信道编解码

PB16061024 陈进泽

2018 年 12 月 10 日

1 实验目的

- 使用 MATLAB 进行卷积码编/译码器的仿真
- 熟练掌握 MATLAB 软件、语句
- 了解卷积码编/译码器的原理、知识

2 实验原理

2.1 卷积码编码器

2.1.1 连接表示

卷积码由 3 个整数 n , k , N 描述。k / n 也表示编码效率 (每编码比特所含的信息量) ; 但 n 与线性分组码中的含义不同, 不再表示分组或码子长度; N 称为约束长度, 表示在编码移位寄存器中 k 元组的级数。卷积码不同于分组码的一个重要特征就是编码器的记忆性, 即卷积码编码过程中产生的 n 元组, 不仅是当前输入 k 元组的函数, 而且还是前面 N - 1 个输入 k 元组的函数。实际情况下, n 和 k 经常取较小的值, 而通过 N 的变化来控制编码的能力和复杂性

下面以图 1 中的卷积码编码器为例介绍卷积码编码器。该图表示一个约束长度 K = 3 的 (2, 1) 卷积译码器, 模 2 加法器的数目为 n = 2 , 因此, 编码效率 k / n = 1/ 2 。在每个输入比特时间上,1 位信息比特移入寄存器最左端的一级, 同时将寄存器中原有比特均右移一级, 接着便交替采样两个模 2 加法器, 得到的码元就是与该输入比特相对应的分支字。对每一个输入信号比特都重复上述采样过程

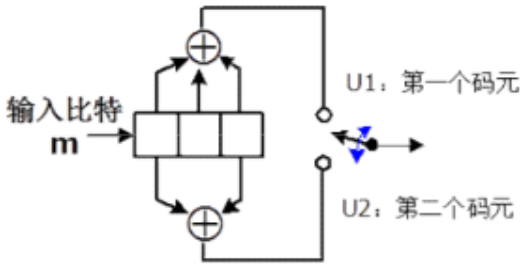

图 1: 卷积码编码器 (编码效率 $1/2, K = 3$)

用于描述反馈移位寄存器实现循环码时所使用的生成多项式也可用户描述卷积码编码器的连接。应用 n 个生成多项式描述编码的移位寄存器与模 2 加法器的连接方式,n 个生成多项式分别对应 n 个模 2 加法器, 每个生成多项式不超过 K - 1 阶。仍以图 1 中的编码器为例, 用生成多项式 $g_1(X)$ 代表上方连接, $g_2(X)$ 代表下方连接, 则有

$$\begin{aligned} g_1(X) &= 1 + X + X^2 \\ g_2(X) &= 1 + X^2 \end{aligned}$$

多项式中的最低阶项对应于寄存器的输入级。输出序列根据如下方式求得:

$$U(X) = m(X)g_1(X)与m(X)g_2(X)交织$$

其中 m 表示输入的信息矢量

2.1.2 状态图

卷积编码器属于有限状态机的器件。“有限”表明状态机制只有有限个不同的状态。有限状态机的状态可以用设备的当前输入和最少的信息量, 来预测设备的输出。状态提供了有关过去序列过程及一组将来可能输出序列的限制, 下一状态总是受到前一状态的限制。将编码器在时刻 t_i 的状态定义为 $X_i = m_{i-1}, m_{i-2}, \ldots, m_{i-K+1}$

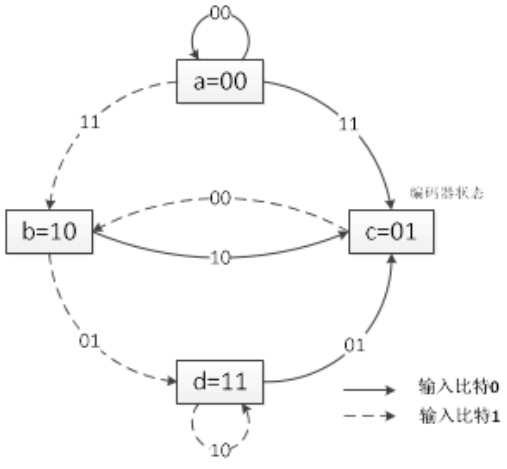

图 2: 状态转移图

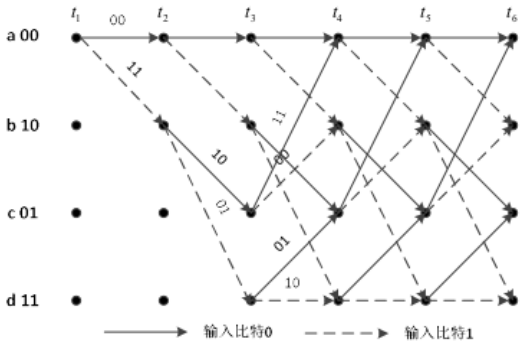

图 3: 编码器网格图

如图 2 所示, 方框内的状态表示寄存器最右端 N - 1 级的内容, 状态间的路径表示由此状态转移时的输出分支字。对应于两种可能的输入 bit, 从每个状态出发只有两种转移

2.1.3 编码器网格图

虽然状态图完全地描述了编码器的特性, 但由于没有表示时间过程, 所以采用状态图跟踪编码器的状态转移很不方便。树状图在状态图的基础上增加了时间尺度。每个相继输入信息比特的编码过程可表述为从左向右经过树状图, 每条数值代表一个输出分支字。树状图上增加的时间尺度是我们可以动态地描述输入序列的编码过程。但由于树状图的规模增长很快, 因而只适于序列中分支子数目较小的情况

我们采用移位寄存器的 4 种可能状态来标注树图的各个节点,a = 00, b = 01, c = 10, d = 11 。树结构的第一次分支在时刻 t_1 , 产生一对节点, 记为 a , b ; 在后继的各个分支处, 节点数翻倍。第二次分支在时刻 t_2 , 生成 4 个节点, 记为 a , b , c , d ; 第三次分支后共有 8 个节点。网格图利用了结构上的重复性, 从而能够更加方便地描述编码器

2.2 维特比译码算法

维特比译码算法由维特比在 1967 年提出。维特比算法的实质是最大似然译码, 但它利用了编码网格图的特殊结构, 从而降低了计算的复杂性。该算法包括计算网格图上在时刻 t_i 到达各个状态的路径和接受序列之间的相似度, 或者说距离。维特比算法考虑的是, 去除不可能成为最大似然选择对

象的网格图上的路径, 即如果有两条路径到达同一状态, 则具有最佳量度的路径被选中, 成为幸存路径。对所有状态都将进行这样的选路操作, 译码器不断在网格图上深入, 通过去除可能性最小的路径实现判决

网格图中每个时刻 t_i 上有 2^{K-1} 个状态, 这里的 K 是约束长度, 每种状态都可经两条路径到达。维特比译码包括计算到达每个状态的两条路径的路径量度, 并舍弃其中一条路径。在时刻 t_i , 算法对 2^{K-1} 个状态 (节点) 都进行上述计算, 然后进入时刻 t_{i+1} , 并重复上述过程。在一个给定的时刻, 各状态的幸存路径量度就是该状态在该时刻的状态量度

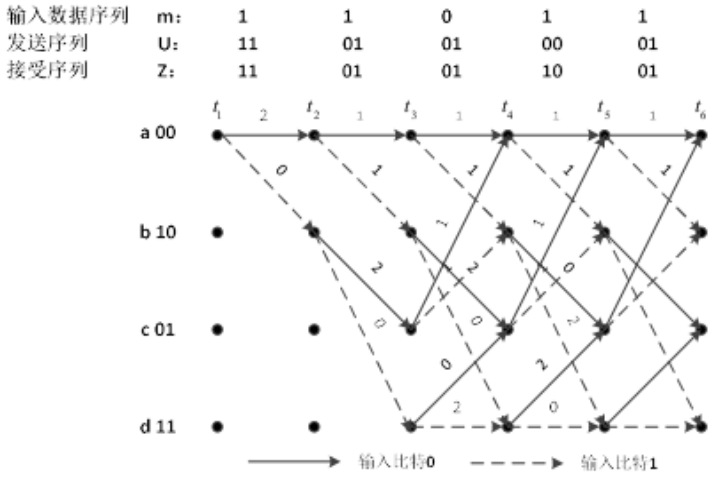

图 4: 译码器网格图

3 实验内容

3.1 源代码

卷积编码与解码代码 channel.m

```
function ber = channel(snr)
  %snr = 0;
  %std_v = 1*10^(-snr/10);
  N = 2000;
  A = [0,0,randi([0,1],1,N)];
  B = zeros(N,3);
```

```
7   for i = 1:N
8     B(i,:) = A(i:i+2);
9   end
10  C(:,1) = mod(sum(B.*[1,1,1],2),2); % g_1 = 1,1,1
11  C(:,2) = mod(sum(B.*[1,0,1],2),2); % g_2 = 1,0,1
12
13  %cr = C + std_v * randn(N,2);
14  cr = awgn(C,snr,'measured');
15  cr = cr>=0.5;
16
17  path = [[0,0; 0,1; 1,0; 1,1],zeros(4,N-2)];
18  newpath = zeros(4,N);
19
20  z = wrapper();
21  len = zeros(4,1);
22  newlen = zeros(4,1);
23  len(1) = sum(cr(1,:)~=[0,0]) + sum(cr(2,:)~=[0,0]);
24  len(2) = sum(cr(1,:)~=[0,0]) + sum(cr(2,:)~=[1,1]);
25  len(3) = sum(cr(1,:)~=[1,1]) + sum(cr(2,:)~=[1,0]);
26  len(4) = sum(cr(1,:)~=[1,1]) + sum(cr(2,:)~=[0,1]);
27
28  flag = zeros(4,1);
29  for i = 3:N
30    if len(1) + sum(z.go(1,0)~=cr(i,:)) < len(3) + sum
        (z.go(3,0)~=cr(i,:))
31      flag(1) = 1;
32      newlen(1) = len(1) + sum(z.go(1,0)~=cr(i,:));
33    else
34      flag(1) = 3;
35      newlen(1) = len(3) + sum(z.go(3,0)~=cr(i,:));
36    end
37
38    if len(1) + sum(z.go(1,1)~=cr(i,:)) < len(3) + sum
```

```
        (z.go(3,1)~=cr(i,:))
39        flag(2) = 1;
40        newlen(2) = len(1) + sum(z.go(1,1)~=cr(i,:));
41      else
42        flag(2) = 3;
43        newlen(2) = len(3) + sum(z.go(3,1)~=cr(i,:));
44      end
45
46      if len(2) + sum(z.go(2,0)~=cr(i,:)) < len(4) + sum
          (z.go(4,0)~=cr(i,:))
47        flag(3) = 2;
48        newlen(3) = len(2) + sum(z.go(2,0)~=cr(i,:));
49      else
50        flag(3) = 4;
51        newlen(3) = len(4) + sum(z.go(4,0)~=cr(i,:));
52      end
53
54      if len(2) + sum(z.go(2,1)~=cr(i,:)) < len(4) + sum
          (z.go(4,1)~=cr(i,:))
55        flag(4) = 2;
56        newlen(4) = len(2) + sum(z.go(2,1)~=cr(i,:));
57      else
58        flag(4) = 4;
59        newlen(4) = len(4) + sum(z.go(4,1)~=cr(i,:));
60      end
61      newpath(1:4,:) = path(flag,:);
62      newpath(:,i) = [0 1 0 1];
63      path = newpath;
64      len = newlen;
65    end
66
67    [v,i] = min(len);
68    errbits = sum(path(i,:)~=A(3:end));
```

```
69    ber = errbits/N;
70  end
```

将计算不同路径选择的部分封装成类 wrapper

```
1   classdef wrapper
2     properties
3       pathRef
4     end
5     methods
6       function obj = wrapper()
7         obj.pathRef = zeros(4,2,3);
8         obj.pathRef(1,1,:) = [1,0,0];
9         obj.pathRef(1,2,:) = [2,1,1];
10        obj.pathRef(2,1,:) = [3,1,0];
11        obj.pathRef(2,2,:) = [4,0,1];
12        obj.pathRef(3,1,:) = [1,1,1];
13        obj.pathRef(3,2,:) = [2,0,0];
14        obj.pathRef(4,1,:) = [3,0,1];
15        obj.pathRef(4,2,:) = [4,1,0];
16      end
17      function [arr,next] = go(obj,channel,input)
18        tmp = squeeze(obj.pathRef(channel,input+1,:))';
19        arr = tmp(2:end);
20        next = tmp(1);
21      end
22      function ret = com(obj)
23        ret = [0 1 1 0 1 0 0 1; 0 1 0 1 1 0 1 0]';
24      end
25    end
26  end
```

原理上来说每次初始化时可以根据 n,k,N 值生成不同路径对应编码结果, 但在这里为了方便我只针对 $n = 2, k = 1, N = 3$ 进行处理。go 方法接受通道与输入比特作为变量, 输出编码比特与编码最终下标,com 方法返回一矩阵, 为加速运算做处理 (未使用)

解码部分原理为每一步对比不同路径到达的汉明距离, 并舍弃数值较大路径。最后找到距离最小路径并加以解码

3.2 评估

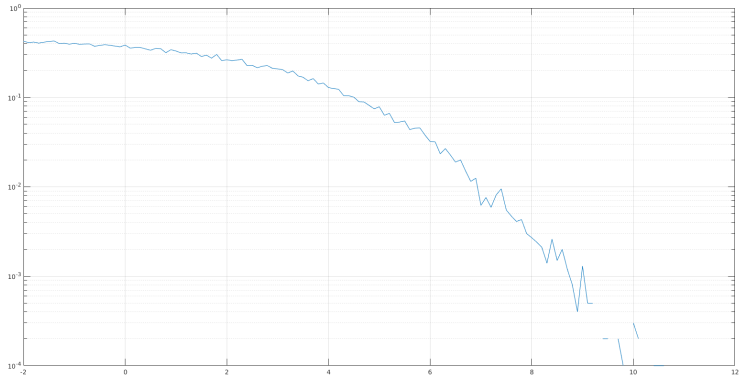

图 5: 误码率曲线

如上图, 为误码率曲线, 可见随 snr 增加误码率下降很快, 说明这种卷积码在差错控制上效果很好