

HUFFMAN CODES AND ENTROPY

Catherine Bénéteau, Caroline Haddad,
David Ruch and Patrick J. Van Fleet

Department of Mathematics & Statistics
University of South Florida
Tampa, FL USA

PREP - Wavelet Workshop, 2009



TODAY'S SCHEDULE

- 9:00-10:15 **Lecture Five:** Cumulative Energy, Quantization, and Peak Signal-to-Noise Ratio
- 10:15-10:30 Coffee Break (SCA 202)
- 10:30-11:45 ⇒ **Lecture Six:** Huffman Coding
- 12:00-1:00 Lunch
- 1:30-2:45 **Lecture Seven:** Putting it All Together: Image Compression
- 2:45-3:00 Coffee Break (SCA 202)
- 3:00-4:15 **Lecture Eight:** Daubechies Wavelet Transformations

OUTLINE

TODAY'S SCHEDULE

HUFFMAN CODING

Huffman Needs Help

A QUANTITATIVE MEASURE FOR CODING

Entropy

IN THE CLASSROOM

Teaching Ideas

Computer Usage

Student Difficulties



HUFFMAN CODING

- ▶ In 1952, David Huffman made a simple observation:

HUFFMAN CODING

- ▶ In 1952, **David Huffman** made a simple observation:
- ▶ *Rather than use the same number of bits to represent each character, why not use a short bit stream for characters that appear often in an image and a longer bit stream for characters that appear infrequently in the image?*

HUFFMAN CODING

- ▶ In 1952, **David Huffman** made a simple observation:
- ▶ *Rather than use the same number of bits to represent each character, why not use a short bit stream for characters that appear often in an image and a longer bit stream for characters that appear infrequently in the image?*
- ▶ He then developed an algorithm to do just that. We refer to his simple algorithm as **Huffman coding**. We will illustrate the algorithm via an example.

HUFFMAN CODING

- ▶ Suppose you want to perform Huffman coding on the word **seesaws**.

HUFFMAN CODING

- ▶ Suppose you want to perform Huffman coding on the word *seesaws*.
- ▶ First observe that *s* appears three times (24 bits), *e* appears twice (16 bits), and *a* and *w* each appear once (16 bits) so the total number of bits needed to represent *seesaws* is 56.

HUFFMAN CODING

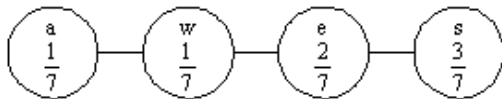
Char.	ASCII	Binary	Frequency
s	115	01110011 ₂	3
e	101	01100101 ₂	2
a	97	01100001 ₂	1
w	119	01110111 ₂	1

So in terms of bits, the word **seesaws** is

01110011 01100101 01100101 01110011 01100001 01110111 01110011

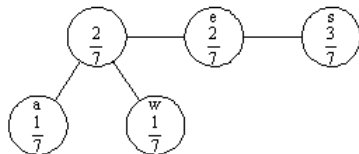
HUFFMAN CODING

The first step in Huffman coding is as follows: Assign probabilities to each character and then sort from smallest to largest. We will put the probabilities in circles called **nodes** and connect them with lines (**branches**).



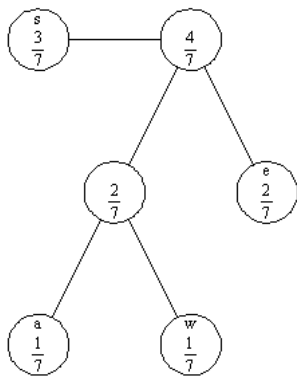
HUFFMAN CODING

Now simply add the two smallest probabilities to create a new node with probability $2/7$. Branch the two small nodes off this one and resort the three remaining nodes:



HUFFMAN CODING

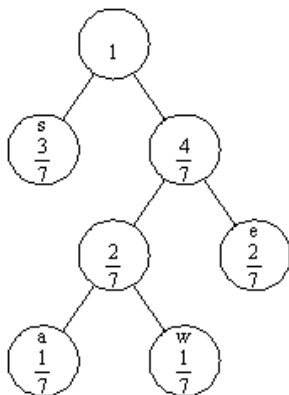
Again we add the smallest two probabilities on the top row ($2/7 + 2/7 = 4/7$), create a new node with everything below these nodes as branches and sort again:



HUFFMAN CODING

Since only two nodes remain on top, we simply add the probabilities of these nodes together to get 1 and obtain our finished tree:

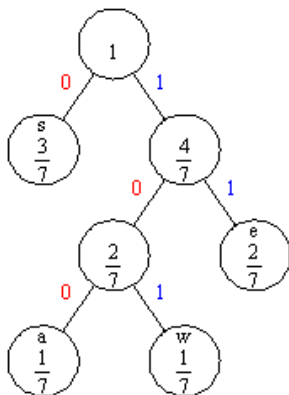
HUFFMAN CODING



HUFFMAN CODING

Now assign to each left branch the value 0 and to each right branch the value 1:

HUFFMAN CODING



HUFFMAN CODING

- ▶ We can read the new bit stream for each character right off the tree!

HUFFMAN CODING

- ▶ We can read the new bit stream for each character right off the tree!
- ▶ Here are the new bit streams for the four characters:

HUFFMAN CODING

Char.	Binary
s	0_2
e	11_2
a	100_2
w	101_2

HUFFMAN CODING

Char.	Binary
s	0_2
e	11_2
a	100_2
w	101_2

- ▶ Since *s* appears three times in *seesaws*, we need 3 bits to represent them. The character *e* appears twice (4 bits), and *a* and *w* each appear once (3 bits each).

HUFFMAN CODING

Char.	Binary
s	0 ₂
e	11 ₂
a	100 ₂
w	101 ₂

- ▶ Since *s* appears three times in *seesaws*, we need 3 bits to represent them. The character *e* appears twice (4 bits), and *a* and *w* each appear once (3 bits each).
- ▶ A total of 13 bits, for an average of $\frac{13}{7} = 1.86$ bits per character.

HUFFMAN CODING

- ▶ If the string were an image and the characters pixels, we would measure our compression in “bits per pixel”, in this case 1.86 bpp.

HUFFMAN CODING

- ▶ If the string were an image and the characters pixels, we would measure our compression in “bits per pixel”, in this case 1.86 bpp.
- ▶ Recall without Huffman coding, we needed 56 bits so we have reduced the number of bits needed by a factor of 4.3!

HUFFMAN CODING

- ▶ If the string were an image and the characters pixels, we would measure our compression in “bits per pixel”, in this case 1.86 bpp.
- ▶ Recall without Huffman coding, we needed 56 bits so we have reduced the number of bits needed by a factor of 4.3!
- ▶ Here is the word *seesaws* using the Huffman codes for each character:

0 11 11 0 100 101 0

HUFFMAN CODING

- ▶ If the string were an image and the characters pixels, we would measure our compression in “bits per pixel”, in this case 1.86 bpp.
- ▶ Recall without Huffman coding, we needed 56 bits so we have reduced the number of bits needed by a factor of 4.3!
- ▶ Here is the word *seesaws* using the Huffman codes for each character:

0 11 11 0 100 101 0

- ▶ Can you see how to decode the message given the Huffman codes/tree?

HUFFMAN CODING

- ▶ Huffman coding is invertible. You can decode a message by looking at each incoming digit in sequence. At each step either the digits you have seen do not yet correspond to any character, or they correspond to exactly one.

HUFFMAN CODING

- ▶ Huffman coding is invertible. You can decode a message by looking at each incoming digit in sequence. At each step either the digits you have seen do not yet correspond to any character, or they correspond to exactly one.
- ▶ Get with a partner for the following challenges. **First:** Determine the Huffman codes for each letter of the string 'abacadac'. Write the word as a binary string using the Huffman codes.

HUFFMAN CODING

- ▶ Huffman coding is invertible. You can decode a message by looking at each incoming digit in sequence. At each step either the digits you have seen do not yet correspond to any character, or they correspond to exactly one.
- ▶ Get with a partner for the following challenges. **First:** Determine the Huffman codes for each letter of the string 'abacadac'. Write the word as a binary string using the Huffman codes.
- ▶ **Second:** given the codes $f = 0$, $h = 110$, $p = 111$, $u = 10$, draw the Huffman code tree. Use the codes tree to decode the bit stream 11010001111000.

HUFFMAN CODING

- ▶ Huffman coding is invertible. You can decode a message by looking at each incoming digit in sequence. At each step either the digits you have seen do not yet correspond to any character, or they correspond to exactly one.
- ▶ Get with a partner for the following challenges. **First:** Determine the Huffman codes for each letter of the string 'abacadac'. Write the word as a binary string using the Huffman codes.
- ▶ **Second:** given the codes $f = 0$, $h = 110$, $p = 111$, $u = 10$, draw the Huffman code tree. Use the codes tree to decode the bit stream 11010001111000.
- ▶ What sort of savings does the coding give you?



ENCODING AN IMAGE

- ▶ The examples are a bit of a sales job - of course we will enjoy great savings with only 4 distinct characters. What happens when we apply Huffman coding to a digital image?
- ▶ Consider the 200×200 image



ENCODING AN IMAGE

- ▶ Unencoded, we need 320000 bits or 8 *bits per pixel* (bpp) to represent the image.

ENCODING AN IMAGE

- ▶ Unencoded, we need 320000 bits or 8 *bits per pixel* (bpp) to represent the image.
- ▶ With Huffman encoding, we need 266993 bits to store the image.

ENCODING AN IMAGE

- ▶ Unencoded, we need 320000 bits or 8 *bits per pixel* (bpp) to represent the image.
- ▶ With Huffman encoding, we need 266993 bits to store the image.
- ▶ This constitutes a 16.5% savings or an average of 6.67 bpp.

ENCODING AN IMAGE

- ▶ Unencoded, we need 320000 bits or 8 *bits per pixel* (bpp) to represent the image.
- ▶ With Huffman encoding, we need 266993 bits to store the image.
- ▶ This constitutes a 16.5% savings or an average of 6.67 bpp.
- ▶ We should be able to do better. The best possible compression measured in bpp is given by the *entropy* of the image. Next topic!

ENCODING AN IMAGE

- ▶ Unencoded, we need 320000 bits or 8 *bits per pixel* (bpp) to represent the image.
- ▶ With Huffman encoding, we need 266993 bits to store the image.
- ▶ This constitutes a 16.5% savings or an average of 6.67 bpp.
- ▶ We should be able to do better. The best possible compression measured in bpp is given by the *entropy* of the image. Next topic!
- ▶ What really helps an encoding method is a *preprocessor* that transforms the image to a setting that is a bit more amenable to the encoding scheme.

ENCODING AN IMAGE

- ▶ Unencoded, we need 320000 bits or 8 *bits per pixel* (bpp) to represent the image.
- ▶ With Huffman encoding, we need 266993 bits to store the image.
- ▶ This constitutes a 16.5% savings or an average of 6.67 bpp.
- ▶ We should be able to do better. The best possible compression measured in bpp is given by the *entropy* of the image. Next topic!
- ▶ What really helps an encoding method is a *preprocessor* that transforms the image to a setting that is a bit more amenable to the encoding scheme.
- ▶ That's where the discrete wavelet transform comes in!

QUANTITATIVE MEASUREMENTS REDUX

ENTROPY

- ▶ **Entropy** is used to assess the effectiveness of compression algorithms.

QUANTITATIVE MEASUREMENTS REDUX

ENTROPY

- ▶ **Entropy** is used to assess the effectiveness of compression algorithms.
- ▶ Claude Shannon, in his landmark 1948 paper, showed that the best we can hope for when compressing data via some *lossless* scheme S is to encode the output of S where the average number of bits per character is the entropy of S .

QUANTITATIVE MEASUREMENTS REDUX

ENTROPY

- ▶ Let $\mathbf{v} = (v_1, \dots, v_n) \in \mathbb{R}^n$ and let $I \subseteq \{1, \dots, n\}$ denote the set of indices that correspond to the distinct elements in \mathbf{v} . That is $j, k \in I$ if and only if $v_j \neq v_k$. For $k \in I$, let $p(v_k)$ denote the relative frequency of v_k in \mathbf{v} . We define the *entropy* of \mathbf{v} by

$$Ent(\mathbf{v}) = \sum_{k \in I} p(v_k) \log_2(1/p(v_k)) = - \sum_{k \in I} p(v_k) \log_2(p(v_k))$$

QUANTITATIVE MEASUREMENTS REDUX

ENTROPY

- ▶ Let $\mathbf{v} = (v_1, \dots, v_n) \in \mathbb{R}^n$ and let $I \subseteq \{1, \dots, n\}$ denote the set of indices that correspond to the distinct elements in \mathbf{v} . That is $j, k \in I$ if and only if $v_j \neq v_k$. For $k \in I$, let $p(v_k)$ denote the relative frequency of v_k in \mathbf{v} . We define the *entropy* of \mathbf{v} by

$$\text{Ent}(\mathbf{v}) = \sum_{k \in I} p(v_k) \log_2(1/p(v_k)) = - \sum_{k \in I} p(v_k) \log_2(p(v_k))$$

- ▶ A weighted average of the optimal number of bits needed to represent each v_k .

QUANTITATIVE MEASUREMENTS REDUX

ENTROPY

- ▶ Let $\mathbf{v} = (v_1, \dots, v_n) \in \mathbb{R}^n$ and let $I \subseteq \{1, \dots, n\}$ denote the set of indices that correspond to the distinct elements in \mathbf{v} . That is $j, k \in I$ if and only if $v_j \neq v_k$. For $k \in I$, let $p(v_k)$ denote the relative frequency of v_k in \mathbf{v} . We define the *entropy* of \mathbf{v} by

$$\text{Ent}(\mathbf{v}) = \sum_{k \in I} p(v_k) \log_2(1/p(v_k)) = - \sum_{k \in I} p(v_k) \log_2(p(v_k))$$

- ▶ A weighted average of the optimal number of bits needed to represent each v_k .
- ▶ For example, if half the elements are identical, say v_{47} , then ideally each of these elements needs $\log_2(1/p(v_{47})) = 1$ bit.

QUANTITATIVE MEASUREMENTS REDUX

ENTROPY

- ▶ Two extreme cases:

QUANTITATIVE MEASUREMENTS REDUX

ENTROPY

- ▶ Two extreme cases:
- ▶ If all vector elements are **identical**, then $I = \{1\}$, $p(v_k) = 1$ and $\log_2(p(v_k)) = 0$. So $Ent(\mathbf{v}) = 0$.

QUANTITATIVE MEASUREMENTS REDUX

ENTROPY

- ▶ Two extreme cases:
- ▶ If all vector elements are **identical**, then $I = \{1\}$, $p(v_k) = 1$ and $\log_2(p(v_k)) = 0$. So $Ent(\mathbf{v}) = 0$.
- ▶ If each vector element is **distinct**, $I = \{1, \dots, n\}$, then $p(v_k) = 1/n$ and

$$Ent(\mathbf{v}) = - \sum_{k=1}^n \frac{1}{n} \log_2(1/n) = \log_2(n)$$

QUANTITATIVE MEASUREMENTS REDUX

ENTROPY

- ▶ It can be shown that $0 \leq Ent(\mathbf{v}) \leq \log_2(n)$ for all $\mathbf{v} \in \mathbb{R}^n$.

QUANTITATIVE MEASUREMENTS REDUX

ENTROPY

- ▶ It can be shown that $0 \leq Ent(\mathbf{v}) \leq \log_2(n)$ for all $\mathbf{v} \in \mathbb{R}^n$.
- ▶ Suppose an image consists of four distinct intensities with relative frequencies $\frac{1}{8}$, $\frac{1}{8}$, $\frac{1}{4}$, and $\frac{1}{2}$. What is the entropy of the image?

QUANTITATIVE MEASUREMENTS REDUX

ENTROPY

- ▶ It can be shown that $0 \leq Ent(\mathbf{v}) \leq \log_2(n)$ for all $\mathbf{v} \in \mathbb{R}^n$.
- ▶ Suppose an image consists of four distinct intensities with relative frequencies $\frac{1}{8}, \frac{1}{8}, \frac{1}{4}$, and $\frac{1}{2}$. What is the entropy of the image?
- ▶ The entropy is:

$$\begin{aligned} Ent(\mathbf{v}) &= 2 \cdot \frac{1}{8} \log_2(8) + \frac{1}{4} \log_2(4) + \frac{1}{2} \log_2(2) \\ &= \frac{3}{4} + \frac{2}{4} + \frac{1}{2} \\ &= 1.75 \end{aligned}$$

QUANTITATIVE MEASUREMENTS REDUX

ENTROPY

- ▶ What about the bbp of this hypothetical image using Huffman coding? *Hint: A previous example.*

QUANTITATIVE MEASUREMENTS REDUX

ENTROPY

- ▶ What about the bbp of this hypothetical image using Huffman coding? *Hint: A previous example.*
- ▶ What is the entropy of an image, say 16×16 , with 256 distinct grayscale intensities?

IN THE CLASSROOM

TEACHING IDEAS

- ▶ The students really enjoy the material on digital images. There is information in the book on histogram equalization and other color spaces. Sometimes, we give extra credit assignments on these topics.

IN THE CLASSROOM

TEACHING IDEAS

- ▶ The students really enjoy the material on digital images. There is information in the book on histogram equalization and other color spaces. Sometimes, we give extra credit assignments on these topics.
- ▶ It takes the student a while to grasp the algorithm for cumulative energy and the concept of entropy. We usually do several examples here. PSNR doesn't show up as often as the other two, so we tend to save those examples for the compression applications.

IN THE CLASSROOM

TEACHING IDEAS

- ▶ The students grasp Huffman coding right away. We have been happy at how well they understand how to decode as well.

IN THE CLASSROOM

COMPUTER USAGE

- ▶ We make sure to have the students write a module for Cumulative Energy and usually we have them write the Entropy module as well. While we won't cover the process in this workshop, we have a template for `DiscreteWavelets` that allows them to “fill in” their code.

IN THE CLASSROOM

COMPUTER USAGE

- ▶ We make sure to have the students write a module for Cumulative Energy and usually we have them write the Entropy module as well. While we won't cover the process in this workshop, we have a template for `DiscreteWavelets` that allows them to “fill in” their code.
- ▶ The Huffman code module is very difficult to write. We have a student-written version and it is impossibly slow. We usually assign the Huffman coding module as extra credit.

IN THE CLASSROOM

STUDENT DIFFICULTIES

- ▶ Most of the difficulties here are with student understanding of what Cumulative Energy and Entropy do. We often ask them to design vectors that will produce a certain “shape” for Cumulative Energy or a certain value for Entropy.

TODAY'S SCHEDULE

9:00-10:15 **Lecture Five:** Cumulative Energy, Quantization, and Peak Signal-to-Noise Ratio

10:15-10:30 Coffee Break (SCA 202)

10:30-11:45 **Lecture Six:** Huffman Coding

12:00-1:00 ⇒ Lunch

1:30-2:45 **Lecture Seven:** Putting it All Together: Image Compression

2:45-3:00 Coffee Break (SCA 202)

3:00-4:15 **Lecture Eight:** Daubechies Wavelet Transformations

