

UNIVERSITY OF NAMIBIA

FACULTY OF ENGINEERING AND TECHNOLOGY

Eng. Jose Eduardo Dos Santos Campus

DEPARTMENT OF ELECTRONICS ENGINEERING

Digital Communications

DESIGN EXERCISE:

IMPLEMENTATION OF HUFFMAN CODING IN MATLAB

MWANDILA VICTOR ()

DUE DATE: 30/10/15

Introduction

In communication theory, bandwidth is always a limitation. It is therefore an important design practice to represent discrete source data efficiently. The process of representing discrete source data is known as source coding and the device that performs source coding is known as a source encoder. For efficient source coding it is of great essence that one acquires statistical knowledge of the source data. In particular, to know whether some source symbols are known to occur more probable than others. This information can be exploited in the generation of a source code where short codes are assigned to frequent source symbols and long codes to lesser frequent source symbols. This type of a source code is known as a variable-length code and it is most likely to be an efficient method of source coding rather than assigning all symbols to a code of equal length, fixed-length code. An example of such source coding methods that exploits probability of occurrence is the Morse code, where letters of the alphabet and numbers are represented by dots and dashes. However this exercise looks at a different type of a variable-length source coding method known as “Huffman coding”.

Objectives

Discuss the Huffman code theory and algorithm

Write a Matlab program to produce Huffman code

Huffman coding

Huffman coding came about in the year 1951 when an MIT professor Robert M Fano gave his Information theory students a term paper on the problem of finding the most efficient binary code to replace of an end of term test. One of his students, David Huffman, unable to prove that any of the binary code were efficient almost gave up on the task until he finally stumbled upon the idea of using a frequency-sorted binary tree and proved this method as the most efficient [1].

Huffman coding is a uniquely decodable code or a prefix-free code meaning that there is no code word in the source encoder that appears as a first segment (prefix) of any other code word in the system. For Example: [01, 11, 011,111] is not a prefix-free code because ‘01’ is a prefix to ‘011’ and ‘11’ is a prefix to ‘111’ however [01,11,001,101] is a prefix-free code because no code is a prefix to the other codes. Thus said, Huffman code does not need any special markers to separate

words in a message. The receiver can unambiguously decode the message by repeatedly finding and removing sequences that form valid code words without making any errors.

To generate a Huffman code for each symbol in a defined set of inputs or random variables, probability is taken into consideration very firmly. Using this probability a Huffman tree diagram can be generated which in return aids the mapping of each individual symbol to a Huffman code. Huffman code is also mostly an optimum code since it uses minimum number of bits to code the symbols in the set of random variables.

The Huffman sorted tree algorithm (procedure)

Assuming that we have a set of random variables $X = x_1, x_2, x_3, x_4$ having probabilities $P(x) = P(x_1), P(x_2), P(x_3)$ and $P(x_4)$ increasing from x_4 to x_1 then to generate a Huffman tree the following procedures can be followed.

Step 1: The probabilities of the random variables are arranged descending order. In the example above the order is maintained since it is stated that the probability increases from x_4 to x_1 .

$$P(x) = P(x_1), P(x_2), P(x_3), P(x_4)$$

Step2: the least two probabilities are then added and the sum become the intermediate root of the two probabilities.

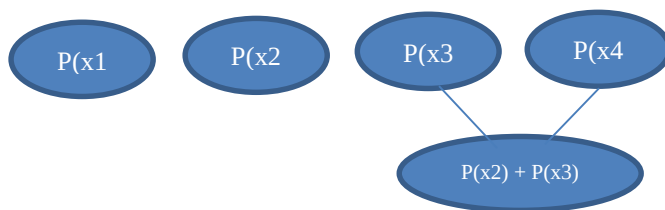


Figure 1. Initial Huffman tree design procedure.

The sum become part of the remaining probabilities and once again the two least probabilities are added together to form yet another root. In this example it will be assumed that the sum $P(x_3)$ and $P(x_4)$ is less than $P(x_2)$ and $P(x_1)$ therefore the next root is made by $P(x_2)$ and $P(x_3)+P(x_4)$

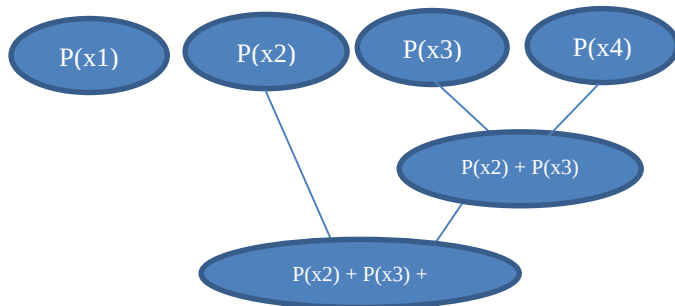


Figure 2. Huffman tree design procedure.

Since the summation of all probabilities of a set of random variables is 1, 1 becomes the ultimate root of the tree at this stage for the example stipulated.

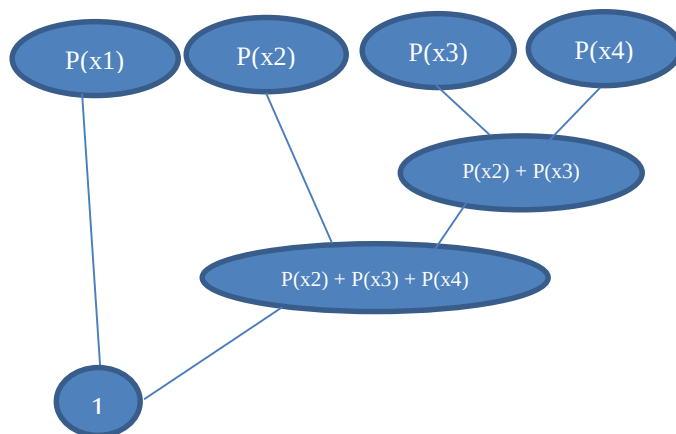


Figure 3. The Huffman tree

Step 3. Each branch of the Huffman tree is assigned either a '0' or a '1'. Putting in mind that at every root, the branches cannot be assigned the same code – they cannot both be '0's or '1's.

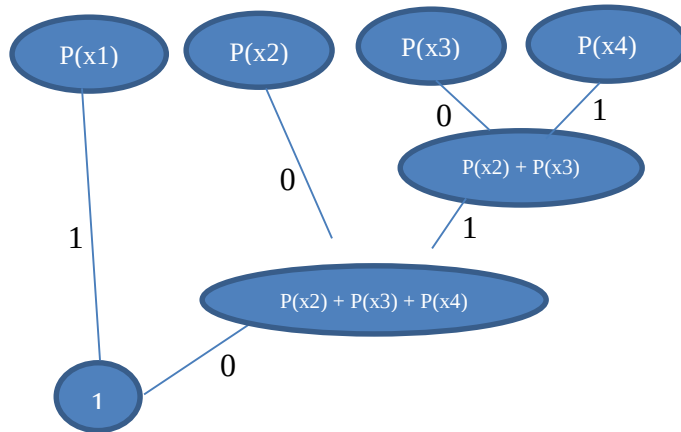


Figure 4. A complete Huffman tree.

This tree now help with deriving a Huffman code by tracing up from the ultimate root, 1, to each of the random variables' probability. For every path taken, the collective values of the branches followed becomes the Huffman code of the destination random variable. In the example above the Huffman code for the variables x_1 , x_2 , x_3 , x_4 can be found as follows:

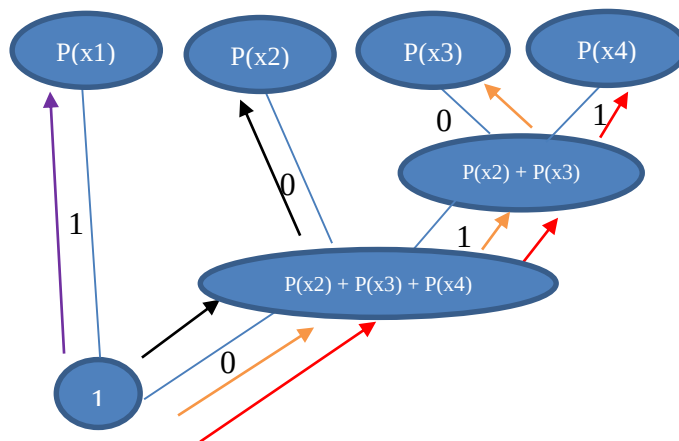


Figure 5. Tracing to acquire the Huffman code from the Huffman tree.

x1 coded purple: 1

x2 coded black: 00

x3 coded orange: 010

x4 coded red: 011

As described in the theory of the Huffman code, the code generated is clearly a prefix-free code and is optimum since it uses the minimum number of bit to code the symbols.

Based on the above discussion, a Matlab program can be designed to take in probabilities of a set of random variables and generate from them a Huffman code.

Matlab code

M-file named Huffman.m

```
function CODE = huffman(p)
%HUFFMAN Builds a variable-length Huffman code for a symbol source.
%CODE = HUFFMAN(P) returns a Huffman code as binary strings in cell
%array CODE for input symbol probability vector P. Each word in CODE
%corresponds to a symbol whose probability is at the corresponding index
% of P.
%
%check the inputs if they are reasonable and returns an error message else
%proceeds
error(nargchk(1,1,nargin));
if (ndims(p) ~= 2) || (min(size(p))>1) || ~isreal(p) || ~isnumeric(p)
error('P must be a real number');
end
% Global variable surviving all recursions of function 'makecode'
global CODE
CODE = cell(length(p),1);           % Initializes the global cell array
if length(p) > 1                   % checks if there is more than one
symbol...
p = p/sum(p);                      % Normalize the input probabilities
s = reduce(p);                     % Does Huffman source symbol reductions
makecode(s, []);                  % Recursively generate the code
else
CODE = {'1'};                      % Else, it is a one symbol case
end
%.....%
function s = reduce(p);
%Creates a Huffman source reduction tree in a MATLAB cell structure by
%performing source symbol reductions until there are only two reduced
%symbols remaining.
s = cell(length(p),1);
%Generate a starting tree with symbol nodes 1,2,3,... to reference the
```

```

%symbol probabilities.
for i = 1:length(p)
s{i} = i;
end
while numel(s) > 2
[p,i] = sort(p);           % Sort the symbol probabilities in descending
order
p(2) = p(1) + p(2);        % Add the 2 lowest probabilities
p(1) = [];                 % and remove the 2 lowest ones
s = s(i);                  % Reorder tree for new probabilities.
s{2} = {s{1},s{2}};        % and add its lowest nodes again
s(1) = [];
end
%.....%
function makecode(sc, codeword)
% Scan the nodes of a Huffman source reduction tree recursively to
% generate the indicated variable length code words.
% Global variable surviving all the recursive calls
global CODE
if isa(sc,'cell')           % For cell array nodes,
makecode(sc{1},[codeword 0]); % add 0 if the 1st element,
makecode(sc{2}, [codeword 1]); % or a 1 if the 2nd
else                         % For leaf
(numeric) nodes,
CODE{sc} = char('0' + codeword); % create a char code string
end

```

M-file named example.m

This file is used for probability entries.

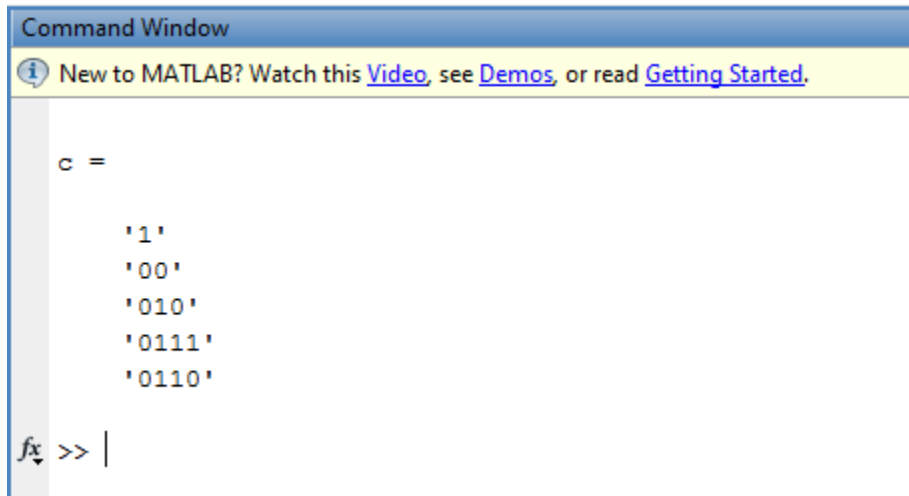
In this matlab program, 5 entries were picked randomly [0.50 0.20 0.15 0.1 0.05]. For each entry a Huffman code was produced and can be seen in the results obtained after the test.

```

clc
% provide the input probabilities to create the huffman code
f = [0.50 0.20 0.15 0.1 0.05];
%format of entries = [0.5 0.25 0.1 0.15];
if (sum(f) ~= 1)
error('The total of the probabilities should be 1');
end
c = huffman(f) %calling the huffman function

```

Results



```
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

c =

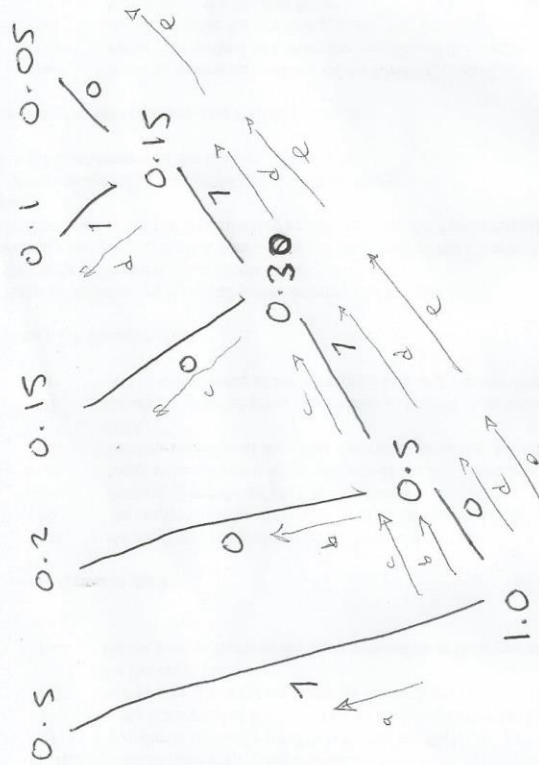
    '1'
   '00'
  '010'
 '0111'
'0110'
```

The screenshot shows a MATLAB Command Window. At the top, there is a blue header bar with the text "Command Window". Below it is a yellow banner with an information icon and the text "New to MATLAB? Watch this [Video](#), see [Demos](#), or read [Getting Started](#)." The main area of the window displays the output of a command. It starts with "c =" followed by five lines of strings: "'1'", "'00'", "'010'", "'0111'", and "'0110'". At the bottom, there is a prompt "fx >> |" with a cursor.

The results show a code that seems to qualify as both a prefix-free code and an optimum code. These qualities of the code produced can lead to a conclusion that the code produced is a Huffman code.

This program is proven to produce the right results by the use of the Huffman sorted tree to calculate the Huffman code for the five symbols.

Manual Result

SORTED HUFFMAN TREE METHOD
 $P(x) = 0.5, 0.2, 0.15, 0.1, 0.05$


Huffman code :

a	1
b	00
c	010
d	011
e	0110

Conclusion

The Huffman program was successfully designed in Matlab and the result was able to match up with the manually obtained result using the sorted tree method. The code produced was also noticed to be a prefix-free code and used the minimum average number of bits which entails that the code was optimum. With the two properties of the code produced, it is safe to emphasize that the Huffman code was successful generated. The program is also able to take any number of inputs that add up to 1 in the m-file “example.m” and still produce a Huffman code. Nevertheless the program was not designed to take any symbols but only takes probabilities, which may be confusing for a user.

References

1. [Thomas H. Cormen](#), [Charles E. Leiserson](#), [Ronald L. Rivest](#), and [Clifford Stein](#). *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. [ISBN 0-262-03293-7](#). Section 16.3, pp. 385–392