

GEORGIA INSTITUTE OF TECHNOLOGY
School of Electrical and Computer Engineering

ECE 6258
Digital Image Processing
Fall 2003

Problem Set #4–Solutions

Issued: Wednesday, October 1, 2003

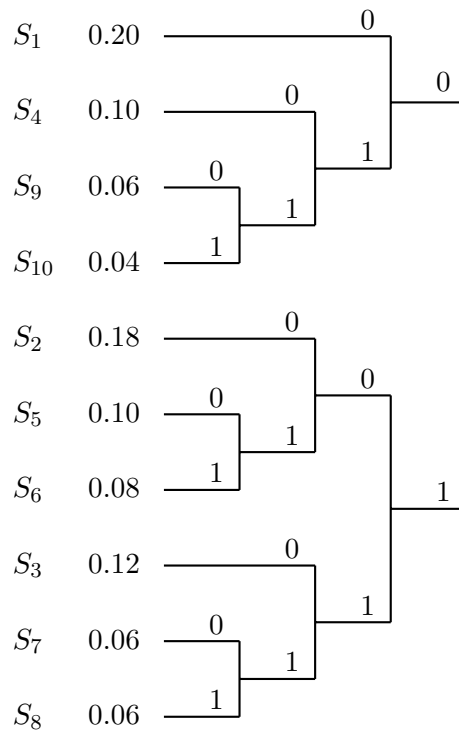
Due (live): Friday, October 10, 2003

Due (video): Friday, October 24, 2003

Problem 4.1 (Huffman Codes): Find a Huffman code for the source shown below. The resulting codes are not unique, but you should try to find a code whose longest code-word is as short as possible.

S_i :	$p(S_i)$
S_1 :	0.20
S_2 :	0.18
S_3 :	0.12
S_4 :	0.10
S_5 :	0.10
S_6 :	0.08
S_7 :	0.06
S_8 :	0.06
S_9 :	0.06
S_{10} :	0.04

Solution: The tree construction for the code is given in the following figure.
(The symbols have been reordered to remove crossovers.)



This gives the following codewords

S_1	00	S_6	1011
S_2	100	S_7	1110
S_3	110	S_8	1111
S_4	010	S_9	0110
S_5	1010	S_{10}	0111

The average codeword length is 3.2 bits/symbol and the entropy is 3.036 bits/symbol.

Problem 4.2 (Tunstall Codes): A Huffman code accepts strings of symbols of equal length and maps them to codewords of variable length. A run-length code, by way of contrast, accepts input strings of variable length and outputs symbols of equal length. The optimal code of the latter type is known as a **Tunstall code**. Tunstall codes are duals of Huffman codes. A Q -ary Tunstall code operating on strings of binary input symbols can be found by the following procedure:

1. First form a tree with two leaves.
 2. Grow the tree by extending the most probable leaf.
 3. Repeat Step 2 until the tree has Q leaves.
 4. The path from the root to the leaf corresponds to the **input** sequence that is mapped to the symbol attached to the leaf.
- (a) Design a Tunstall code for encoding the output of a zero-memory source $S = \{0, 1\}$ with $p(0) = 0.75$, $p(1) = 0.25$. The number of symbols to be encoded in $Q = 8$. Each symbol will be represented by a three-bit number.
- (b) Compute the entropy of the source $H(S)$.
- (c) Compute the ratio of the average number of output bits per input bit.

(d) As an alternative, suppose a run-length code is used to encode the input strings

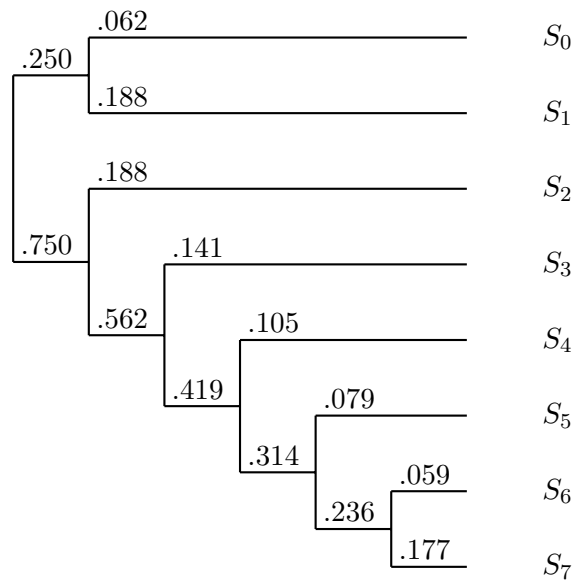
S_1 : 1
 S_2 : 01
 S_3 : 001
 S_4 : 0001
 S_5 : 00001
 S_6 : 000001
 S_7 : 0000001
 S_8 : 0000000

and the results are then Huffman coded. What is the average number of output bits per input bit?

(e) Can you improve on the performance of the Tunstall code by following it with a Huffman code as you did in part (d)?

Solution:

(a) The tree for the code is shown below. The numbers on the branches correspond to probabilities. An input bit of **0** takes the lower branch; an input bit of **1** takes the upper one.



The resulting code is

input string	output string
11	111
10	110
01	101
001	110
0001	011
00001	010
000001	001
000000	000

(b) The entropy is $H(S) = -0.75 \log_2(0.75) - 0.25 \log_2(0.25) = 0.81$ bits/symbol.

(c) The average number of input bits per three output bits is 3.530. Therefore the average number of output bits per input bit is $3.0/3.53 = 0.85$.

- (d) The average number of output bits per input bit is $2.844/3.457 = 0.822$.
 - (e) The eight Tunstall symbols can be coded at 2.93 bits/symbol by following that code with a Huffman code. This reduces the overall bitrate to $2.926/3.53 = 0.829$ output bits/input bit.
-

Problem 4.3 (Arithmetic Coding): This problem is concerned with using an arithmetic coder to encode a Markov source. Consider the simple first-order Markov source defined by the following conditional probabilities:

$$\begin{aligned} p(0|0) &= 0.8 & p(1|0) &= 0.2 \\ p(0|1) &= 0.4 & p(1|1) &= 0.6 \end{aligned}$$

- (a) What are the stationary probabilities, $p(0)$ and $p(1)$?
 - (b) Assuming that this information source is initially modeled as a zero-memory source with the stationary probabilities given in (a), determine the entropy of the source.
 - (c) Encode the source sequence 0100101111000 using an arithmetic coder using the zero-memory source model.
 - (d) What is the entropy of the source using the Markov model?
 - (e) Assuming that the first bit of the sequence is sent unencoded, encode the remainder of the sequence in part (c) using the Markov model.
 - (f) Decode your sequence from part (e).
-

Solution:

- (a) The stationary probabilities must satisfy the equations

$$\begin{aligned} P(0) &= 0.8P(0) + 0.4P(1) \\ P(1) &= 0.2P(0) + 0.6P(1) \end{aligned}$$

Using either of these equations plus the fact that $P(0) + P(1) = 1$ gives $P(0) = 2/3$, $P(1) = 1/3$.

- (b) The entropy is $H(S) = -0.67 \log_2(0.67) - 0.33 \log_2(0.33) = 0.918$ bits/sample.
- (c) Each row in the table below shows the state of the coder before the symbol is sent. A , left edge of interval; B , right edge of interval; W width of interval; F boundary between portion for 0 and 1.

Symbol	<i>A</i>	<i>B</i>	<i>W</i>	<i>F</i>
0	0.000000	1.000000	1.000000	0.666667
1	0.000000	0.666667	0.666667	0.444444
0	0.444444	0.666667	0.222222	0.592592
0	0.444444	0.592592	0.148148	0.543210
1	0.444444	0.543210	0.098765	0.510288
0	0.510288	0.543210	0.032922	0.532236
1	0.510288	0.532236	0.021948	0.524920
1	0.524920	0.532236	0.007316	0.529797
1	0.529797	0.532236	0.002439	0.531423
1	0.531423	0.532236	0.000813	0.531965
0	0.531965	0.532236	0.000271	0.532146
0	0.531965	0.532146	0.000181	0.532085
0	0.531965	0.532085	0.000121	0.532045

So we need to send a fraction between 0.531965 and 0.532045. Since the width of this interval is 80×10^{-6} , we are guaranteed that there is a 15-bit fraction in this interval. The first fifteen bits of the binary expansion of the lower limit are: 100010000010111. Notice that this is longer than the original. This is because this particular sequence is a particularly improbable one for this model.

- (d) The entropy for a Markov source is given by

$$H(S) = \sum_{s_j} P(s_i, s_j) \log_2 \frac{1}{P(s_i | s_j)}$$

Using the model parameters and the results of part (a), this evaluates to 0.805 bits/sample.

- (e) To do this part, we build a table as we did in part (c)

Symbol	<i>A</i>	<i>B</i>	<i>W</i>	<i>F</i>
0				
1	0.000000	1.000000	1.000000	0.800000
0	0.800000	1.000000	0.200000	0.880000
0	0.800000	0.880000	0.080000	0.864000
1	0.800000	0.864000	0.064000	0.851200
0	0.851200	0.864000	0.012800	0.856320
1	0.851200	0.856320	0.005120	0.855296
1	0.855296	0.856320	0.001024	0.855706
1	0.855706	0.856320	0.000614	0.855952
1	0.855952	0.856320	0.000368	0.856099
0	0.856099	0.856320	0.000221	0.856187
0	0.856099	0.856187	0.000088	0.856170
0	0.856099	0.856170	0.000071	0.856156

Finally we need to send a number between 0.856099 and 0.856156. This will require 16 bits. (This particular sequence is even less likely with the Markov model. The data sequence is: 01101101100101010. (Don't forget the initial 0).

- (f) The first step is to convert the sequence back into the number 0.856099+. Knowing the initial intervals, knowing into which interval the number falls allows us to determine the first symbol sent. We then update the intervals and continue.

Problem 4.4 (Optimal Quantization): A sequence of independent Gaussian random variables has a mean of zero and a standard deviation of one. These random variables are to be quantized to five levels.

- ((a)) Design an optimal quantizer (that minimizes the mean-squared quantization error). Specify the thresholds and reproduction values of your quantizer. Your solution may be analytical or numerical, although the latter is recommended.
- (b) Compute the entropy of a sequence of quantized samples.
- (c) Compute the mean-squared error associated with your quantizer.
- (d) Now design an optimal **vector** quantizer, with a codebook of length 25 that quantizes a pair or consecutive samples. *Recommended approach:* Start with a separable version of your quantizer from part (a). Using a test sequence of say 1000 samples, improve your quantizer if possible using several Lloyd iterations.
- (e) What is the entropy **per sample** of your codebook?
- (f) What is the mean-squared error per sample?

Solution:

- (a) The following code uses 1000 samples of a Gaussian random variable to design an optimal quantizer using 10 iterations of the Lloyd algorithm.

```
%
% generate test data
%
x=randn(1000,1);
%
% initial codebook
%
cb(1)= -1.0; cb(2)= -0.5; cb(3)= 0; cb(4)= 0.5; cb(5)= 1.0;
%
% get initial codebook distortion
%
for i=1:5,
    error(:,i)=(x(:)-cb(i)).^2;
end
[dist,indexes]=min(error');
error_var=sum(dist)/1000;
%
% update codebook
%
for lup=1:10,
    cb=zeros(5,1);
    count=zeros(5,1);
    for i=1:1000,
        cb(indexes(i))=cb(indexes(i)) + x(i);
        count(indexes(i))=count(indexes(i))+1;
```

```

end
for i=1:5,
    cb(i)=cb(i)/count(i);
end
for i=1:5,
    error(:,i)=(x(:)-cb(i)).^2;
end
[dist,indexes]=min(error');
error_var=sum(dist)/1000
end
%
% print final codebook
%
cb
error_var
count/1000

```

The final set of reproduction values is: -1.7827 , -0.8004 , -0.0346 , 0.7333 , and 1.7574 .

- (b) The probabilities associated with the four quantized values are 0.102, 0.225, 0.303, 0.254, and 0.116. The entropy associated with this particular distribution is $H(S) = 2.205$ bits/sample.
- (c) The mean squared error as output by the above script was $E = 0.0859$.
- (d) The code for the 2-D codebook is shown below.

```

%
% generate test data
%
x=randn(5000,2);
%
% initial codebook
%
cb(1,:)= [-1.0 -1.0]; cb(2,:)= [-1.0 -0.5]; cb(3,:)= [-1.0 0.0];
cb(4,:)= [-1.0 0.5]; cb(5,:)= [-1.0 1.0]; cb(6,:)= [-0.5 -1.0];
cb(7,:)= [-0.5 -0.5]; cb(8,:)= [-0.5 0.0]; cb(9,:)= [-0.5 0.5];
cb(10,:)= [0.5 1.0]; cb(11,:)= [0.0 -1.0]; cb(12,:)= [0.0 -0.5];
cb(13,:)= [0.0 0.0]; cb(14,:)= [0.0 0.5]; cb(15,:)= [0.0 1.0];
cb(16,:)= [0.5 -1.0]; cb(17,:)= [0.5 -0.5]; cb(18,:)= [0.5 0.0];
cb(19,:)= [0.5 0.5]; cb(20,:)= [0.5 1.0]; cb(21,:)= [1.0 -1.0];
cb(22,:)= [1.0 -0.5]; cb(23,:)= [1.0 0.0]; cb(24,:)= [1.0 0.5];
cb(25,:)= [1.0 1.0];
%
% get initial codebook distortion
%
[rcb,ccb]=size(cb);
for i=1:rcb,
    error(:,i)=(x(:,1)-cb(i,1)).^2+(x(:,2)-cb(i,2)).^2;
end
[dist,indexes]=min(error');
[r,c]=size(x); error_var=sum(dist)/r;

```

```

%
% update codebook
%
for lup=1:10,
    count=zeros(25,1);
    for i=1:r,
        cb(indexes(i),:)=cb(indexes(i),:) + x(i,:);
        count(indexes(i))=count(indexes(i))+1;
    end
    for i=1:25,
        cb(i,:)=cb(i,+)/count(i);
    end
    [rcb,ccb]=size(cb);
    for i=1:rcb,
        error(:,i)=(x(:,1)-cb(i,1)).^2+(x(:,2)-cb(i,2)).^2;
    end
    [dist,indexes]=min(error');
    [r,c]=size(x);
    error_var=sum(dist)/r
end
logcount=(count/5000).*log(5000./count)/log(2);
entropy=sum(logcount');
%
% print final codebook
%
cb
error_var
entropy

```

The number of samples was increased to 5000 and the initial codebook was made tighter than suggested in the problem statement to reduce the likelihood of unpopulated decision regions, which were observed to occur. The final codebook was:

-1.6131	-1.5776	-2.2140	-0.5994
-1.3814	-0.0565	-1.9974	0.7539
-1.3753	1.7816	-0.7343	-1.6196
-0.8293	-0.7096	-0.6014	0.0202
-0.7976	0.7709	0.2770	2.3518
0.1258	-2.1015	-0.0569	-0.8367
0.0117	-0.1244	0.0161	0.6234
-0.3225	1.4910	0.4920	-1.3415
0.7838	-0.6582	0.5972	-0.0035
0.9410	0.5624	0.5592	1.1900
1.4620	-1.4430	2.1306	-0.5779
1.3339	-0.0945	2.0862	0.6992
1.3776	1.6678		

- (e) The entropy was 4.4653 per vector (pair or samples).
(f) The final mean squared error after ten iterations 0.1420.