Optimality of Huffman codes

Recursive construction of a „Huffman code" $C: \mathcal{X} \to A^+$ by building an $|A|$-ary tree from its leaves:

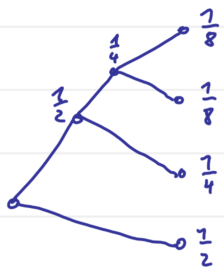
step 0) assign each symbol from $\mathcal{X}$ to a leaf,

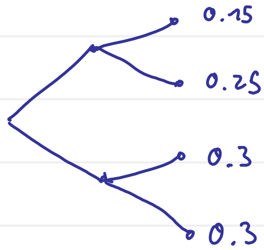
step 1) assign the $|A|$ least probable symbols to leaves with a common vertex,

step 2) 'combine' these symbols to a single one whose probability equals the sum of the $|A|$ probabilities.

Then iterate $1) \to 2) \to 1) \to \ldots$ until there is only one symbol (= root) left.

Examples:





Thm.: Given a random variable X and an alphabet A. The Huffman code is optimal in the set of all prefix-free symbol codes $C: \mathcal{X} \to A^+$.

proof: by induction (only $|A|=2$):

- basis for induction: optimality holds for $n := |\mathcal{X}| = 2$
- induction Hypothesis: Huffman code is optimal for n. $(*)$
- induction step: assume $(*)$, let $|\mathcal{X}| = n+1$ and $p(x_n), p(x_{n+1})$ be smallest probabilities. Then

$$\underbrace{L(C_{n+1})}_{\text{Huffman codes}} = \underbrace{L(C_n)}_{\text{Huffman codes}} + p(x_n) + p(x_{n+1})$$

since $L(C_{n+1}) = l_1 p(x_1) + \dots l_{n-1} p(x_{n-1}) + l_n p(x_n) + l_{n+1} p(x_{n+1})$ with $l_n = l_{n+1}$

$L(C_n) =$ —"— $+ (l_n - 1)(p(x_n) + p(x_{n+1}))$

Now assume $L(C'_{n+1}) < L(C_{n+1})$ for an optimal prefix-free code C'_{n+1}.

Optimality $\Rightarrow l'_n = l'_{n+1} = \max\{l'_i\}$ and we can assume x_n, x_{n+1} to be neighbors on the tree of C'_{n+1}

$$\text{Then } L(C_n) \overset{(*)}{\leq} L(C'_n) = L(C'_{n+1}) - p(x_{n+1}) - p(x_n)$$
$$< L(C_{n+1}) - p(x_{n+1}) - p(x_n) = L(C_n) \ \lightning$$

$\square$

<u>Exp.:</u> Huffman code for English language (see MacKay)

$L(C) = 4.15$ bits, compared to $H(X) = 4.11$ bits

(entropy rate, however, is about 1bit/symbol)

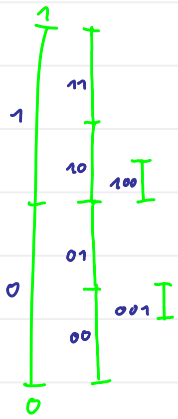
<u>remarks:</u>
- Huffman codes are used in the final level of the JPEG algorithm,
- since any strategy for the Bar Kochba game corresponds to a prefix-free code and vice versa, Huffman codes provide the optimal strategy.

III.2. Stream codes

Motivation: "guessing game" (→ see MacKay)

III.2.1. Arithmetic codes

Basic idea:



- $I: \{0,1\}^+ \to \{ [a,b) \mid 0 \leq a < b \leq 1 \}$

  $I(a_1 \dots a_n) := \left[\sum_{k=1}^{n} a_k 2^{-k}, \sum_{k=1}^{n} a_k 2^{-k} + 2^{-n} \right)$

  Note: $I(a_1 \dots a_n) \supseteq I(a_1 \dots a_{n+1})$

- Define $J: \mathcal{X}^+ \to \{ [a,b) \mid 0 \leq a < b \leq 1 \}$ similarly, but

  s.t. $|J(x_1 \dots x_n)| = p(x_1, \dots, x_n)$

- Encode $x_1 \dots x_n$ into $a_1 \dots a_K$ s.t.

  $I(a_1 \dots a_K) \subseteq J(x_1 \dots x_n)$ & K is smallest possible.

- Encoding & decoding can be done 'on the fly'

- For a practicable algorithm J is constructed s.t. it depends only on a window of a fixed number of x_i's.

- arithmetic coding requires a model for the probabilities

- Applications:
  - Dasher
  - DjVu

III.2.2. Lempel-Ziv coding

Idea: Replace a substring by a pointer to an earlier occurrence of the same substring.

Example:

original string:	1	0	11	01	010	00	0101	01010
# of substring:	1	2	3	4	5	6	7	8
(pointer, additional bit):	(0,1)	(0,0)	(1,1)	(2,1)	(4,0)	(2,0)	(5,1)	(7,0)

remarks:

- applied in compress & gzip
- does not require a probabilistic model for the source
- Lempel-Ziv coding compresses asymptotically down to the entropy rate (for ergodic stationary stochastic processes)
- for too short strings the 'compressed' message can be longer than the original one
- a variant of LZ (Lempel-Ziv-Welch) is used in the image format GIF.