  
 out(offset) = temp(1);  
 Min = path(Min);  
 end  
  
 PM(:,1) = [];  
 PM = [PM , zeros( 2^k , 1) ];  
 end  
end

روند این مسیله در شکل های زیر دیده میشود:

با توجه به خروجی آن مسیری قابل قبول است که کمترین فاصله را دارد. بنابراین مسیر انتخابی مورد نظر، به صورت زیر می باشد. از روی آن به راحتی ورودی قابل محاسبه است.





الگوریتم دوم زیر در زیر ضمیمه شده است.

function out = viterbi\_decoder2( in )  
 load('viterbi\_conf.mat');%,'conf\_size');  
 k = size(conf,2) -1; % k = length memory  
 parent = zeros(1, 2 ^ k);  
 Size = (k)\*5;  
 PM = zeros( 2^k, Size);  
 out = zeros(1,length(in));  
 for offset = 0:( length(in) - Size )  
 for j = ( 1 : Size )  
 current = Inf\*ones(1, 2^k);  
 for i = 1:2^k  
 stat = encoder(i-1,k);  
 for x = [0 1]  
  
 [y,mem] = viterbi\_terlis(x, stat);  
 BM = sum(y ~= in(:,j+offset) );  
  
 temp = parent(i) + BM;  
 next = decoder(mem)+1;  
 if temp < current(next)  
 PM(next,j) = i;  
 current(next) = temp;  
 end  
 end  
 end  
 parent = current;  
 end  
 Min = find\_min(current);  
  
 for i = Size:-1:1  
 path = PM(:,i);  
 temp = encoder(Min-1,k);  
 out(offset+1) = temp(1);  
 Min = path(Min);  
 end  
 end  
end