Huffman Coding

- Huffman (1951)
- Uses frequencies of symbols in a string to build a variable rate prefix code.
  - Each symbol is mapped to a binary string.
  - More frequent symbols have shorter codes.
  - No code is a prefix of another.
- Example:
  - a = 0
  - b = 100
  - c = 101
  - d = 11

Variable Rate Code Example

- Example: a 0, b 100, c 101, d 11
- Coding:
  - aabddcaa = 16 bits
  - 0 0 100 11 11 101 0 0 = 14 bits
- Prefix code ensures unique decodability.
  - 0010011111010
  - a a b d d c a a
- Morse Code an example of variable rate code. E = . and Z = _ _ . .

Huffman Tree for a Prefix Code

- Example: a 0, b 100, c 101, d 11

Encoding and Decoding

- Encoding: send the code, then the encoded data
  - x = aabddcaa
  - c(x) = a0 b100 c101 d11 0010011111010 0
  - Huffman code code of x
- Decoding: Build the Huffman tree, then use it to decode.

Cost of a Huffman Tree

- Let \( p_1, p_2, \ldots, p_m \) be the probabilities for the symbols \( a_1, a_2, \ldots, a_m \) respectively.
- Define the cost of the Huffman tree \( T \) to be
  \[
  C(T) = \sum_{i=1}^{m} p_i r_i
  \]
  where \( r_i \) is the length of the path from the root to \( a_i \).
- \( C(T) \) is the expected length of the code of a symbol coded by the tree \( T \).
**Example of Cost**

- Example: $a \, 1/2$, $b \, 1/8$, $c \, 1/8$, $d \, 1/4$

  \[
  C(T) = 1 \times 1/2 + 3 \times 1/8 + 3 \times 1/8 + 2 \times 1/4 = 1.75
  \]

**Optimal Huffman Tree**

- Input: Probabilities $p_1, p_2, \ldots, p_m$ for symbols $a_1, a_2, \ldots, a_m$, respectively.
- Output: A Huffman tree that minimizes the average number of bits to code a symbol. That is, minimizes $C(T) = \sum_{i=1}^m p_i r_i$, where $r_i$ is the length of the path from the root to $a_i$.

**Optimality Principle 1**

- In an optimal Huffman tree a lowest probability symbol has maximum distance from the root.
  - If not exchanging a lowest probability symbol with one at maximum distance will lower the cost.

  \[
  C(T') = C(T) + hq - hr + kr - kq = C(T) - (h-k)(r-q) < C(T)
  \]

**Optimality Principle 2**

- The second lowest probability is a sibling of the the smallest in some optimal Huffman tree.
  - If not, we can move it there not raising the cost.

  \[
  C(T') = C(T) + hq - hr + kr - kq = C(T) - (h-k)(r-q) < C(T)
  \]

**Optimality Principle 3**

- Assuming we have an optimal Huffman tree $T$ whose two lowest probability symbols are siblings at maximum depth, they can be replaced by a new symbol whose probability is the sum of their probabilities.
  - The resulting tree is optimal for the new symbol set.

  \[
  C(T') = C(T) + (h-1)(p+q) - hp - hq = C(T) - (p+q)
  \]

**Optimality Principle 3 (cont')**

- If $T'$ were not optimal then we could find a lower cost tree $T''$. This will lead to a lower cost tree $T'''$ for the original alphabet.

  \[
  C(T'') = C(T') + p + q < C(T') + p + q = C(T) \text{ which is a contradiction}
  \]
Huffman Tree Algorithm

- form a node for each symbol $a_i$ with weight $p_i$;
- insert the nodes in a min priority queue ordered by probability;
- while the priority queue has more than one element do
  - min1 := delete-min;
  - min2 := delete-min;
  - create a new node $n$;
  - $n$ weight := min1.weight + min2.weight;
  - $n$.left := min1;
  - $n$.right := min2;
  - insert($n$);

return the last node in the priority queue.

Example of Huffman Tree Algorithm (1)

- $P(a)$ = .4, $P(b)$ = .1, $P(c)$ = .3, $P(d)$ = .1, $P(e)$ = .1

Example of Huffman Tree Algorithm (2)

Example of Huffman Tree Algorithm (3)

Example of Huffman Tree Algorithm (4)

Optimal Huffman Code

average number of bits per symbol is

\[ .4 \times 1 + .1 \times 4 + .3 \times 2 + .1 \times 3 + .1 \times 4 = 2.1 \]
Huffman Code vs. Entropy

• P(a) = .4, P(b) = .1, P(c) = .3, P(d) = .1, P(e) = .1

Entropy

\[ H = -(.4 \times \log_2(.4) + .1 \times \log_2(.1) + .3 \times \log_2(.3) + .1 \times \log_2(.1) + .1 \times \log_2(.1)) \]

= 2.05 bits per symbol

Huffman Code

\[ HC = .4 \times 1 + .1 \times 4 + .3 \times 2 + .1 \times 3 + .1 \times 4 \]

= 2.1 bits per symbol

pretty good!

Quality of the Huffman Code

• The Huffman code is within one bit of the entropy lower bound.

\[ H \leq HC \leq H + 1 \]

• Huffman code does not work well with a two symbol alphabet.

– Example: P(0) = 1/100, P(1) = 99/100

– HC = 1 bits/symbol

\[ H = -((1/100)\log_2(1/100) + (99/100)\log_2(99/100)) \]

= .08 bits/symbol

Extending the Alphabet

• Assuming independence P(ab) = P(a)P(b), so we can lump symbols together.

• Example: P(0) = 1/100, P(1) = 99/100

– P(00) = 1/10000, P(01) = P(10) = 99/10000,

P(11) = 9801/10000.

Still not that close to H = .08 bits/bit

Including Context

• Suppose we add a one symbol context. That is in compressing a string \( x_1 x_2 \ldots x_n \) we want to take into account \( x_{k-1} \) when encoding \( x_k \).

– New model, so entropy based on just independent probabilities of the symbols doesn’t hold. The new entropy model (2nd order entropy) has for each symbol a probability for each other symbol following it.

– Example: \{a, b, c\}

\[
\begin{array}{c|ccc}
\text{prev} & a & b & c \\
\hline
a & .4 & .2 & .4 \\
b & 1 & 0 & 0 \\
c & .1 & .1 & .8
\end{array}
\]

Multiple Codes

• Time to design Huffman Code is \( O(n \log n) \)

where \( n \) is the number of symbols.

• Typically, for compressing a string the probabilities are chosen as the actual frequencies of the symbols in the string.

• Huffman works better for larger alphabets.

• There are adaptive (one pass) Huffman coding algorithms. No need to transmit code.

• Huffman is still popular. It is simple and it works.
Arithmetic Coding

- Huffman coding works well for larger alphabets and gets to within one bit of the entropy lower bound. Can we do better. Yes

- Basic idea in arithmetic coding:
  - Represent each string $x$ of length $n$ by an interval $A$ in $[0,1)$.
  - The width of the interval $A$ represents the probability of $x$ occurring.
  - The interval $A$ can itself be represented by any number, called a tag, within the half open interval.
  - The significant bits of the tag is the code of $x$.

Example of Arithmetic Coding (1)

Example of Arithmetic Coding (2)