# An introduction to Information Theory

Adrish Banerjee

Department of Electrical Engineering
Indian Institute of Technology Kanpur
Kanpur, Uttar Pradesh
India

July 25, 2016



## Lecture #4: Block to variable length coding-III: Huffman Coding

# Outline of the lecture

- Huffman coding



Huffman coding Coding an information source

# Outline of the lecture

- Huffman coding
- Coding an information source

Adrish Banerjee Department of Electrical Engineering Indian Institute of Technology Kanpur Kanpur, Uttar Pradesh India
An introduction to Information Theory

# Outline







# Huffman code

- The binary tree of an optimum binary prefix-free code for U has no unused leaves.

  Sketch of Proof:

# Huffman code

- The binary tree of an optimum binary prefix-free code for U has no unused leaves.

  Sketch of Proof:
- If the tree has unused leaves, they must be at maximum depth as the code is optimal.

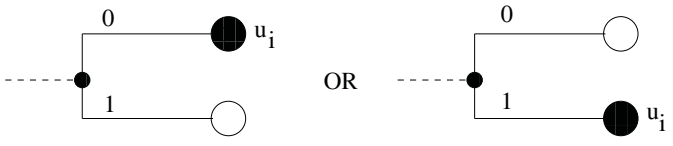




# Huffman code

- The binary tree of an optimum binary prefix-free code for U has no unused leaves.

  Sketch of Proof:
- If the tree has unused leaves, they must be at maximum depth as the code is optimal.
- For atleast one value of $u_i$ of U, we have following situation.

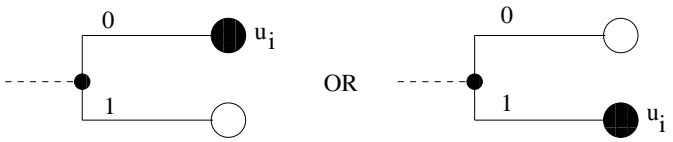


# Huffman code

- The binary tree of an optimum binary prefix-free code for U has no unused leaves.

  Sketch of Proof:

- If the tree has unused leaves, they must be at maximum depth as the code is optimal.
- For atleast one value of $u_i$ of U, we have following situation.




- In either case, we can delete the last digit of the codeword, and still have a prefix-free code. The new code has smaller E[W] and thus original code must not be optimal.





# Huffman code

- There is an optimal binary prefix-free code for U such that the two least likely codewords, say those for $u_{K-1}$ and $u_K$, differ only in their last digit.

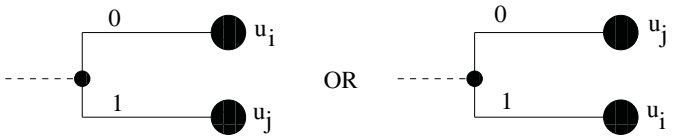
  Sketch of Proof:

# Huffman code

- There is an optimal binary prefix-free code for U such that the two least likely codewords, say those for $u_{K-1}$ and $u_K$, differ only in their last digit.

  Sketch of Proof:

- Assume $P_U(u_{K-1}) \geq P_U(u_K)$. We have the following situation.





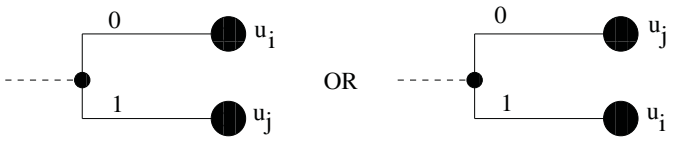



# Huffman code

- There is an optimal binary prefix-free code for U such that the two least likely codewords, say those for $u_{K-1}$ and $u_K$, differ only in their last digit.

  Sketch of Proof:

- Assume $P_U(u_{K-1}) \geq P_U(u_K)$. We have the following situation.



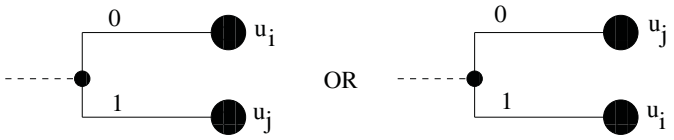

- If $j \neq K$, we switch the leaves for $u_j$ and $u_K$ without increasing E[W].

# Huffman code

- There is an optimal binary prefix-free code for U such that the two least likely codewords, say those for $u_{K-1}$ and $u_K$, differ only in their last digit.

  Sketch of Proof:

- Assume $P_U(u_{K-1}) \geq P_U(u_K)$. We have the following situation.




- If $j \neq K$, we switch the leaves for $u_j$ and $u_K$ without increasing E[W].
- Similarly if $i \neq K-1$, we switch the leaves for $u_i$ and $u_{K-1}$ without increasing E[W].



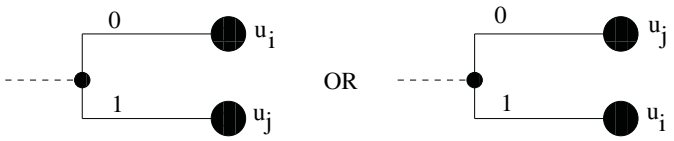


# Huffman code

- There is an optimal binary prefix-free code for U such that the two least likely codewords, say those for $u_{K-1}$ and $u_K$, differ only in their last digit.

  Sketch of Proof:

- Assume $P_U(u_{K-1}) \geq P_U(u_K)$. We have the following situation.




- If $j \neq K$, we switch the leaves for $u_j$ and $u_K$ without increasing E[W].
- Similarly if $i \neq K-1$, we switch the leaves for $u_i$ and $u_{K-1}$ without increasing E[W].
- The new optimum code has its two least likely codewords differing only in their last digit.

# Huffman code

- Huffman coding: Algorithm for constructing binary prefix-free codes for K-ary random variable U





# Huffman code

- Huffman coding: Algorithm for constructing binary prefix-free codes for K-ary random variable U
- Step 0: Designate K vertices as $u_1, u_2, \cdots, u_K$ and assign probability $P_U(u_i)$ to vertex $u_i$. These vertices are designated as "active" nodes.

# Huffman code

- Huffman coding: Algorithm for constructing binary prefix-free codes for K-ary random variable U
- Step 0: Designate K vertices as $u_1, u_2, \cdots, u_K$ and assign probability $P_U(u_i)$ to vertex $u_i$. These vertices are designated as "active" nodes.
- Step 1: Create a node that ties together the two least likely active vertices with binary branches and assign probability equal to the sum of two vertices to this new node. Active this new node and deactive the two vertices that are joined.



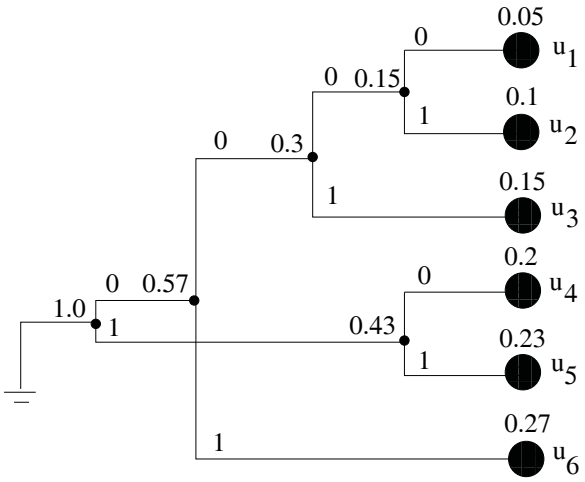


# Huffman code

- Huffman coding: Algorithm for constructing binary prefix-free codes for K-ary random variable U
- Step 0: Designate K vertices as $u_1, u_2, \cdots, u_K$ and assign probability $P_U(u_i)$ to vertex $u_i$. These vertices are designated as "active" nodes.
- Step 1: Create a node that ties together the two least likely active vertices with binary branches and assign probability equal to the sum of two vertices to this new node. Active this new node and deactive the two vertices that are joined.
- If there is only one active vertex left. make it root and stop, otherwise go back to step 1.

# Huffman code

- Construct binary Huffman code for the following example: $P(u_1) = 0.05, P(u_2) = 0.1, P(u_3) = 0.15, P(u_4) = 0.2, P(u_5) = 0.23, P(u_6) = 0.27$




| U | Z |
|---|---|
| $u_1$ | 0000 |
| $u_2$ | 0001 |
| $u_3$ | 001 |
| $u_4$ | 10 |
| $u_5$ | 11 |
| $u_6$ | 01 |





# Huffman code

- The number of leaves in a finite D-ary tree is always given by D + q(D-1) where q is the number of nodes not counting the root.

  Sketch of proof:

# Huffman code

- The number of leaves in a finite D-ary tree is always given by D + q(D-1) where q is the number of nodes not counting the root.

  Sketch of proof:
- In constructing a tree from root, we get D leaves initially.



Huffman coding Coding an information source

# Huffman code

- The number of leaves in a finite D-ary tree is always given by D + q(D-1) where q is the number of nodes not counting the root.

  Sketch of proof:
- In constructing a tree from root, we get D leaves initially.
- At each subsequent step, if we extend any leaf we get D new leaves and lose one old leaf.

# Huffman code

- There are atmost D-2 unused leaves in the tree of an optimal prefix-free D-ary code for U and all are at maximum length.

  Sketch of proof:



Huffman coding Coding an information source

# Huffman code

- There are atmost D-2 unused leaves in the tree of an optimal prefix-free D-ary code for U and all are at maximum length.

  Sketch of proof:

- If unused leaves are not at maximum length, we could decrease E[W] by transferring one of the codewords at maximum length to this leaf. Hence, unused leaves for optimal prefix-free D-ary code can only be at maximum length.

Adrish Banerjee Department of Electrical Engineering Indian Institute of Technology Kanpur Kanpur, Uttar Pradesh India
An introduction to Information Theory

# Huffman code

- There are atmost D-2 unused leaves in the tree of an optimal prefix-free D-ary code for U and all are at maximum length.

  Sketch of proof:
- If unused leaves are not at maximum length, we could decrease E[W] by transferring one of the codewords at maximum length to this leaf. Hence, unused leaves for optimal prefix-free D-ary code can only be at maximum length.
- If there are $D-1$ unused leaves in the tree of optimal prefix-free code, then we can shorten the codeword by removing its last digit.





# Huffman code

- The number of unused leaves in the tree of an optimal D-ary prefix-free code for a random variable U with K possible values, $K \geq D$, is the remainder when $(K-D)(D-2)$ is divided by $D-1$. Proof:

# Huffman code

- The number of unused leaves in the tree of an optimal D-ary prefix-free code for a random variable U with K possible values, $K \geq D$, is the remainder when $(K-D)(D-2)$ is divided by $D-1$. Proof:
- Let r be the number of unused leaves. Then if U has K values,

$$r = [\text{number of leaves in D-ary tree of the code}] - K$$





# Huffman code

- The number of unused leaves in the tree of an optimal D-ary prefix-free code for a random variable U with K possible values, $K \geq D$, is the remainder when $(K-D)(D-2)$ is divided by $D-1$. Proof:
- Let r be the number of unused leaves. Then if U has K values,

$$r = [\text{number of leaves in D-ary tree of the code}] - K$$

- This implies

$$r = [D + q(D-1)] - K$$

or

$$D - K = -q(D-1) + r \quad \text{where} \quad 0 \leq r < D-1$$

# Huffman code

- The number of unused leaves in the tree of an optimal D-ary prefix-free code for a random variable U with K possible values, $K \geq D$, is the remainder when $(K-D)(D-2)$ is divided by $D-1$. Proof:
- Let r be the number of unused leaves. Then if U has K values,

$$r = [\text{number of leaves in D-ary tree of the code}] - K$$

- This implies

$$r = [D + q(D-1)] - K$$

or

$$D - K = -q(D-1) + r \quad \text{where} \quad 0 \leq r < D-1$$

- Adding (K-D)(D-1) to both sides of the above equation, we get

$$(K-D)(D-2) = (K-D-q)(D-1) + r \quad \text{where} \quad 0 \leq r < D-1$$





# Huffman code

- There is an optimal D-ary prefix-free code for a random variable U with K possible values such that $D-r$ least likely codewords differ only in their last digit, where r is the remainder when $(K-D)(D-2)$ is divided by $D-1$.

  Sketch of proof:

# Huffman code

- There is an optimal D-ary prefix-free code for a random variable U with K possible values such that $D - r$ least likely codewords differ only in their last digit, where r is the remainder when $(K - D)(D - 2)$ is divided by $D - 1$.

  Sketch of proof:
- Arguments similar to binary case can be used to prove this.





# Huffman code

- Huffman coding: Algorithm for constructing D-ary (D $\geq$ 3) prefix-free codes for K-ary random variable U

# Huffman code

- Huffman coding: Algorithm for constructing D-ary (D $\geq$ 3) prefix-free codes for K-ary random variable U
- Step 0: Designate K vertices as $u_1, u_2, \cdots, u_K$ and assign probability $P_U(u_i)$ to vertex $u_i$. These vertices are designated as "active" nodes. Compute r as the remainder when $(K-D)(D-2)$ is divided by $D-1$.





# Huffman code

- Huffman coding: Algorithm for constructing D-ary (D $\geq$ 3) prefix-free codes for K-ary random variable U
- Step 0: Designate K vertices as $u_1, u_2, \cdots, u_K$ and assign probability $P_U(u_i)$ to vertex $u_i$. These vertices are designated as "active" nodes. Compute r as the remainder when $(K-D)(D-2)$ is divided by $D-1$.
- Step 1: Create a node that ties together the $D-r$ least likely active vertices with $D-r$ branches of a D-ary branch and assign probability equal to the sum of the probabilities of these $D-r$ vertices to this new node. Active this new node and deactive the $D-r$ vertices that are joined.

# Huffman code

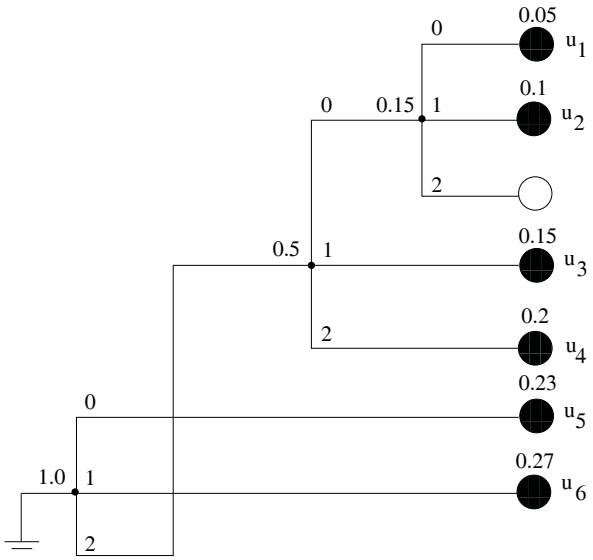
- Huffman coding: Algorithm for constructing D-ary (D $\geq$ 3) prefix-free codes for K-ary random variable U
- Step 0: Designate K vertices as $u_1, u_2, \cdots, u_K$ and assign probability $P_U(u_i)$ to vertex $u_i$. These vertices are designated as "active" nodes. Compute r as the remainder when $(K-D)(D-2)$ is divided by $D-1$.
- Step 1: Create a node that ties together the $D-r$ least likely active vertices with $D-r$ branches of a D-ary branch and assign probability equal to the sum of the probabilities of these $D-r$ vertices to this new node. Active this new node and deactive the $D-r$ vertices that are joined.
- If there is only one active vertex left. make it root and stop, otherwise set r=0 and go back to step 1.





# Huffman code

- Construct ternary Huffman code for the following example: $P(u_1) = 0.05, P(u_2) = 0.1, P(u_3) = 0.15, P(u_4) = 0.2, P(u_5) = 0.23, P(u_6) = 0.27$




| U | Z |
|---|---|
| $u_1$ | 200 |
| $u_2$ | 201 |
| $u_3$ | 21 |
| $u_4$ | 22 |
| $u_5$ | 0 |
| $u_6$ | 1 |

# Outline

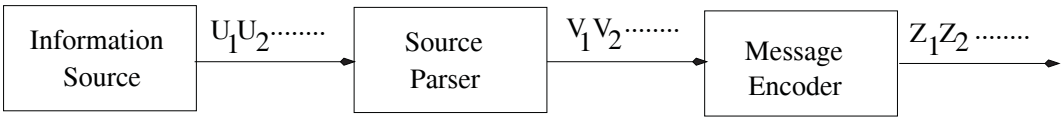






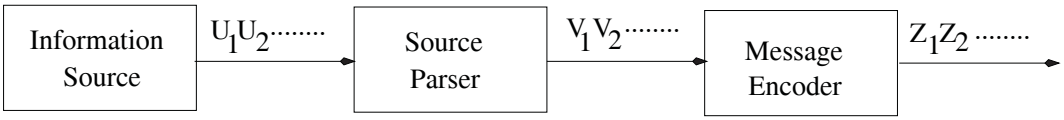
# Coding an Information source

- Parsing an information source



# Coding an Information source

- Parsing an information source



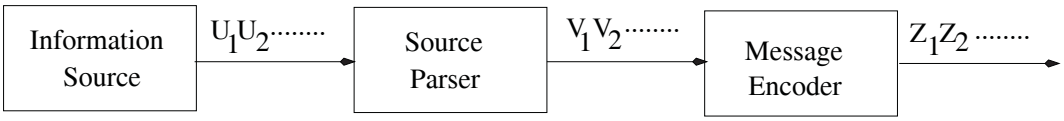

- The source parser divides the output sequences from the information source into messages that are encoded by the message encoder.





# Coding an Information source

- Parsing an information source




- The source parser divides the output sequences from the information source into messages that are encoded by the message encoder.
- We consider an L-block source parser, i.e.

$$\begin{aligned} V_1 &= [U_1, U_2, \cdots, U_L] \\ V_2 &= [U_{L+1}, U_{L+2}, \cdots, U_{2L}] \\ \vdots & \quad \vdots \end{aligned}$$

# Coding an Information source

- Block to variable length coding theorem for a DMS. There exists a D-ary prefix-free coding of a L-block message from DMS such that the average number of D-ary code digits per source satisfies

$$\frac{E[W]}{L} < \frac{H(U)}{\log D} + \frac{1}{L}$$

where H(U) is the uncertainty of a single source letter. Conversely for every D-ary prefix-free coding of an L-block message

$$\frac{E[W]}{L} \geq \frac{H(U)}{\log D}$$





# Coding an Information source

- Proof: Since the message $V = [U_1, U_2, \cdots, U_L]$ has L i.i.d. components, we have

$$\begin{aligned} H(V) &= H(U_1) + H(U_2) + \cdots + H(U_L) \\ &= LH(U) \end{aligned}$$

# Coding an Information source

- Proof: Since the message $V = [U_1, U_2, \cdots, U_L]$ has L i.i.d. components, we have

$$\begin{aligned} H(V) &= H(U_1) + H(U_2) + \cdots + H(U_L) \\ &= LH(U) \end{aligned}$$

- For any D-ary prefix-free code for V, we have

$$\begin{aligned} \frac{H(V)}{\log D} &\leq E[W] < \frac{H(V)}{\log D} + 1 \\ \implies \frac{LH(U)}{\log D} &\leq E[W] < \frac{LH(U)}{\log D} + 1 \\ \implies \frac{H(U)}{\log D} &\leq \frac{E[W]}{L} < \frac{H(U)}{\log D} + \frac{1}{L} \end{aligned}$$