Elements of Information Theory

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 500

Subsection 1

Examples of Codes

Source Code and Length of a Codeword

Definition

A **source code** C for a random variable X is a mapping

$$C : \mathcal{X} \to \mathcal{D}^*$$

from $\mathcal{X}$, the range of X, to $\mathcal{D}^*$, the set of finite-length strings of symbols from a D-ary alphabet $\mathcal{D}$.
Let $C(x)$ denote the codeword corresponding to x.
Let $\ell(x)$ denote the length of $C(x)$.

Example: Let $\mathcal{X} = \{\text{red}, \text{blue}\}$ and $\mathcal{D} = \{0, 1\}$.

Then $C(\text{red}) = 00$, $C(\text{blue}) = 11$ is a source code for $\mathcal{X}$ with alphabet $\mathcal{D}$.

Expected Length

Definition

The **expected length** $L(C)$ of a source code $C(x)$ for a random variable X with probability mass function $p(x)$ is given by

$$L(C) = \sum_{x \in \mathcal{X}} p(x)\ell(x),$$

where $\ell(x)$ is the length of the codeword associated with x.

- Without loss of generality, we can assume that the D-ary alphabet is

$$\mathcal{D} = \{0, 1, \ldots, D-1\}.$$

Example

- Let X be a random variable with the following distribution and codeword assignment:
$$\begin{array}{ll} \Pr(X=1)=\frac{1}{2}, & C(1)=0, \\ \Pr(X=2)=\frac{1}{4}, & C(2)=10, \\ \Pr(X=3)=\frac{1}{8}, & C(3)=110, \\ \Pr(X=4)=\frac{1}{8}, & C(4)=111. \end{array}$$
The entropy of X is $H(X)=\frac{1}{2}\log\frac{1}{2}+\frac{1}{4}\log\frac{1}{4}+2\frac{1}{8}\log\frac{1}{8}=1.75$ bits.
The expected length of C is
$$L(C)=E\ell(X)=\frac{1}{2}\cdot 1+\frac{1}{4}\cdot 2+2\frac{1}{8}\cdot 3=1.75 \text{ bits.}$$
Here we have a code that has the same average length as the entropy.
- We note that any sequence of bits can be uniquely decoded into a sequence of symbols of $\mathcal{X}$.
For example, the bit string 0110111100110 is decoded as 134213.

Example

- Consider another simple example of a code for a random variable:

$$\begin{array}{ll} \Pr(X=1)=\frac{1}{3}, & C(1)=0, \\ \Pr(X=2)=\frac{1}{3}, & C(2)=10, \\ \Pr(X=3)=\frac{1}{3}, & C(3)=11. \end{array}$$

  The code is uniquely decodable.

  In this case:

  - The entropy is $H(X)=\log 3=1.58$ bits.
  - The average length of C is $L(C)=\frac{1}{3}\cdot 1+2\frac{1}{3}\cdot 2=1.66$ bits.

  So we have $E\ell(X)>H(X)$.

Morse Code

- The Morse code is a reasonably efficient code for the English alphabet using an alphabet of four symbols:
    - A dot;
    - A dash;
    - A letter space;
    - A word space.
- Short sequences represent frequent letters (e.g., a single dot represents E).
- Long sequences represent infrequent letters (e.g., Q is represented by "dash,dash,dot,dash").
- This is not the optimal representation for the alphabet in four symbols.
- Many possible codewords are not utilized because:
    - The codewords for letters do not contain spaces except for a letter space at the end of every codeword;
    - No space can follow another space.

Nonsingular Codes

- Let x^n denote $(x_1, x_2, \ldots, x_n)$.

Definition

A code is said to be **nonsingular** if every element of the range of X maps into a different string in $\mathcal{D}^*$, i.e.,

$$x \neq x' \Rightarrow C(x) \neq C(x').$$

- Nonsingularity suffices for an unambiguous description of a single value of X.
- But we usually wish to send a sequence of values of X.
- In such cases we can ensure decodability by adding a special symbol (a "comma") between any two codewords.
- This is an inefficient use of the special symbol.
- To improve, we develop selfpunctuating or instantaneous codes.

Extension

Definition

The **extension** C^* of a code C is the mapping from finite length strings of X to finite-length strings of $\mathcal{D}$, defined by

$$C^*(x_1 x_2 \cdots x_n) = C(x_1)C(x_2)\cdots C(x_n),$$

where $C(x_1)C(x_2)\cdots C(x_n)$ indicates concatenation of the corresponding codewords.

Example: Suppose $C(x_1) = 00$ and $C(x_2) = 11$.

Then we have

$$C^*(x_1 x_2) = 0011.$$

Unique Decodability

Definition

A code is called **uniquely decodable** if its extension is nonsingular.

- The advantage is that any encoded string in a uniquely decodable code has only one possible source string producing it.
- The drawback is that one may have to look at the entire string to determine even the first symbol in the corresponding source string.

Prefix Codes

Definition

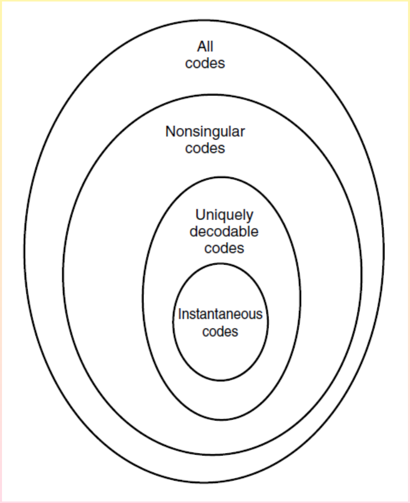
A code is called a **prefix code** or an **instantaneous code** if no codeword is a prefix of any other codeword.

- An instantaneous code can be decoded without reference to future codewords, since the end of a codeword is immediately recognizable.
- Hence, for an instantaneous code, the symbol x_i can be decoded as soon as we come to the end of the codeword corresponding to it.
- There is no need wait to see the codewords that come later.
- An instantaneous code is a **selfpunctuating code**, since one can look down the sequence of code symbols and add the commas to separate the codewords, without looking at later symbols.

  Example: The binary string 0101111111010 produced by the code of the previous example is parsed as $0, 10, 111, 110, 10$.

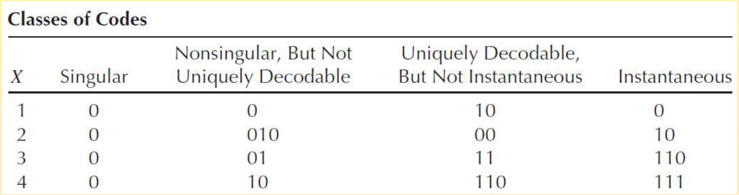
Hierarchy of Classes of Codes

- The nesting of the definitions is shown in the figure:



Differences Between Classes of Codes

- Consider the examples of codeword assignments $C(x)$ to $x \in \mathcal{X}$:

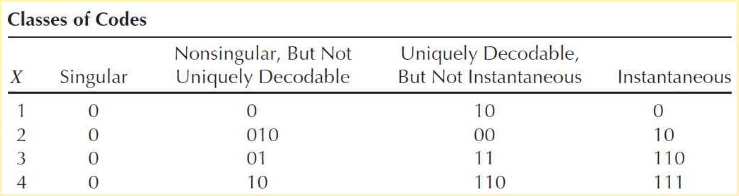


Classes of Codes

X	Singular	Nonsingular, But Not Uniquely Decodable	Uniquely Decodable, But Not Instantaneous	Instantaneous
1	0	0	10	0
2	0	010	00	10
3	0	01	11	110
4	0	10	110	111

- For the nonsingular code, the code string 010 has three possible source sequences: 2 or 14 or 31. Hence, the code is not uniquely decodable.
- The uniquely decodable code is not prefix-free and hence is not instantaneous.
- The fact that the last code is instantaneous is obvious, since no codeword is a prefix of any other.

Showing Unique Decodability



Classes of Codes

X	Singular	Nonsingular, But Not Uniquely Decodable	Uniquely Decodable, But Not Instantaneous	Instantaneous
1	0	0	10	0
2	0	010	00	10
3	0	01	11	110
4	0	10	110	111

- We show uniquely decodability of the third code.

  Take any code string and start from the beginning.
  - If the first two bits are 00 or 10, they can be decoded immediately.
  - If the first two bits are 11, we must look at the following bits.
    - If the next bit is a 1, the first source symbol is a 3;
    - If the length of the string of 0's immediately following the 11 is odd, the first codeword must be 110 and the first source symbol must be 4;
    - If the length of the string of 0's is even, the first source symbol is a 3.

  By repeating this argument, we can see that this code is uniquely decodable.

Subsection 2

Kraft Inequality

Kraft Inequality

Theorem

For any instantaneous code (prefix code) over an alphabet of size D, the codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ must satisfy the inequality

$$\sum_i D^{-\ell_i} \leq 1.$$

Conversely, given a set of codeword lengths that satisfy this inequality, there exists an instantaneous code with these word lengths.

- Consider a D-ary tree in which each node has D children.

  Let the branches of the tree represent the symbols of the codeword.

  For example, the D branches arising from the root node represent the D possible values of the first symbol of the codeword.

  Then each codeword is represented by a leaf on the tree.

  The path from the root traces out the symbols of the codeword.

Kraft Inequality (Cont'd)

- The prefix condition on the codewords implies that no codeword is an ancestor of any other codeword on the tree.

  Hence, a codeword eliminates its descendants as possible codewords.

  Let $\ell_{\max}$ be the length of the longest codeword.

  Consider all nodes of the tree at level $\ell_{\max}$.

  They may be codewords, descendants of codewords or neither.

  A codeword at level ℓ_i has $D^{\ell_{\max}-\ell_i}$ descendants at level $\ell_{\max}$.

  - Each of these descendant sets must be disjoint.
  - Also, the total number of nodes in these sets must be less than or equal to $D^{\ell_{\max}}$.

  So, summing over all codewords, we get $\sum_i D^{\ell_{\max}-\ell_i} \leq D^{\ell_{\max}}$.

  Equivalently,

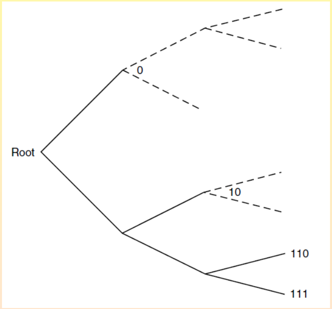
$$\sum_i D^{-\ell_i} \leq 1.$$

Kraft Inequality (Converse)

- Conversely, given any set of codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$ that satisfy the Kraft inequality, we can always construct a tree.

  Label the first node (lexicographically) of depth ℓ_1 as codeword 1.

  Remove its descendants from the tree.



  Then label the first remaining node of depth ℓ_2 as codeword 2.

  Continue in the same way.

  In the end, we obtain a prefix code with the specified codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$.

Extended Kraft Inequality

Theorem (Extended Kraft Inequality)

For any countably infinite set of codewords that form a prefix code, the codeword lengths satisfy the extended Kraft inequality,

$$\sum_{i=1}^{\infty} D^{-\ell_i} \leq 1.$$

Conversely, given any $\ell_1, \ell_2, \ldots$ satisfying the extended Kraft inequality, we can construct a prefix code with these codeword lengths.

- Let the D-ary alphabet be $\{0, 1, \ldots, D-1\}$.
  Consider the i-th codeword $y_1 y_2 \cdots y_{\ell_i}$.
  Let $0.y_1 y_2 \cdots y_{\ell_i}$ be the real number given by the D-ary expansion

$$0.y_1 y_2 \cdots y_{\ell_i} = \sum_{j=1}^{\ell_i} y_j D^{-j}.$$

Extended Kraft Inequality (Cont'd)

- The codeword $y_1 y_2 \cdots y_{\ell_i}$ corresponds to the interval
$$\left[0.y_1 y_2 \cdots y_{\ell_i}, 0.y_1 y_2 \cdots y_{\ell_i} + \frac{1}{D^{\ell_i}}\right),$$
the set of all real numbers whose D-ary expansion begins with $0.y_1 y_2 \cdots y_{\ell_i}$.
This is a subinterval of the unit interval $[0, 1]$.
By the prefix condition, these intervals are disjoint.
Hence, the sum of their lengths has to be less than or equal to 1.
This proves that
$$\sum_{i=1}^{\infty} D^{-\ell_i} \leq 1.$$

Extended Kraft Inequality (Converse)

- Just as in the finite case, we can reverse the proof to construct the code for a given $\ell_1, \ell_2, \ldots$ that satisfies the Kraft inequality.

  First, reorder the indexing so that $\ell_1 \leq \ell_2 \leq \cdots$.

  Then assign the intervals in order from the low end of the unit interval.

  For example, if we wish to construct a binary code with $\ell_1 = 1$, $\ell_2 = 2$, ..., we assign the intervals

$$\left[0, \frac{1}{2}\right), \quad \left[\frac{1}{2}, \frac{1}{4}\right), \ldots$$

  to the symbols, with corresponding codewords $0, 10, \ldots$.

Subsection 3

Optimal Codes

Minimization of Expected Length

- We consider the problem of finding the prefix code with the minimum expected length.
- This is equivalent to finding the set of lengths $\ell_1, \ell_2, \ldots, \ell_m$, such that:
  - $\ell_1, \ell_2, \ldots, \ell_m$ satisfy the Kraft inequality;
  - The expected length $L = \sum p_i \ell_i$ is less than the expected length of any other prefix code.
- In a standard optimization problem form:

$$\begin{gathered}\text{Minimize } L = \sum p_i \ell_i \\ \text{over all integers } \ell_1, \ell_2, \ldots, \ell_m \\ \text{satisfying } \sum D^{-\ell_i} \leq 1.\end{gathered}$$

Using Lagrange Multipliers

- Neglecting the integer constraint on ℓ_i and assuming equality in the constraint, we can use Lagrange multipliers to minimize

$$J = \sum p_i \ell_i + \lambda \left(\sum D^{\ell_i} \right).$$

Differentiating with respect to ℓ_i, we obtain

$$\frac{\partial J}{\partial \ell_i} = p_i - \lambda D^{-\ell_i} \log_e D.$$

Setting the derivative to 0, we obtain

$$D^{-\ell_i} = \frac{p_i}{\lambda \log_e D}.$$

Substituting this in the constraint, we find $\lambda = \frac{1}{\log_e D}$.

Hence $p_i = D^{-\ell_i}$. This yields optimal code lengths

$$\ell_i^* = -\log_D p_i.$$

Using Lagrange Multipliers (Cont'd)

- This noninteger choice of codeword lengths yields expected codeword length
$$L^* = \sum p_i \ell_i^* = -\sum p_i \log_D p_i = H_D(X).$$
But the ℓ_i must be integers.
So we should choose a set of codeword lengths ℓ_i "close" to the optimal set.

Lower Bound for the Expected Length

Theorem

The expected length L of any instantaneous D-ary code for a random variable X is greater than or equal to the entropy $H_D(X)$, i.e.,

$$L \geq H_D(X),$$

with equality if and only if $D^{-\ell_i} = p_i$.

- Express the difference between the expected length L and the entropy
$$\begin{array}{rcl} L - H_D(X) & = & \sum_i p_i \ell_i - \sum_i p_i \log_D \frac{1}{p_i} \\ & = & -\sum_i p_i \log_D D^{-\ell_i} + \sum_i p_i \log_D p_i. \end{array}$$
Now let $r_i = \frac{D^{-\ell_i}}{\sum_j D^{-\ell_j}}$ and $c = \sum_i D^{-\ell_i}$. Then we get
$$L - H = \sum_i p_i \log \frac{p_i}{r_i \sum_j D^{-\ell_j}}.$$

Lower Bound for the Expected Length (Cont'd)

- We obtained

$$\begin{aligned} L - H &= \sum_i p_i \log \frac{p_i}{r_i \sum_j D^{-\ell_j}} \\ &= \sum_i p_i \log_D \frac{p_i}{r_i} - \log_D c \\ &= D(\boldsymbol{p} \| \boldsymbol{r}) + \log_D \frac{1}{c}. \end{aligned}$$

  But relative entropy is nonnegative and, by Kraft, $c \leq 1$.

  Hence, $L \geq H$, with equality if and only if $p_i = D^{-\ell_i}$.

  Equivalently, if and only if $-\log_D p_i$ is an integer for all i.

Definition

A probability distribution is called D-**adic** if each of the probabilities is equal to D^{-n}, for some n.

- Thus, we have equality in the theorem if and only if the distribution of X is D-adic.

Procedure for Finding an Optimal Code

- Find the D-adic distribution that is closest (in the relative entropy sense) to the distribution of X.
  - This distribution provides the set of codeword lengths.
  - Then we construct the code by choosing the first available node as in the proof of the Kraft inequality.
  - We then have an optimal code for X.

  This procedure is not easy, since the search for the closest D-adic distribution is not obvious.
- In the next section, we give a good suboptimal procedure (Shannon-Fano coding).
- Then, we describe a simple procedure (Huffman coding) for actually finding the optimal code.

Subsection 4

Bounds on the Optimal Code Length

Recalling the Optimization Problem

- We wish to minimize $L = \sum p_i \ell_i$ subject to the constraint that $\ell_1, \ell_2, \ldots, \ell_m$ are integers and $\sum D^{-\ell_i} \leq 1$.
- We proved that the optimal codeword lengths can be found by finding the D-adic probability distribution closest to the distribution of X in relative entropy.
- That is, we need to find the D-adic $\boldsymbol{r}$, with

$$r_i = \frac{D^{-\ell_i}}{\sum_j D^{-\ell_j}},$$

minimizing

$$L - H_D = D(\boldsymbol{p} \| \boldsymbol{r}) - \log\left(\sum D^{-\ell_i}\right) \geq 0.$$

- The choice of word lengths $\ell_i = \log_D \frac{1}{p_i}$ yields $L = H$.

Approximating the Solution

- Since $\log_D \frac{1}{p_i}$ may not equal an integer, we round it up to give integer word-length assignments,
$$\ell_i = \left\lceil \log_D \frac{1}{p_i} \right\rceil,$$
where $\lceil x \rceil$ is the smallest integer $\geq x$.
- These lengths satisfy the Kraft inequality, since
$$\sum D^{-\left\lceil \log_D \frac{1}{p_i} \right\rceil} \leq \sum D^{-\log_D \frac{1}{p_i}} = \sum p_i = 1.$$
- This choice of codeword lengths satisfies $\log_D \frac{1}{p_i} \leq \ell_i < \log_D \frac{1}{p_i} + 1$.
- Multiplying by p_i and summing over i, we obtain
$$H_D(X) \leq L < H_D(X) + 1.$$
- An optimal code can only be better than this code.

Bounds for the Optimal Expected Codeword Length

Theorem

Let $\ell_1^*, \ell_2^*, \ldots, \ell_m^*$ be optimal codeword lengths for a source distribution $\boldsymbol{p}$ and a D-ary alphabet, and let L^* be the associated expected length of an optimal code ($L^* = \sum p_i \ell_i^*$). Then

$$H_D(X) \le L^* < H_D(X) + 1.$$

- Let $\ell_i = \left\lceil \log_D \frac{1}{p_i} \right\rceil$. Then ℓ_i satisfies the Kraft inequality.
  By the preceding slide, we have
  $$H_D(X) \le L = \sum p_i \ell_i < H_D(X) + 1.$$
  - L^*, the expected length of the optimal code, is less than $L = \sum p_i \ell_i$. So $L^* < L < H_D + 1$.
  - $L^* \ge H_D$ by a previous theorem.

  The conclusion follows.

Large Block Lengths to Spread the Overhead

- In the preceding theorem, there is an overhead that is at most 1 bit, due to the fact that $\log \frac{1}{p_i}$ is not always an integer.
- We can reduce the overhead per symbol by spreading it out over many symbols.
- Consider a system in which we send a sequence of n symbols from X.
- The symbols are assumed to be drawn i.i.d. according to $p(x)$.
- Consider these n symbols to be a supersymbol from alphabet $\mathcal{X}^n$.
- Define L_n to be the expected codeword length per input symbol.
- That is, if $\ell(x_1, x_2, \ldots, x_n)$ is the length of the binary codeword associated with $(x_1, x_2, \ldots, x_n)$ (we assume that $D = 2$, for simplicity), then

$$L_n = \frac{1}{n} \sum p(x_1, x_2, \ldots, x_n) \ell(x_1, x_2, \ldots, x_n) = \frac{1}{n} E\ell(X_1, X_2, \ldots, X_n).$$

Large Block Lengths to Spread the Overhead (Cont'd)

- We can now apply the bounds derived above to the code:

$$H(X_1, X_2, \ldots, X_n) \leq E\ell(X_1, X_2, \ldots, X_n) < H(X_1, X_2, \ldots, X_n) + 1.$$

- Since $X_1, X_2, \ldots, X_n$ are i.i.d.,

$$H(X_1, X_2, \ldots, X_n) = \sum H(X_i) = nH(X).$$

- Dividing the inequality by n, we obtain

$$H(X) \leq L_n < H(X) + \frac{1}{n}.$$

- Hence, by using large block lengths we can achieve an expected codelength per symbol arbitrarily close to the entropy.

The Non I.I.D. Case

Theorem

The minimum expected codeword length per symbol satisfies

$$\frac{H(X_1, X_2, \ldots, X_n)}{n} \leq L_n^* < \frac{H(X_1, X_2, \ldots, X_n)}{n} + \frac{1}{n}.$$

Moreover, if $X_1, X_2, \ldots, X_n$ is a stationary stochastic process,

$$L_n^* \to H(\mathcal{X}),$$

where $H(\mathcal{X})$ is the entropy rate of the process.

- As before, we have

$$H(X_1, X_2, \ldots, X_n) \leq E\ell(X_1, X_2, \ldots, X_n) < H(X_1, X_2, \ldots, X_n) + 1.$$

The Non I.I.D. Case (Cont'd)

- we have

$$H(X_1, X_2, \ldots, X_n) \leq E\ell(X_1, X_2, \ldots, X_n) < H(X_1, X_2, \ldots, X_n) + 1.$$

Dividing by n again and defining L_n to be the expected description length per symbol, we obtain

$$\frac{H(X_1, X_2, \ldots, X_n)}{n} \leq L_n < \frac{H(X_1, X_2, \ldots, X_n)}{n} + \frac{1}{n}.$$

If the stochastic process is stationary, then

$$\frac{H(X_1, X_2, \ldots, X_n)}{n} \to H(\mathcal{X}).$$

So the expected description length tends to the entropy rate as $n \to \infty$.

Wrong Distribution and Relative Entropy

- We ask what happens to the expected description length if the code is designed for the wrong distribution.
- The wrong distribution may be the best estimate that we can make of the unknown true distribution.
- We consider the Shannon code assignment $\ell(x) = \left\lceil \log \frac{1}{q(x)} \right\rceil$ designed for the probability mass function $q(x)$.
- Suppose that the true probability mass function is $p(x)$.
- We will not achieve expected length $L \approx H(p) = -\sum p(x) \log p(x)$.
- We show that the increase in expected description length due to the incorrect distribution is the relative entropy $D(p\|q)$.
- Thus, $D(p\|q)$ has a concrete interpretation as the increase in descriptive complexity due to incorrect information.

The Wrong Code Theorem

Theorem (Wrong Code)

The expected length under $p(x)$ of the code assignment $\ell(x) = \left\lceil \log \frac{1}{q(x)} \right\rceil$ satisfies

$$H(p) + D(p\|q) \leq E_p \ell(X) < H(p) + D(p\|q) + 1.$$

- The expected codelength is

$$\begin{array}{rcl} E\ell(X) & = & \sum_x p(x) \left\lceil \log \frac{1}{q(x)} \right\rceil \\ & < & \sum_x p(x) \left(\log \frac{1}{q(x)} + 1 \right) \\ & = & \sum_x p(x) \log \frac{p(x)}{q(x)} \frac{1}{p(x)} + 1 \\ & = & \sum_x p(x) \log \frac{p(x)}{q(x)} + \sum_x p(x) \log \frac{1}{p(x)} + 1 \\ & = & D(p\|q) + H(p) + 1. \end{array}$$

The lower bound can be derived similarly.

Subsection 5

Kraft Inequality for Uniquely Decodable Codes

Introduction

- We showed that any instantaneous code must satisfy the Kraft inequality.
- The class of uniquely decodable codes is larger than the class of instantaneous codes.
- So one expects to achieve a lower expected codeword length if L is minimized over all uniquely decodable codes.
- We prove that the class of uniquely decodable codes does not offer any further possibilities for the set of codeword lengths than do instantaneous codes.

Kraft Inequality for Uniquely Decodable Codes

Theorem (McMillan)

The codeword lengths of any uniquely decodable D-ary code must satisfy the Kraft inequality

$$\sum D^{-\ell_i} \leq 1.$$

Conversely, given a set of codeword lengths that satisfy this inequality, it is possible to construct a uniquely decodable code with these codeword lengths.

- Consider C^k, the k-th extension of the code (i.e., the code formed by the concatenation of k repetitions of the given code C).

  By unique decodability, the k-th extension of the code is nonsingular.

  But there are only D^n different D-ary strings of length n.

  Thus, unique decodability implies that the number of code sequences of length n in the k-th extension must be no greater than D^n.

Proof of McMillan's Theorem

- Let the codeword lengths of the symbols $x \in \mathcal{X}$ be denoted by $\ell(x)$. Concerning the extension code,

$$\ell(x_1, x_2, \ldots, x_k) = \sum_{i=1}^{k} \ell(x_i).$$

The inequality that we wish to prove is $\sum_{x \in \mathcal{X}} D^{-\ell(x)} \leq 1$.

The trick is to consider the k-th power of this quantity.

$$\begin{aligned} &(\textstyle\sum_{x \in \mathcal{X}} D^{-\ell(x)})^k \\ &= \textstyle\sum_{x_1 \in \mathcal{X}} \sum_{x_2 \in \mathcal{X}} \cdots \sum_{x_k \in \mathcal{X}} D^{-\ell(x_1)} D^{-\ell(x_2)} \ldots D^{-\ell(x_k)} \\ &= \textstyle\sum_{x_1, x_2, \ldots, x_k \in \mathcal{X}^k} D^{-\ell(x_1)} D^{-\ell(x_2)} \ldots D^{-\ell(x_k)} \\ &= \textstyle\sum_{x^k \in \mathcal{X}^k} D^{-\ell(x^k)}. \end{aligned}$$

Proof of McMillan's Theorem (Cont'd)

- We obtained
$$\left(\sum_{x\in\mathcal{X}} D^{-\ell(x)}\right)^k = \sum_{x^k\in\mathcal{X}^k} D^{-\ell(x^k)}.$$
Gathering the terms by word lengths, we obtain
$$\sum_{x^k\in\mathcal{X}^k} D^{-\ell(x^k)} = \sum_{m=1}^{k\ell_{\max}} a(m)D^{-m},$$
where:
    - $\ell_{\max}$ is the maximum codeword length;
    - $a(m)$ is the number of source sequences x^k mapping into codewords of length m.

  By unique decodability, there is at most one sequence mapping into each code m-sequence and there are at most D^m code m-sequences. Thus, $a(m) \leq D^m$.

Proof of McMillan's Theorem (Cont'd)

- Now we have

$$\left(\sum_{x\in\mathcal{X}} D^{-\ell(x)}\right)^k = \sum_{m=1}^{k\ell_{\max}} a(m)D^{-m} \leq \sum_{m=1}^{k\ell_{\max}} D^m D^{-m} = k\ell_{\max}.$$

Hence,

$$\sum_j D^{-\ell_j} \leq (k\ell_{\max})^{1/k}.$$

Since this inequality is true for all k, it is true in the limit as $k \to \infty$.

Since $(k\ell_{\max})^{1/k} \to 1$, we have $\sum_j D^{-\ell_j} \leq 1$.

Conversely, given any set of $\ell_1, \ell_2, \ldots, \ell_m$ satisfying the Kraft inequality, we can construct an instantaneous code.

But every instantaneous code is uniquely decodable.

So we have also constructed a uniquely decodable code.

Infinite Source Alphabets

Corollary

A uniquely decodable code for an infinite source alphabet $\mathcal{X}$ also satisfies the Kraft inequality.

- The preceding proof breaks down for infinite $|\mathcal{X}|$ at the inequality $\sum_j D^{-\ell_j} \leq (k\ell_{\max})^{1/k}$, since for an infinite code $\ell_{\max}$ is infinite.

  But there is a simple fix to the proof.

  Any subset of a uniquely decodable code is also uniquely decodable.

  Thus, any finite subset of codewords satisfies the Kraft inequality.

  Hence, $\sum_{i=1}^{\infty} D^{-\ell_i} = \lim_{N\to\infty} \sum_{i=1}^{N} D^{-\ell_i} \leq 1$.

  Given a set of word lengths $\ell_1, \ell_2, \ldots$ that satisfy the Kraft inequality, we can construct an instantaneous code.

  But, instantaneous codes are uniquely decodable. So we have a uniquely decodable code with an infinite number of codewords.

Subsection 6

Huffman Codes

Example

- Consider a random variable X taking values in $\mathcal{X} = \{1, 2, 3, 4, 5\}$, with probabilities $0.25, 0.25, 0.2, 0.15, 0.15$, respectively.

x	1	2	3	4	5
$p(x)$	0.25	0.25	0.2	0.15	0.15

We expect the optimal binary code for X to have the longest codewords assigned to the symbols 4 and 5.

Claim: These two lengths must be equal.

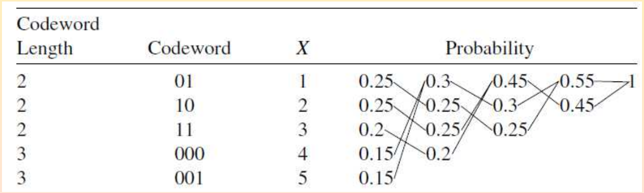
Otherwise, we can delete a bit from the longer codeword and still have a prefix code, but with a shorter expected length.

In general, we can construct a code in which the two longest codewords differ only in the last bit.

Example (Cont'd)

- For the optimal code, we can combine the symbols 4 and 5 into a single source symbol, with a probability assignment 0.30. Proceeding this way, until we are left with only one symbol:
  - Combine the two least likely symbols into one symbol.

  Then assign codewords to the symbols.



Codeword Length	Codeword	X	Probability				
2	01	1	0.25	0.3	0.45	0.55	1
2	10	2	0.25	0.25	0.3	0.45	
2	11	3	0.2	0.25	0.25		
3	000	4	0.15	0.2			
3	001	5	0.15				

- This code has average length

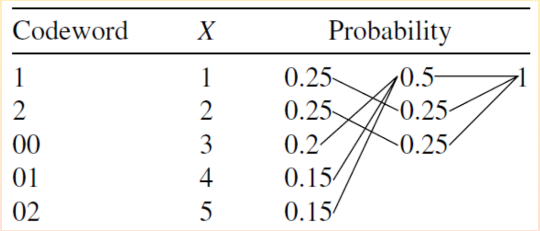
$$\begin{aligned} L &= 0.25 \cdot 2 + 0.25 \cdot 2 + 0.2 \cdot 2 + 0.15 \cdot 3 + 0.15 \cdot 3 \\ &= 0.5 + 0.5 + 0.4 + 0.45 + 0.45 = 2.3 \text{ bits.} \end{aligned}$$

Example

- Consider a ternary code for the same random variable X.

x	1	2	3	4	5
$p(x)$	0.25	0.25	0.2	0.15	0.15

We combine the three least likely symbols into one supersymbol.



Codeword	X	Probability		
1	1	0.25	0.5	1
2	2	0.25	0.25	
00	3	0.2	0.25	
01	4	0.15		
02	5	0.15		

This code has an average length of

$$\begin{aligned} L &= 0.25 \cdot 1 + 0.25 \cdot 1 + 0.2 \cdot 2 + 0.15 \cdot 2 + 0.15 \cdot 2 \\ &= 0.25 + 0.25 + 0.4 + 0.3 + 0.3 = 1.5 \text{ ternary digits.} \end{aligned}$$

Codes With At Least Three Symbols

- If $D \geq 3$, we may not have a sufficient number of symbols so that we can combine them D at a time.
- Then we add dummy symbols to the end of the set of symbols.
- These symbols have probability 0 and are inserted to fill the tree.
- At each stage, the number of symbols is reduced by $D - 1$.
- So we want the total number of symbols to be

$$1 + k(D - 1),$$

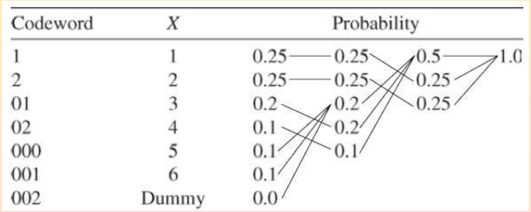
where k is the number of merges.

Example

- Consider $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$, with

x	1	2	3	4	5	6
$p(x)$	0.25	0.25	0.2	0.1	0.1	0.1

The merging process will take three steps of 3-merges. So we need a total of $1 + 3(3-1) = 7$ symbols.



Codeword	X	Probability			
1	1	0.25	0.25	0.5	1.0
2	2	0.25	0.25	0.25	
01	3	0.2	0.2	0.25	
02	4	0.1	0.2		
000	5	0.1	0.1		
001	6	0.1			
002	Dummy	0.0			

This code has an average length of

$$\begin{aligned} L &= 0.25 \cdot 1 + 0.25 \cdot 1 + 0.2 \cdot 2 + 0.1 \cdot 2 + 0.1 \cdot 3 + 0.1 \cdot 3 \\ &= 0.25 + 0.25 + 0.4 + 0.2 + 0.3 + 0.3 = 1.7 \text{ ternary digits.} \end{aligned}$$

Subsection 7

Some Comments on Huffman Codes

Comment 1: Source Coding and 20 Questions

- Suppose that we wish to find the most efficient series of yes-no questions to determine an object from a class of objects.
- To determine an object, we need to ensure that the responses to the sequence of questions uniquely identifies the object from the set of possible objects.
- It is not necessary that the last question have a “yes” answer.
- Assuming that we know the probability distribution on the objects, can we find the most efficient sequence of questions?

Source Coding and 20 Questions

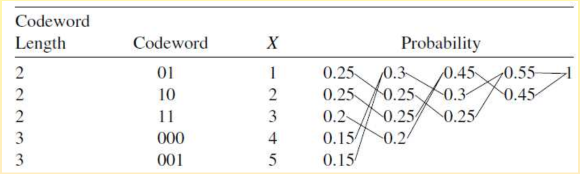
- Any question depends only on the answers to the questions before it.
- Since the sequence of answers uniquely determines the object, each object has a different sequence of answers.
- If we represent the yes-no answers by 0's and 1's, we have a binary code for the set of objects.
- The average length of this code is the average number of questions for the questioning scheme.

Source Coding and 20 Questions (Converse)

- From a binary code for the set of objects, we can find a sequence of questions that correspond to the code.
- The average number of questions equals the expected codeword length of the code.
- The first question in this scheme is "Is the first bit equal to 1 in the object's codeword?"
- The Huffman code is the best source code for a random variable.
- So the optimal series of questions is that determined by the Huffman code.

Source Coding and 20 Questions (Example)

- Consider again



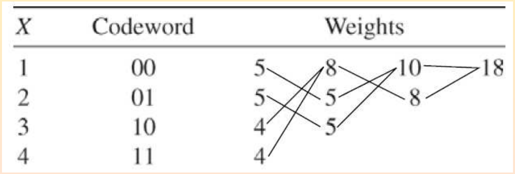
Codeword Length	Codeword	X	Probability				
2	01	1	0.25	0.3	0.45	0.55	1
2	10	2	0.25	0.25	0.3	0.45	
2	11	3	0.2	0.25	0.25		
3	000	4	0.15	0.2			
3	001	5	0.15				

- The optimal first question is: "Is X equal to 2 or 3?"
  The answer to this determines the first bit of the Huffman code.
- If the answer to the first question is "yes", the next question should be "Is X equal to 3?", which determines the second bit.
- We need not wait for the answer to the first to ask the second.
- We can ask as our second question "Is X equal to 1 or 3?", determining the second bit independent of the first.
- The expected number of questions EQ in this optimal scheme satisfies $H(X) \leq \text{EQ} < H(X) + 1$.

Comment 2: Huffman Coding for Weighted Codewords

- Huffman's algorithm for minimizing $\sum p_i \ell_i$ can be applied to any set of numbers $p_i \geq 0$, regardless of $\sum p_i$.
- In this case, the Huffman code minimizes the sum of weighted code lengths $\sum w_i \ell_i$ rather than the average code length.

  Example: We perform the weighted minimization.



X	Codeword	Weights			
1	00	5	8	10	18
2	01	5	5	8	
3	10	4	5		
4	11	4			

The code minimizes the weighted sum of the codeword lengths.

The minimum weighted sum is

$$L = 5 \cdot 2 + 5 \cdot 2 + 4 \cdot 2 + 4 \cdot 2 = 36.$$

Comment 3: Huffman Coding and “Slice” Questions

- We consider the game “20 questions” with a restricted set of questions.
- We assume that the elements of $X = \{1, 2, \ldots, m\}$ are ordered so that
$$p_1 \geq p_2 \geq \cdots \geq p_m.$$
- We also assume that the only questions allowed are of the form “Is $X > a$?”, for some a.

Huffman Coding and “Slice” Questions: Example

- The Huffman code constructed by the Huffman algorithm may not correspond to slices (sets of the form $\{x : x < a\}$).
- We construct another optimal code, as follows:
  - We take the codeword lengths ($\ell_1 \leq \ell_2 \leq \cdots \leq \ell_m$) derived from the Huffman code;
  - We use them to assign the symbols to the code tree by taking the first available node at the corresponding level.
- Now, each question (each bit of the code) splits the tree into sets of the form $\{x : x > a\}$ and $\{x : x < a\}$.
- So, unlike the Huffman code itself, this code is a slice code.

Huffman Coding and “Slice” Questions: Example

- Consider again a random variable X taking values in the set $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities 0.25, 0.25, 0.2, 0.15, 0.15, respectively.

  Using the Huffman coding procedure, we obtained the code

  $$C(1) = 01,\ C(2) = 10,\ C(3) = 11,\ C(4) = 000,\ C(5) = 001.$$

  This is not a slice code.

  We now use the lengths $\{2, 2, 2, 3, 3\}$ from the Huffman procedure.

  We assign the symbols to the first available node on the tree:

  $$1 \to 00,\ 2 \to 01,\ 3 \to 10,\ 4 \to 110,\ 5 \to 111.$$

  It can be verified that this code is a slice code.

  Such codes are also known as **alphabetic codes**, since the codewords are ordered alphabetically.

Comment 4: Huffman Codes and Shannon Codes

- Using codeword lengths of $\left\lceil \log \frac{1}{p_i} \right\rceil$ (which is called **Shannon coding**) may be much worse than the optimal code for some particular symbol.

  Example: Consider two symbols, one of which occurs with probability 0.9999 and the other with probability 0.0001.

  Then using codeword lengths of $\left\lceil \log \frac{1}{p_i} \right\rceil$ gives:

  - A codeword length of 1 bit for the first symbol.
  - A codeword of 14 bits for the second symbol.

  The optimal codeword length is obviously 1 bit for both symbols.

  Hence, the codeword for the infrequent symbol is much longer in the Shannon code than in the optimal code.

Huffman Codes and Shannon Codes (Cont'd)

- It is not always true that the codeword lengths for an optimal code are less than $\left\lceil \log \frac{1}{p_i} \right\rceil$.

  Example: Let X be a random variable with a distribution $(\frac{1}{3}, \frac{1}{3}, \frac{1}{4}, \frac{1}{12})$. The Huffman procedure results in one of the length tuples:
  - $(2, 2, 2, 2)$;
  - $(1, 2, 3, 3)$.

  Both these codes achieve the same expected codeword length.

  In the second code, the third symbol has length $3 > \left\lceil \log \frac{1}{p_3} \right\rceil$.

Huffman Codes and Shannon Codes (Cont'd)

- We saw that the codeword length for a Shannon code could be less than the codeword length of the corresponding symbol of an optimal (Huffman) code.
- We also saw that the set of codeword lengths for an optimal code is not unique.
- Although either the Shannon code or the Huffman code can be shorter for individual symbols, the Huffman code is shorter on average.
- Also, both the Shannon code and the Huffman code have expected codelength between H and $H + 1$.
- So they differ by less than 1 bit in expected codelength.

Comment 5: Fano Codes

- Fano proposed a suboptimal procedure for constructing a source code, which is similar to the idea of slice codes.
- In his method:
  - We first order the probabilities in decreasing order.
  - Then we choose k such that $|\sum_{i=1}^{k} p_i - \sum_{i=k+1}^{m} p_i|$ is minimized. This point divides the source symbols into two sets of almost equal probability.
  - Assign 0 for the first bit of the upper set and 1 for the lower set.
  - Repeat this process for each subset.
- By this recursive procedure, we obtain a code for each source symbol.
- This scheme, although not optimal in general, achieves

$$L(C) \leq H(X) + 2.$$

Subsection 8

Optimality of Huffman Codes

Properties of an Optimal Code

- Suppose the probability masses are ordered $p_1 \geq p_2 \geq \cdots \geq p_m$.
- Recall that a code is optimal if $\sum p_i \ell_i$ is minimal.

Lemma

For any distribution, there exists an optimal instantaneous code (with minimum expected length) that satisfies the following properties:

1. The lengths are ordered inversely with the probabilities, i.e.,
$$p_j > p_k \quad \text{implies} \quad \ell_j \leq \ell_k.$$
2. The two longest codewords have the same length.
3. Two of the longest codewords differ only in the last bit and correspond to the two least likely symbols.

- The proof amounts to swapping, trimming and rearranging.

  We consider an optimal code C_m.

Properties of an Optimal Code (Swapping)

- If $p_j > p_k$, then $\ell_j \leq \ell_k$.

  Here we swap codewords.

  Consider C'_m, with the codewords j and k of C_m interchanged.

  Then we have

$$\begin{aligned} L(C'_m) - L(C_m) &= \sum p_i \ell'_i - \sum p_i \ell_i \\ &= p_j \ell_k + p_k \ell_j - p_j \ell_j - p_k \ell_k \\ &= (p_j - p_k)(\ell_k - \ell_j). \end{aligned}$$

  But $p_j - p_k > 0$.

  Since C_m is optimal, $L(C'_m) - L(C_m) \geq 0$.

  Hence, we must have $\ell_k \geq \ell_j$.

  Thus, C_m itself satisfies Property 1.

Properties of an Optimal Code (Trimming)

- The two longest codewords are of the same length.

  Here we trim the codewords.

  Suppose the two longest codewords are not of the same length.

  Then we can delete the last bit of the longer one so that:
  - The prefix property is preserved;
  - A lower expected codeword length s achieved.

  Hence, the two longest codewords must have the same length.

  By Property 1, the longest codewords must belong to the least probable source symbols.

Properties of an Optimal Code (Rearranging)

- The two longest codewords differ only in the last bit and correspond to the two least likely symbols.

  If there is a maximal-length codeword without a sibling, we can delete the last bit of the codeword and still satisfy the prefix property.

  This reduces the average codeword length and contradicts the optimality of the code.

  So every maximal-length codeword in any optimal code has a sibling.

  Now we can exchange the longest codewords so that the two lowest probability source symbols are associated with two siblings on the tree.

  This does not change the expected length, $\sum p_i \ell_i$.

  Thus, the codewords for the two lowest-probability source symbols have maximal length and agree in all but the last bit.

Canonical Code

- Summarizing, suppose that

$$p_1 \geq p_2 \geq \cdots \geq p_m.$$

  Then, there exists an optimal code, with:
  - $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_{m-1} = \ell_m$;
  - Codewords $C(x_{m-1})$ and $C(x_m)$ differ only in the last bit.
- An optimal code satisfying the properties of the lemma is called a **canonical code**.

The Huffman Reduction

- For any probability mass function for an alphabet of size m, $\boldsymbol{p} = (p_1, p_2, \ldots, p_m)$ with $p_1 \geq p_2 \geq \cdots \geq p_m$, we define the **Huffman reduction**

  $$\boldsymbol{p}' = (p_1, p_2, \ldots, p_{m-2}, p_{m-1} + p_m)$$

  over an alphabet of size $m - 1$.
- Let $C^*_{m-1}(\boldsymbol{p}')$ be an optimal code for $\boldsymbol{p}'$.
- Let $C^*_m(\boldsymbol{p})$ be the canonical optimal code for $\boldsymbol{p}$.
- The proof of optimality will follow from two constructions:
  - First, we expand an optimal code for $\boldsymbol{p}'$ to construct a code for $\boldsymbol{p}$;
  - Then we condense an optimal canonical code for $\boldsymbol{p}$ to construct a code for the Huffman reduction $\boldsymbol{p}'$.

  Comparing the average codeword lengths establishes that the optimal code for $\boldsymbol{p}$ can be obtained by extending the optimal code for $\boldsymbol{p}'$.

Constructing an Extension Code

- From the optimal code for $\boldsymbol{p}'$, we construct an extension code for m elements as follows:
- Take the codeword in C^*_{m-1} corresponding to weight $p_{m-1}+p_m$.

$$\begin{array}{ccccc}
 & C^*_{m-1}(\boldsymbol{p}') & & C_m(\boldsymbol{p}) & \\
p_1 & w'_1 & \ell'_1 & w_1 = w'_1 & \ell_1 = \ell'_1 \\
p_2 & w'_2 & \ell'_2 & w_2 = w'_2 & \ell_2 = \ell'_2 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
p_{m-2} & w'_{m-2} & \ell'_{m-2} & w_{m-2} = w'_{m-2} & \ell_{m-2} = \ell'_{m-2} \\
p_{m-1}+p_m & w'_{m-1} & \ell'_{m-1} & w_{m-1} = w'_{m-1}0 & \ell_{m-1} = \ell'_{m-1}+1 \\
 & & & w_m = w'_{m-1}1 & \ell_m = \ell'_{m-1}+1
\end{array}$$

- Extend it by:
  - Adding a 0 to form a codeword for symbol $m-1$;
  - Adding 1 to form a codeword for symbol m.
- Calculating $\sum_i p'_i\ell'_i$ shows that $L(\boldsymbol{p}) = L^*(\boldsymbol{p}') + p_{m-1} + p_m$.

Contracting a Code

- From the canonical code for $\boldsymbol{p}$, we construct a code for $\boldsymbol{p}'$ by merging the codewords for the two lowest-probability symbols $m-1$ and m with probabilities p_{m-1} and p_m, which are siblings by the properties of the canonical code.
- The new code for $\boldsymbol{p}'$ has average length

$$\begin{aligned} L(\boldsymbol{p}') &= \sum_{i=1}^{m-2} p_i \ell_i + p_{m-1}(\ell_{m-1}-1) + p_m(\ell_m - 1) \\ &= \sum_{i=1}^{m} p_i \ell_i - p_{m-1} - p_m \\ &= L^*(\boldsymbol{p}) - p_{m-1} - p_m. \end{aligned}$$

Combining the Results

- We have
$$\begin{array}{rcl} L(\boldsymbol{p}) & = & L^*(\boldsymbol{p}') + p_{m-1} + p_m; \\ L(\boldsymbol{p}') & = & L^*(\boldsymbol{p}) - p_{m-1} - p_m. \end{array}$$
  Adding, we obtain $L(\boldsymbol{p}') + L(\boldsymbol{p}) = L^*(\boldsymbol{p}') + L^*(\boldsymbol{p})$.

  Equivalently,
$$(L(\boldsymbol{p}') - L^*(\boldsymbol{p}')) + (L(\boldsymbol{p}) - L^*(\boldsymbol{p})) = 0.$$
  We examine the two terms:
  - Since $L^*(\boldsymbol{p}')$ is the optimal length for $\boldsymbol{p}'$, $L(\boldsymbol{p}') - L^*(\boldsymbol{p}') \geq 0$.
  - The length of the extension of the optimal code for $\boldsymbol{p}'$ has to have an average length at least as large as the optimal code for $\boldsymbol{p}$. Therefore, $L(\boldsymbol{p}) - L^*(\boldsymbol{p}) \geq 0$.

  The sum of two nonnegative terms can be 0 only if both are 0.

  So the extension of the optimal code for $\boldsymbol{p}'$ is optimal for $\boldsymbol{p}$.

Optimality of the Huffman Code

- We start with a code for two elements.

  In this case the optimal code is obvious.
- Assume an optimal code for $\boldsymbol{p}'$ with $m-1$ symbols.
- We construct an optimal code for m symbols by extending the codeword corresponding to $p_{m-1}+p_m$.
- By induction, we construct an optimal code for any number of elements.

Theorem

Huffman coding is optimal, that is, if C^* is a Huffman code and C' is any other uniquely decodable code, $L(C^*) \leq L(C')$.

- Although we have proved the theorem for a binary alphabet, the proof can be extended to establishing optimality of the Huffman coding algorithm for a D-ary alphabet as well.

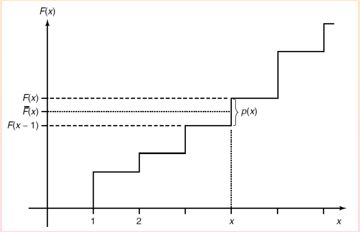
Subsection 9

Shannon-Fano-Elias Coding

Cumulative Distribution Functions

- We showed that the codeword lengths $\ell(x) = \left\lceil \log \frac{1}{p(x)} \right\rceil$ satisfy the Kraft inequality and can therefore be used to construct a uniquely decodable code for the source.
- We now describe a simple constructive procedure that uses the cumulative distribution function to allot codewords.
- Without loss of generality, we can take $\mathcal{X} = \{1, 2, \ldots, m\}$.
- Assume that $p(x) > 0$, for all x.
- The **cumulative distribution function** $F(x)$ is defined as

$$F(x) = \sum_{a \leq x} p(a).$$



Modified Cumulative Distribution Function

- Consider the modified cumulative distribution function
$$\overline{F}(x) = \sum_{a<x} p(a) + \frac{1}{2} p(x),$$
where $\overline{F}(x)$ denotes the sum of the probabilities of all symbols less than x plus half the probability of the symbol x.
- Since the random variable is discrete, the cumulative distribution function consists of steps of size $p(x)$.
- The value of $\overline{F}(x)$ is the midpoint of the step corresponding to x.
- Since all the probabilities are positive, $\overline{F}(a) \neq \overline{F}(b)$, if $a \neq b$.
- Hence, we can determine x if we know $\overline{F}(x)$.
- Thus, the value of $\overline{F}(x)$ can be used as a code for x.
- But, $\overline{F}(x)$ is a real number expressible only by infinitely many bits.
- So it is not efficient to use the exact value of $\overline{F}(x)$ as a code for x.

The Required Accuracy

- Assume that we truncate $\overline{F}(x)$ to $\ell(x)$ bits (denoted by $\lfloor \overline{F}(x) \rfloor_{\ell(x)}$).
- Thus, we use the first $\ell(x)$ bits of $\overline{F}(x)$ as a code for x.
- By definition of rounding off, we have $\overline{F}(x) - \lfloor \overline{F}(x) \rfloor_{\ell(x)} < \frac{1}{2^{\ell(x)}}$.
- If $\ell(x) = \left\lceil \log \frac{1}{p(x)} \right\rceil + 1$, then

$$\frac{1}{2^{\ell(x)}} < \frac{p(x)}{2} = \overline{F}(x) - F(x-1).$$

- Therefore, $\lfloor \overline{F}(x) \rfloor_{\ell(x)}$ lies within the step corresponding to x.
- Thus, $\ell(x)$ bits suffice to describe x.

The Prefix Property

- We also require the set of codewords to be prefix-free.
- To check whether the code is prefix-free, we consider each codeword $z_1 z_2 \dots z_\ell$ to represent the interval $\left[0.z_1 z_2 \cdots z_\ell, 0.z_1 z_2 \cdots z_\ell + \frac{1}{2^\ell}\right)$.
- The code is prefix-free if and only if the intervals corresponding to codewords are disjoint.
- The interval corresponding to any codeword has length $2^{-\ell(x)}$.
- This is less than half the height of the step corresponding to x.
- The lower end of the interval is in the lower half of the step.
- Thus, the upper end of the interval lies below the top of the step.
- So, the interval corresponding to any codeword lies entirely within the step corresponding to that symbol in $\overline{F}(x)$.
- Therefore, the intervals corresponding to different codewords are disjoint and the code is prefix-free.

Expected Codeword Length

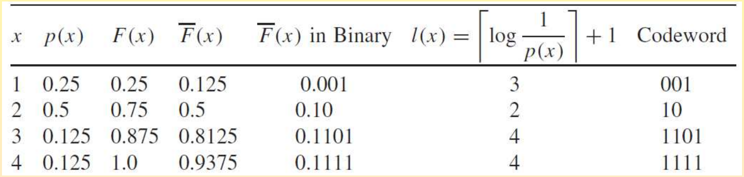
- In this code, we use $\ell(x) = \left\lceil \log \frac{1}{p(x)} \right\rceil + 1$ bits to represent x.
- So the expected length of the code is

$$\begin{aligned} L &= \sum_x p(x)\ell(x) \\ &= \sum_x p(x) \left(\left\lceil \log \frac{1}{p(x)} \right\rceil + 1 \right) \\ &< H(X) + 2. \end{aligned}$$

- Thus, this coding scheme achieves an average codeword length that is within 2 bits of the entropy.

Example (Dyadic Distribution)

- Consider an example where all the probabilities are dyadic.
  We construct the code in the following table.



x	$p(x)$	$F(x)$	$\overline{F}(x)$	$\overline{F}(x)$ in Binary	$l(x) = \left\lceil \log \frac{1}{p(x)} \right\rceil + 1$	Codeword
1	0.25	0.25	0.125	0.001	3	001
2	0.5	0.75	0.5	0.10	2	10
3	0.125	0.875	0.8125	0.1101	4	1101
4	0.125	1.0	0.9375	0.1111	4	1111

  We have

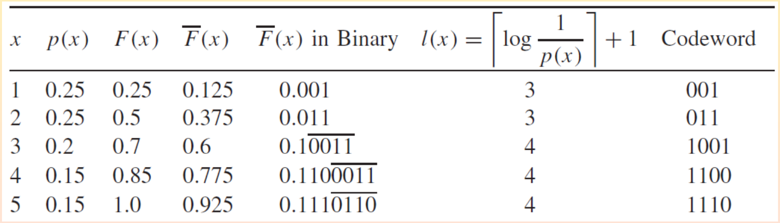
$$\begin{aligned} L &= 0.25 \cdot 3 + 0.5 \cdot 2 + 0.125 \cdot 4 + 0.125 \cdot 4 = 2.75; \\ H &= 0.25 \cdot 2 + 0.5 \cdot 1 + 0.125 \cdot 3 + 0.125 \cdot 3 = 1.75. \end{aligned}$$

  The Huffman code for this case achieves the entropy bound.
  Looking at the codewords, it is obvious that there is some inefficiency.
  E.g., the last bit of the last two codewords can be omitted.
  If we remove however, the last bit from all the codewords, the code is no longer prefix-free.

Example (Non-Dyadic Distribution)

- The distribution is not dyadic, so the representation of $F(x)$ in binary may have an infinite number of bits.

  We construct the code in the following table.



x	$p(x)$	$F(x)$	$\overline{F}(x)$	$\overline{F}(x)$ in Binary	$l(x) = \left\lceil \log \frac{1}{p(x)} \right\rceil + 1$	Codeword
1	0.25	0.25	0.125	0.001	3	001
2	0.25	0.5	0.375	0.011	3	011
3	0.2	0.7	0.6	$0.1\overline{0011}$	4	1001
4	0.15	0.85	0.775	$0.110\overline{0011}$	4	1100
5	0.15	1.0	0.925	$0.111\overline{0110}$	4	1110

  The above code is 1.2 bits longer on the average than the Huffman code for this source.

Subsection 10

Competitive Optimality of the Shannon Code

Idea of Competitive Optimality

- We have shown that Huffman coding is optimal in that it has minimum expected length.
- But, there are codes that assign shorter codewords to infrequent source symbols.
- So the Fuffman code's performance is not better than any other code on any particular sequence.
- Such comparisons give rise to the idea of competitive optimality.

Competitive Optimality Through Games

- To formalize the question of competitive optimality, we introduce a two-person zero-sum game.
- Two people are given a probability distribution and are asked to design an instantaneous code for the distribution.
- Then a source symbol is drawn from this distribution.
- Depending on whether the codeword of Player A for the source symbol drawn is shorter or longer than the codeword of Player B, the payoff to Player A is 1 or -1.
- The payoff is 0 for ties.

Competitive Optimality of the Shannon Code

- Consider the Shannon code with codeword lengths

$$\ell(x) = \left\lceil \log \frac{1}{p(x)} \right\rceil.$$

Theorem

Let $\ell(x)$ be the codeword lengths associated with the Shannon code and let $\ell'(x)$ be the codeword lengths associated with any other uniquely decodable code. Then

$$\Pr(\ell(X) \geq \ell'(X) + c) \leq \frac{1}{2^{c-1}}.$$

Competitive Optimality of the Shannon Code (Cont'd)

- We have

$$\begin{array}{rcl}
\Pr(\ell(X) \geq \ell'(X) + c) & = & \Pr\left(\left\lceil \log \frac{1}{p(X)} \right\rceil \geq \ell'(X) + c\right) \\
& \leq & \Pr\left(\log \frac{1}{p(X)} \geq \ell'(X) + c - 1\right) \\
& = & \Pr(p(X) \leq 2^{-\ell'(X)-c+1}) \\
& = & \sum_{x:p(x) \leq 2^{-\ell'(x)-c+1}} p(x) \\
& \leq & \sum_{x:p(x) \leq 2^{-\ell'(x)-c+1}} 2^{-\ell'(x)-(c-1)} \\
& \leq & \sum_x 2^{-\ell'(x)} 2^{-(c-1)} \\
& \overset{\text{Kraft}}{\leq} & 2^{-(c-1)}.
\end{array}$$

Strengthening the Theorem

- In a game-theoretic setting, one would like to ensure that $\ell(x) < \ell'(x)$ more often than $\ell(x) > \ell'(x)$.
- Recall that the probability mass function $p(x)$ is **dyadic** if $\log \frac{1}{p(x)}$ is an integer, for all x.

Theorem

For a dyadic probability mass function $p(x)$, let $\ell(x) = \log \frac{1}{p(x)}$ be the word lengths of the binary Shannon code for the source and let $\ell'(x)$ be the lengths of any other uniquely decodable binary code for the source. Then

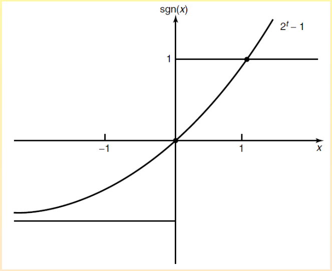
$$\Pr(\ell(X) < \ell'(X)) \geq \Pr(\ell(X) > \ell'(X)),$$

with equality if and only if $\ell'(x) = \ell(x)$, for all x. Thus, the code length assignment $\ell(x) = \log \frac{1}{p(x)}$ is uniquely competitively optimal.

Proof of the Strengthening

- Define the function $\operatorname{sgn}(t)$ by

$$\operatorname{sgn}(t) = \begin{cases} 1 & \text{if } t > 0 \\ 0 & \text{if } t = 0 \\ -1 & \text{if } t < 0 \end{cases}.$$



- Consider the graph showing $\operatorname{sgn}(t)$ and $2^t - 1$.
- It is easy to see that, for all $t = 0, \pm 1, \pm 2, \ldots$,

$$\operatorname{sgn}(t) \leq 2^t - 1.$$

- Although this inequality is not satisfied for all t, it is satisfied at all integer values of t.

Proof of the Strengthening (Cont'd)

- We can now write

$$\begin{aligned}
\Pr(\ell'(X) < \ell(X)) - \Pr(\ell'(X) > \ell(X)) \\
&= \textstyle\sum_{x:\ell'(x)<\ell(x)} p(x) - \sum_{x:\ell'(x)>\ell(x)} p(x) \\
&= \textstyle\sum_x p(x)\mathrm{sgn}(\ell(x) - \ell'(x)) \\
&= E\mathrm{sgn}(\ell(X) - \ell'(X)) \\
&\leq \textstyle\sum_x p(x)(2^{\ell(x)-\ell'(x)} - 1) \\
&= \textstyle\sum_x 2^{-\ell(x)}(2^{\ell(x)-\ell'(x)} - 1) \\
&= \textstyle\sum_x 2^{-\ell'(x)} - \sum_x 2^{-\ell(x)} \\
&= \textstyle\sum_x 2^{-\ell'(x)} - 1 \\
&\leq 1 - 1 = 0.
\end{aligned}$$

Proof of the Strengthening (Cont'd)

- We have equality in the above chain only if we have:
  - Equality in the bound for sgn;
  - Equality in the Kraft inequality.

  Now observe that:
  - We have equality in the bound for sgn(t) only if t is 0 or 1. I.e., $\ell(x) = \ell'(x)$ or $\ell(x) = \ell'(x) + 1$.
  - Equality in Kraft implies that $\ell'(x)$ satisfies the Kraft inequality with equality.

  Combining these two facts implies that $\ell'(x) = \ell(x)$, for all x.

A Consequence

Corollary

For nondyadic probability mass functions,

$$E\mathrm{sgn}(\ell(X) - \ell'(X) - 1) \leq 0,$$

where $\ell(x) = \left\lceil \log \frac{1}{p(x)} \right\rceil$ and $\ell'(x)$ is any other code for the source.

- Along the same lines as the preceding proof.
- We have shown that Shannon coding $\ell(x) = \left\lceil \log \frac{1}{p(x)} \right\rceil$ is optimal under a variety of criteria.
- It is robust with respect to the payoff function.
- In particular, for dyadic p:
  - $E(\ell - \ell') \leq 0$;
  - $E\mathrm{sgn}(\ell - \ell') \leq 0$;
  - $Ef(\ell - \ell') \leq 0$, for any function f, satisfying $f(t) \leq 2^t - 1$, for all $t = 0, \pm 1, \pm 2, \ldots$.

Subsection 11

Generation of Discrete Distributions from Fair Coins

Example

- How many fair coin flips does it take to generate a random variable X drawn according to a specified probability mass function $\boldsymbol{p}$?

  Example: Assume given a sequence of fair coin tosses (fair bits).

  Suppose we wish to generate a random variable X with distribution

$$X = \begin{cases} a, & \text{with probability } \frac{1}{2} \\ b, & \text{with probability } \frac{1}{4} \\ c, & \text{with probability } \frac{1}{4} \end{cases}.$$

  - If the first bit is 0, we let $X = a$.
  - If the first two bits are 10, we let $X = b$.
  - If we see 11, we let $X = c$.

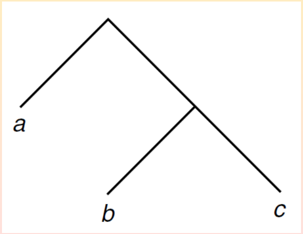
  The average number of fair bits required for generating X is

$$0.5 \cdot 1 + 0.25 \cdot 2 + 0.25 \cdot 2 = 1.5 \text{ bits.}$$

  This is also the entropy of the distribution.

The General Problem and the Binary Tree

- We are given a sequence of fair coin tosses $Z_1, Z_2, \ldots$.
- We generate a discrete random variable $X \in \mathcal{X} = \{1, 2, \ldots, m\}$, with probability mass function $\boldsymbol{p} = (p_1, p_2, \ldots, p_m)$.
- Let the random variable T denote the number of coin flips used in the algorithm.
- We can describe the algorithm mapping strings of bits $Z_1, Z_2, \ldots$ to possible outcomes X by a binary tree.
- The leaves of the tree are marked by output symbols X.



- The path to the leaves is given by the sequence of bits produced by the fair coin.

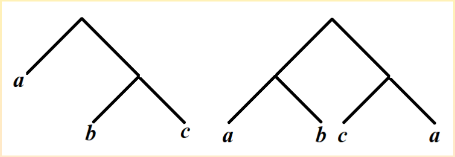
Tree Properties

- The tree representing the algorithm must satisfy certain properties.
  1. The tree should be complete (i.e., every node is either a leaf or has two descendants in the tree).
     The tree may be infinite.
  2. The probability of a leaf at depth k is 2^{-k}.
     Many leaves may be labeled with the same output symbol and the total probability of all these leaves should equal the desired probability of the output symbol.
  3. The expected number of fair bits ET required to generate X is equal to the expected depth of this tree.

Possible Algorithms

- There are many possible algorithms that generate the same output distribution.

  Example: Consider the trees



  They represent the mappings:

  - $0 \to a$, $10 \to b$, $11 \to c$;
  - $00 \to a$, $01 \to b$, $10 \to c$, $11 \to a$.

  Both yield the distribution $(\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$.

  However, the second uses two fair bits to generate each sample.

  So it is less efficient than the first which uses only 1.5 bits per sample.

Efficiency of Possible Algorithms

- What is the most efficient algorithm to generate a given distribution, and how is this related to the entropy of the distribution?
- We expect that we need at least as much randomness in the fair bits as we produce in the output samples.
  - Entropy is a measure of randomness.
  - Each fair bit has an entropy of 1 bit.

  So we expect that the number of fair bits used will be at least equal to the entropy of the output.

Lemma About Trees

Lemma

For any complete tree, consider a probability distribution on the leaves, such that the probability of a leaf at depth k is 2^{-k}. Then the expected depth of the tree is equal to the entropy of this distribution.

- Let $\mathcal{Y}$ denote the set of leaves of a complete tree.

  Let Y be a random variable with the distribution described.

  Let $k(y)$ denote the depth of leaf y.

  The expected depth of the tree is $ET = \sum_{y \in \mathcal{Y}} k(y) 2^{-k(y)}$.

  The entropy of the distribution of Y is

$$H(Y) = -\sum_{y \in \mathcal{Y}} \frac{1}{2^{k(y)}} \log \frac{1}{2^{k(y)}} = \sum_{y \in \mathcal{Y}} k(y) 2^{-k(y)}.$$

  Thus, $H(Y) = ET$.

Expected Number of Fair Bits

Theorem

For any algorithm generating X, the expected number of fair bits used is greater than the entropy $H(X)$, i.e.,

$$ET \geq H(X).$$

- We represent the algorithm generating X by a complete binary tree.

  Label the leaves of this tree by distinct symbols $y \in \mathcal{Y} = \{1, 2, \ldots\}$.

  If the tree is infinite, the alphabet $\mathcal{Y}$ is also infinite.

  Let Y be the random variable defined on the leaves of the tree.

  For any leaf y at depth k, the probability that $Y = y$ is 2^{-k}.

  By the lemma, the expected depth of the tree equals the entropy of Y: $ET = H(Y)$.

Expected Number of Fair Bits (Cont'd)

- The random variable X is a function of Y.

  In fact, one or more leaves map onto an output symbol.

  Hence, by a preceding problem, we have

  $$H(X) \leq H(Y).$$

  We conclude that, for any algorithm generating the random variable X, we have

  $$H(X) \leq ET.$$

Optimality for a Dyadic Distribution

Theorem

Let the random variable X have a dyadic distribution. The optimal algorithm to generate X from fair coin flips requires an expected number of coin tosses precisely equal to the entropy:

$$ET = H(X).$$

- By the preceding theorem, we need at least $H(X)$ bits to generate X.

  For the converse, we use the Huffman code tree for X as the tree to generate the random variable.

  For a dyadic distribution, the Huffman code is the same as the Shannon code and achieves the entropy bound.

Optimality for a Dyadic Distribution (Cont'd)

- For any $x \in \mathcal{X}$, the depth of the leaf in the code tree corresponding to x is the length of the corresponding codeword, $\log \frac{1}{p(x)}$.

  Hence, when this code tree is used to generate X, the leaf will have a probability
  $$2^{-\log \frac{1}{p(x)}} = p(x).$$

  The expected number of coin flips is the expected depth of the tree.

  The expected depth of the tree equals the entropy.

  Hence, for a dyadic distribution, the optimal generating algorithm achieves $ET = H(X)$.

The Non-Dyadic Case

- The code tree for the Huffman code generates a dyadic distribution on the leaves.
- Thus, if the given distribution is not dyadic, the code tree for the Huffman code will not generate the given distribution.
- Now all the leaves of the tree have probabilities of the form 2^{-k}.
- So we must do the following:
  - Split any probability p_i that is not of this form into atoms of this form;
  - Allot these atoms to leaves on the tree.

Example

- Suppose one of the outcomes x has probability $p(x) = \frac{1}{4}$.

  In this case, we need only one atom;

  Namely, a leaf of the tree at level 2.
- On the other hand, suppose $p(x) = \frac{7}{8}$.

  Then we decompose

  $$p(x) = \frac{7}{8} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8}.$$

  So, in this case, we need three atoms:
  - One at level 1;
  - One at level 2;
  - One at level 3.

The Non-Dyadic Case (Cont'd)

- To minimize the expected depth of the tree, we should use atoms with as large a probability as possible.
- So given a probability p_i:
  - We find the largest atom of the form 2^{-k} that is less than p_i. We allot this atom to the tree.
  - We calculate the remainder. We find that largest atom that will fit in the remainder.
  - Continuing this process, we can split all the probabilities into dyadic atoms.
- This process is equivalent to finding the binary expansions of the probabilities.

The Non-Dyadic Case (Formalism)

- Let the binary expansion of the probability p_i be
$$p_i = \sum_{j \geq 1} p_i^{(j)},$$
where $p_i^{(j)} = 2^{-j}$ or 0.
- The atoms of the expansion are the $\{p_i^{(j)} : i = 1, 2, \ldots, m, j \geq 1\}$.
- Since $\sum_i p_i = 1$, the sum of the probabilities of these atoms is 1.
- We will allot an atom of probability 2^{-j} to a leaf at depth j.
- The depths of the atoms satisfy the Kraft inequality.
- Hence, we can always construct such a tree with all the atoms at the right depths.

Example

- Let X have the distribution

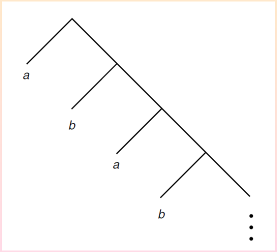
$$X = \begin{cases} a, & \text{with probability } \frac{2}{3} \\ b, & \text{with probability } \frac{1}{3} \end{cases}.$$

We find the binary expansions of these probabilities:

$$\begin{aligned} \frac{2}{3} &= 0.10101010\ldots; \\ \frac{1}{3} &= 0.01010101\ldots. \end{aligned}$$

The atoms for the expansion are

$$\begin{aligned} \frac{2}{3} &\rightarrow (\frac{1}{2}, \frac{1}{8}, \frac{1}{32}, \ldots); \\ \frac{1}{3} &\rightarrow (\frac{1}{4}, \frac{1}{16}, \frac{1}{64}, \ldots). \end{aligned}$$



Bounds for the Expected Length

Theorem

The expected number of fair bits required by the optimal algorithm to generate a random variable X lies between $H(X)$ and $H(X)+2$,

$$H(X) \leq ET < H(X)+2.$$

- The lower bound was proved in a previous theorem.

  For the upper bound, we write down an explicit expression for the expected number of coin tosses required for the procedure.

  Split all the probabilities $(p_1, p_2, \ldots, p_m)$ into dyadic atoms

  $$p_i \rightarrow (p_i^{(1)}, p_i^{(2)}, \ldots).$$

  Using these atoms, which form a dyadic distribution, we construct a tree with leaves corresponding to each of these atoms.

Bounds for the Expected Length (Cont'd)

- The number of coin tosses required to generate each atom is its depth in the tree.

  Therefore, the expected number of coin tosses is the expected depth of the tree.

  This is equal to the entropy of the dyadic distribution of the atoms.

  Hence, $ET = H(Y)$, where Y has the distribution

  $$(p_1^{(1)}, p_1^{(2)}, \ldots, p_2^{(1)}, p_2^{(2)}, \ldots, p_m^{(1)}, p_m^{(2)}, \ldots).$$

  Since X is a function of Y, we have

  $$H(Y) = H(Y, X) = H(X) + H(Y|X).$$

  Our objective is to show that $H(Y|X) < 2$.

Bounds for the Expected Length (Cont'd)

- Expanding the entropy of Y, we have

$$\begin{array}{rcl} H(Y) & = & -\sum_{i=1}^{m}\sum_{j\geq 1} p_i^{(j)}\log p_i^{(j)} \\ & = & \sum_{i=1}^{m}\sum_{j:p_i^{(j)}>0} j2^{-j}, \end{array}$$

  since each of the atoms is either 0 or 2^{-k} for some k.

  Consider the term T_i in the sum $T_i = \sum_{j:p_i^{(j)}>0} j2^{-j}$.

  We can find an n, such that $2^{-(n-1)} > p_i \geq 2^{-n}$.

  Equivalently, $n-1 < -\log p_i \leq n$. Then $p_i^{(j)} > 0$ only if $j \geq n$.

  So

$$T_i = \sum_{j:j\geq n, p_i^{(j)}>0} j2^{-j}.$$

  Using the definition of the atom, we write $p_i = \sum_{j:j\geq n,p_i^{(j)}>0} 2^{-j}$.

Bounds for the Expected Length (Claim)

Claim: $T_i < -p_i \log p_i + 2p_i$.

$$\begin{array}{rcl}
T_i + p_i \log p_i - 2p_i & < & T_i - p_i(n-1) - 2p_i \\
& = & T_i - (n-1+2)p_i \\
& = & \sum_{j:j \geq n, p_i^{(j)} > 0} j2^{-j} \\
& & \qquad - (n+1) \sum_{j:j \geq n, p_i^{(j)} > 0} 2^{-j} \\
& = & \sum_{j:j \geq n, p_i^{(j)} > 0} (j-n-1)2^{-j} \\
& = & -2^{-n} + 0 + \sum_{j:j \geq n+2, p_i^{(j)} > 0} (j-n-1)2^{-j} \\
& = & -2^{-n} + \sum_{k:k \geq 1, p_i^{(k+n+1)} > 0} k2^{-(k+n+1)} \\
& \leq & -2^{-n} + \sum_{k:k \geq 1} k2^{-(k+n+1)} \\
& = & -2^{-n} + 2^{-(n+1)}2 = 0.
\end{array}$$

We have shown that $T_i < -p_i \log p_i + 2p_i$.

Now $ET = \sum_i T_i < -\sum_i p_i \log p_i + 2\sum_i p_i = H(X) + 2$.