COMPSCI 650 Applied Information Theory Jan 28, 2016

Lecture 4

Instructor: Arya Mazumdar *Scribe:* [redacted]

English keyboard has 96 keys. To encode it into a binary representation with fix length (such as ASCII code), we need 7 bits for each character:

$$\lceil log_2 96 \rceil = 7 \text{bits / char.} \tag{1}$$

Since the appearing probability of each English symbol $P(a), ...P(b), ...P(" "), ..., P(;)$ is not uniform, we should be able to reduce the number of required bits. Based on the probability of each English symbol, we can compute the entropy $H(E^1) \cong 4.5$ bits / char. If we use Huffman coding taught in this lecture to encode English keyboard, then we only need around 4.7 bits / char.

Furthermore, the characters in English are not independent. For example, the number of meaningful English strings with length 8 should be much smaller than 96^8 and some strings occur much often than the others. In fact, we can get the entropy of such string $H(E^8) \cong 2.4$ bits/char. If we further extend the length of string to infinity, the estimated entropy of English would be $H(E^\infty) \cong 1.3$ bits / char.

Using lossless data compression algorithms to encode English keyboard results in the following:

$LZW \cong 3.7$ bits / char

$GZip \cong 2.7$ bits / char

$BW \cong 1.89$ bits / char

where LZW will be taught in the future lecture, and BW is an industrial standard, which achieves good performance by combining LZW, Gzip and several other compression algorithms.

Nowadays, the cost of storage becomes much cheaper, but the internet bandwidth still remains an expensive resource. This is one of reasons why the compression techniques still play important roles in our life.

1 Recall: Uniquely Decodable Codes

Given $\mathcal{X} = \{1, 2, 3, 4\}$,

C(1) = 00, C(2) = 10, C(3) = 11, and C(4) = 110.

The code is uniquely decodable, but you have to look forward because C(3) is a prefix of C(4).

As we discussed in the previous lecture, uniquely decodable code satisfies Kraft inequaltiy:

$$\sum_{x \in \mathcal{X}} 2^{-l(x)} \leq 1 \tag{2}$$

From the derivation of $L(C) \geq H(X)$, we know that $l(x)$ should be close to $\log_2 \frac{1}{p(x)}$.

2 Instantaneous Code

This type of codes is also called Prefix-free code or Prefix code, because no code word would be a prefix of another code word. For example,

$\mathcal{X} = \{1, 2, 3, 4\}$

C(1) = 0, C(2) = 10, C(3) = 110, and C(4) = 111 is an instantaneous code.

If we know that $p(1) = p(2) = p(3) = p(4) = 1/4$, $H(X) = \log_2 4 = 2$ bits.

$L(C) = \frac{1}{4} + 1 + \frac{1}{4} \cdot 2 + \frac{1}{4} \cdot 3 + \frac{1}{4} \cdot 3 = 2.25$ bits

In this case, the probability distribution of each symbol is uniform, so it is incompressible. In order to have coding with $H(X) = 2$ bits, you should consider the following fix-length code:

$C'(1) = 00$, $C'(2) = 10$, $C'(3) = 01$, and $C'(4) = 11$.

Notice that the code is an instantaneous code. Since an instantaneous code implies an uniquely decodable code, it is also an uniquely decodable code.

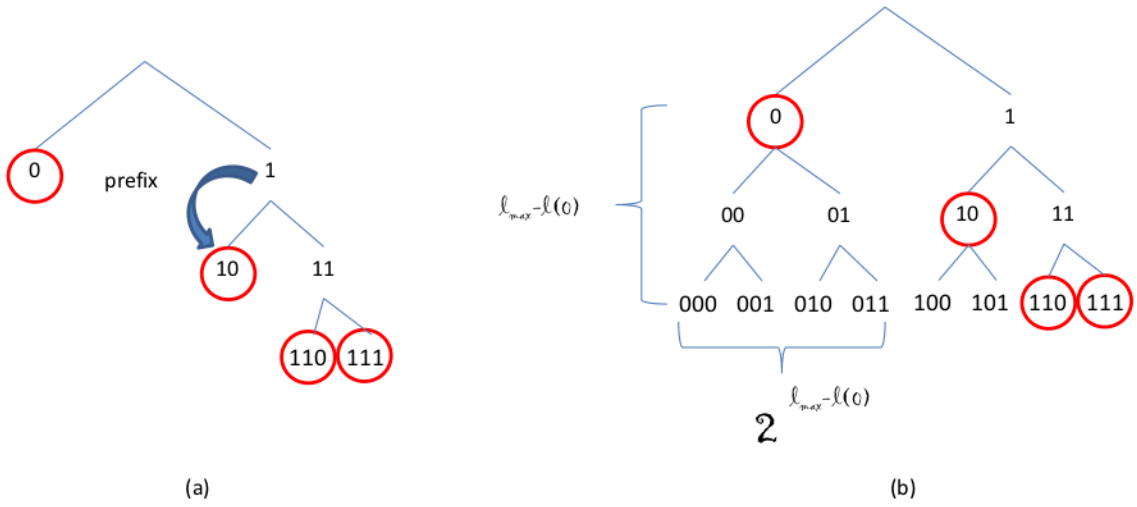

Figure 1: Huffman tree in example 1

3 Huffman Tree

3.1 Example 1

Using the same coding in the previous example, we encode 1 to 4 as C(1) = 0, C(2) = 10, C(3) = 110, and C(4) = 111. We can draw a binary tree to check whether it is a prefix-free code, and this tree is called Huffman tree. As we can see in Figure 1(a), the parent of a node is a prefix of the code corresponding to the node.

From Huffman tree, we can prove Kraft inequality using a different way. First, we define $l_{max} = max_{x \in X} l(x)$, which is also the height of Huffman tree. From Figure 1(b), we can see that the number of leaves in the left subtree is $2^{l_{max}-l(0)}$. If we sum number of leaves in all subtrees bellow the encoded nodes $\sum_{x \in \mathcal{X}} 2^{l_{max}-l(x)}$, this number will smaller or equal to the number of leaves in the whole tree $2^{l_{max}}$. Therefore, we get

$$\sum_{x \in \mathcal{X}} 2^{l_{max}-l(x)} \leq 2^{l_{max}}$$
$$\Rightarrow \sum_{x \in \mathcal{X}} 2^{-l(x)} \leq 1. \quad (3)$$

3.2 Example 2

In this example, we will show how to construct a instantaneous code from a set of coding lengths which satisfy Kraft inequality. Given l(1) = 1, l(2), = 2, l(3) = 3, and l(4) = 3, we first verify that the coding length satisfies Kraft inequality: $2^{-1} + 2^{-2} + 2^{-3} + 2^{-3} = 1$. Then, we encode the symbol with lowest length using the rightmost node with such length, and delete the subtree bellow the node. From the remaining tree, we repeat the above process till all symbols are encoded as shown in Figure 2(a). The codes are C(1) = 1, C(2) = 01, C(3) = 000, and C(4) = 001.

3.3 Example 3

In the previous example, all leaves in Huffman tree are either chosen or deleted. This is because $\sum_{x \in \mathcal{X}} 2^{-l(x)} = 1$ in that example. To show the other cases, we let l(1) = 1, l(2) = 2, l(3) = 3, and

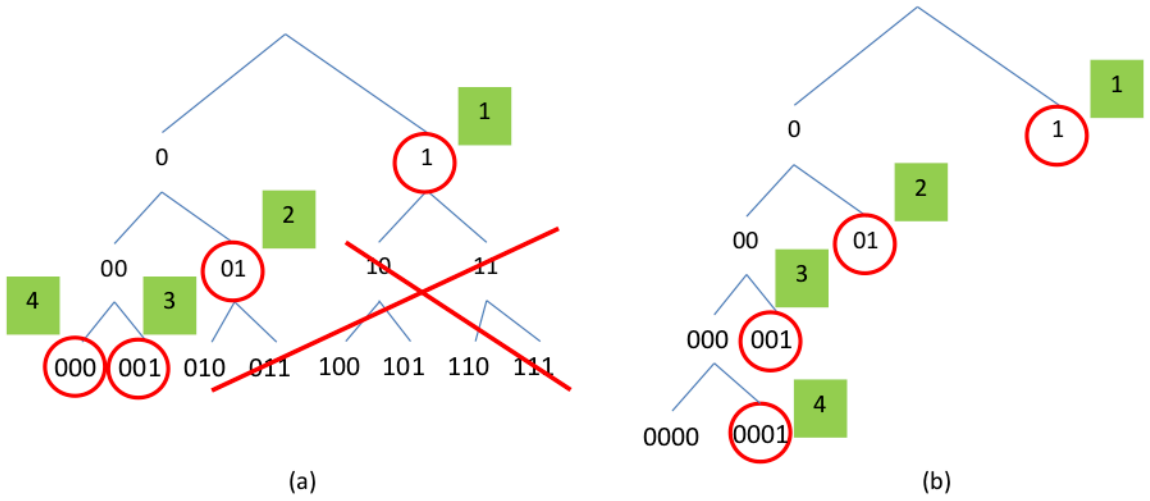

Figure 2: Construction of Huffman tree based on length of codes

l(4) = 4. Following the construction procedure described in the previous sample, we can assign a code to each symbol as shown in Figure 2(b). Since there is a leaf which is not used, the code is clearly not optimal.

4 Shannon Coding

Shannon Coding, which is also known as Shannon-Fano coding, is a way to construct an instantaneous coding based on the probability distribution of symbols. At the end of lecture 3, We have shown that an optimal coding should make the coding length of each symbol $l(x)$ as close to $\log(\frac{1}{p(x)})$ as possible, while keeping $l(x)$ as an integer. To solve the problem, Shannon Coding simply lets

$$l(x) = \left\lceil \log(\frac{1}{p(x)}) \right\rceil \forall x \in \mathcal{X}. \tag{4}$$

We can easily show that the length assignment in (4) satisfies Kraft Inequality by

$$\begin{aligned} \sum_{x\in\mathcal{X}} 2^{-l(x)} &= \sum_{x\in\mathcal{X}} 2^{-\left\lceil \log_2(\frac{1}{p(x)}) \right\rceil} \\ &\leq \sum_{x\in\mathcal{X}} 2^{-\log_2(\frac{1}{p(x)})} = \sum_{x\in\mathcal{X}} 2^{\log_2(p(x))} = \sum_{x\in\mathcal{X}} p(x) = 1. \end{aligned} \tag{5}$$

In the previous section, we have shown that an instantaneous coding can be constructed if the length of its code words $l(x)$ satisfy Kraft inequality. Thus, once we can map each symbol x which occurs with probability $p(x)$ to $l(x)$ using (4), the Shannon code can be constructed using the mentioned procedure.

Intuitively speaking, Shannon coding is close to optimal because the largest difference between $l(x)$ and $\log(\frac{1}{p(x)})$ is 1. This could also be shown by deriving the upper bound of expected length of coding

$$\begin{aligned} L_{\text{Shannon}}(C) &= \sum_{x\in\mathcal{X}} p(x)l(x) = \sum_{x\in\mathcal{X}} p(x) \left\lceil \log_2(\frac{1}{p(x)}) \right\rceil \\ &< \sum_{x\in\mathcal{X}} p(x)(\log_2(\frac{1}{p(x)}) + 1) = H(X) + 1 \end{aligned} \tag{6}$$

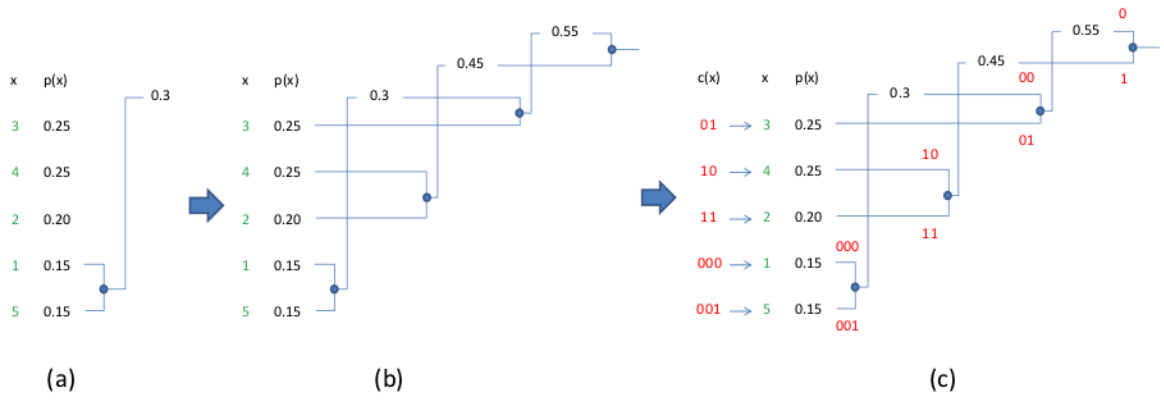

Figure 3: The procedure of constructing Huffman code

At the end of lecture 3, we also knows that the length of optimal coding $L_{\text{Shannon}}(x)$ cannot lower than the entropy of the system. This leads to

$$H(x) \leq L_{\text{Shannon}}(C) < H(X) + 1. \tag{7}$$

5 Huffman Coding

Huffman Coding is a type of optimal instantaneous coding, which starts from a class project. We will use two examples to example the compression process of Huffman coding.

5.1 Example 1

Assuming $\mathcal{X} = \{1, 2, 3, 4, 5\}$, $p(1) = 0.15$, $p(2) = 0.2$, $p(3) = 0.25$, $p(4) = 0.25$, and $p(5) = 0.15$, our goal is to perform Huffman coding on this data source. The first step we need to do is sorting every symbol x according to its appearing probability $p(x)$. As we can see, one valid order is $3, 4, 2, 1, 5$ because $p(3) = p(4) > p(2) > p(1) = p(5)$.

We layout the symbols and their probability according to the order. Then, we combine two symbols with lowest probability at the bottom into a new symbol, add the probability of combined symbols as the probability of the new symbol, and sort the new symbol list according to their probability again as shown in Figure 3(a). The process repeats till all symbols are merged into the final new symbol as shown in Figure 3(b).

Finally, we assign the code by viewing the linkage in merging process as a Huffman tree. Starting from the root created by the final merge, we label 0 and 1 to the upper and lower branch, respectively. Next, we append 0 and 1 to the existing labels whenever encountering new branches. To simplify the process, we always append 0 to the upper branch and 1 to the lower branch. The process continues till every original symbol is assigned a code as shown in Figure 3(c).

To show the efficiency of the code, we compare the average length of Huffman code and entropy of the system.

$$\begin{aligned} L_{\text{Huffman}}(C) &= 0.25 \cdot 2 + 0.25 \cdot 2 + 0.2 \cdot 2 + 0.15 \cdot 3 + 0.15 \cdot 3 \\ &= 2 \cdot 0.7 + 3 \cdot 0.3 = 2.3 \end{aligned} \tag{8}$$

$$H(X) = -0.25 \log 0.25 - 0.25 \log 0.25 - 0.2 \log 0.2 - 0.15 \log 0.15 - 0.15 \log 0.15 \cong 2.2855 \tag{9}$$

HW: Constructing Shannon code and compare its efficiency with Huffman code.

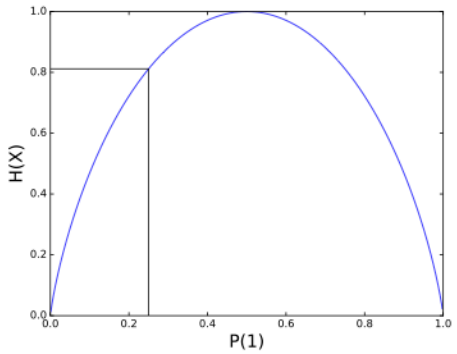

Figure 4: The entropy of random variable X in example 2

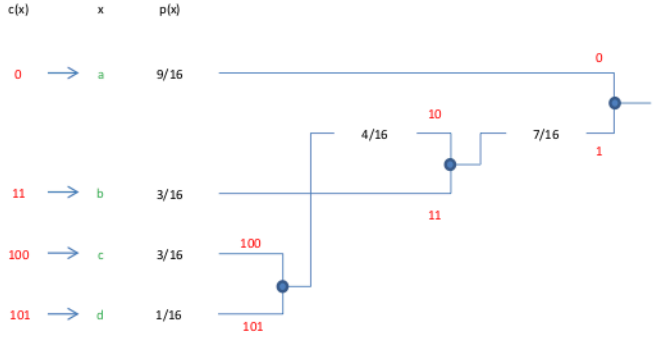

Figure 5: The construction of Huffman code in the example 2

5.2 Example 2

Assuming $\mathcal{X} = \{0, 1\}$, $p(0) = \frac{3}{4}$, and $p(1) = \frac{1}{4}$, we can easily compute its entropy $H(X) = -p(0)\log_2 p(0) - p(1)\log_2 p(1) \cong 0.81$ as shown in Figure 4. From the figure, we can see that if $p(1) = \frac{1}{2}$, the system is incompressible because it requires 1 bit to convey the information of each symbol. However, our $p(1)$ is equal to $\frac{1}{4}$, but we still use 1 bit to represent each symbol. How could we do better?

As we mentioned at the beginning of this lecture, we can compress English words more efficiently by handling more symbols at a time. Similarly, we can combine more symbols in the transmission sequence into a new set of symbols here. For example, there are four possibilities for any two consecutive symbols in this example, and we use four new symbol to represent them: $a = 00, b = 01, c = 10$, and $d = 11$. This implies $p(a) = p(0) \cdot p(0) = \frac{9}{16}$, $p(b) = p(0) \cdot p(1) = \frac{3}{16}$, $p(c) = p(1) \cdot p(0) = \frac{3}{16}$, and $p(d) = p(1) \cdot p(1) = \frac{1}{16}$. Then, we can apply Huffman coding in this situation as shown in Figure 5.

To show how to perform encoding process, we present a sequence of input symbols and encode them as following:

$$\begin{array}{lcccccccccccc} x: & \underline{01} & \underline{01} & \underline{01} & \underline{10} & \underline{11} & \underline{10} & \underline{10} & \underline{10} & \underline{11} & \underline{10} & \underline{11} & \underline{11} \\ x': & b & b & b & c & d & c & c & c & d & c & d & d \\ c(x'): & \underline{11} & \underline{11} & \underline{11} & \underline{100} & \underline{101} & \underline{100} & \underline{100} & \underline{100} & \underline{101} & \underline{100} & \underline{101} & \underline{101}. \end{array}$$

In the input sequence, we can see that the length of our encoding is actually longer than the length of original sequence. The reason is that there are much more 1 than 0 in the sequence but we assume that $p(1) > p(0)$. This tells us that any coding based on $p(x)$ only works well when the observed input sequence actually follows $p(x)$.

Finally, like we did in example 1, we can compute the efficient of coding in this example as following:

$$L_{\text{Huffman}}(C) = 1 \cdot \frac{9}{16} + 2 \cdot \frac{3}{16} + 3 \cdot \frac{3}{16} + 3 \cdot \frac{1}{16} = \frac{27}{16} \cong 1.68 \tag{10}$$

Since we compress two input symbols at a time, the average length of representing one symbol is

$$\widetilde{L}_{\text{Huffman}}(C) = \frac{L_{\text{Huffman}}(C)}{2} \cong 0.84. \tag{11}$$

This result is much closer to the entropy of the system ($H(X) \cong 0.81$) than using 1 bit to represent each of the original symbols.