SIGNAL COMPRESSION
Lecture 7

Variable–to–Fix Encoding

1. Tunstall codes

2. Petry codes

1. Variable-to-Fix coding

- Parse the input string into variable length segments.
- Associate a index to each possible segment.
- A parsing tree will fully specify the code.

Some of the variants of the Ziv-Lempel methods can be seen as variable-to-fix codes, defining a parsing tree (which is adaptively grown) and allocating an index to each segment, but in most implementations Ziv-Lempel is a variable-to-variable code, since the segment indices are further encoded by fix-to variable coding.

Run-length coding can also be seen as a VTF coding scheme: it parses the input into strings (either isolated characters, or runs of the same character) and encodes them into a fixed number of bits (for some variants of run-length coding also the length of the bitstream included for each segment is varying).
A simple example

<table>
<thead>
<tr>
<th>Variable length segment</th>
<th>Codewords</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>0010</td>
<td>001</td>
</tr>
<tr>
<td>0011</td>
<td>010</td>
</tr>
<tr>
<td>01</td>
<td>011</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>110</td>
<td>101</td>
</tr>
<tr>
<td>1110</td>
<td>110</td>
</tr>
<tr>
<td>1110</td>
<td>111</td>
</tr>
</tbody>
</table>

![Variable length segment tree](image)
A simple example

• assume an independent Bernoulli source, with probability \( p(1) = 0.2 \). Then \( p(000)=0.512; \)
  \( p(0010)=0.1024; p(0011)=0.0256; p(01)=p(10)=0.16; p(110)=0.032; \)
  \( p(1110)=0.0064; p(1111)=0.0016; \)

• the number of segments to be used in parsing is \( N = 8. \)

• The average length of the parsed string

\[
EL = [0.512 0.1024 0.0256 0.16 0.16 0.032 0.0064 0.0016] \ast [3 4 4 2 2 3 4 4]' = 2.8160 \tag{1}
\]

• The number of bits to encode a parsed string is \( k = \log_2 N = 3 \) bits.

• The average rate of the code (number of bits per sample) is

\[
R = \frac{\log_2 N}{EL} = \frac{3}{2.8160} = 1.0653 \tag{2}
\]

• The entropy of the Bernoulli source is

\[
H = 0.7219 \tag{3}
\]
• The rate of the Huffman code for blocks of $n_b$ symbols is

\[
\begin{align*}
  n_b = 1 & \quad EL_H = 1 \\
  n_b = 2 & \quad EL_H = 0.78 \\
  n_b = 3 & \quad EL_H = 0.728
\end{align*}
\]
The optimal design of the Variable to Fix code for a Bernoulli source

- For a given number \( N \) of leaves in the parsing tree, find the parsing tree which minimizes the rate \( R = \frac{\log_2 N}{EL} \), i.e. maximizes the average length of the parsed strings \( EL \). The parsing tree should be complete (in order to be able to parse every possible input string).

- **Lemma 1** The average length of the parsed strings equals the sum of probabilities at all interior nodes of the parsing tree. Denote \( W \) the parsing tree and \( \text{Int}(W) \) the set of interior nodes of the parsing tree.

\[
EL = \sum_{w \in \text{Int}(W)} p(w) \tag{4}
\]

- In the previous example \( \text{Int}(W) = \{\emptyset, 0, 00, 001, 1, 11, 111\} \). \( P(\emptyset) + P(0) + P(00) + P(001) + P(1) + P(11) + P(111) = 1 + 0.8^1 \cdot 0.2^0 + 0.8^2 \cdot 0.2^0 + 0.8^2 \cdot 0.2^1 + 0.8^0 \cdot 0.2^1 + 0.8^0 \cdot 0.2^2 + 0.8^0 \cdot 0.2^3 = 2.8160 \)

- **Proof of Lemma 1** Start with the elementary tree having two leaves. The claim holds true for this particular tree: the average length is \( EL = 1 \), the interior node is the root, having probability \( P(\emptyset) = 1 \), so \( EL = \sum_{w \in \text{Int}(W)} p(w) \). By induction we show that this property should hold for a tree with \( N + 1 \) leaves, if it holds for a tree with \( N \) leaves.

Every parsing tree has at least two sibling leaves (it must be complete). We transform the parsing tree \( W \) with \( N + 1 \) leaves into a parsing tree \( W' \) with \( N \) leaves (see the figure on
next page) by pruning two sibling leaves $w'0$ and $w'1$.

\[
EL(W) = \sum_{w \in W} P(w) |w| = (P(w'0) + P(w'0))(|w'| + 1) - P(w')|w'| + \sum_{w \in W'} P(w) |w|
\]

\[
= EL(W') + P(w')
\]

(5)

where $|w|$ is the length of leave $w$. Note that the interior nodes of $W$ are the interior nodes of $W'$ plus the node $w'$. But conforming to the inductive hypothesis $EL(W')$ is equal to the sum of probabilities of the interior node probabilities in the tree $W'$, therefore the induction step is finished.

- Now the task of building an optimal parsing tree with $N$ leaves (having maximal average parsing length) is equivalent to building a tree where the sum of the probabilities of the $N - 1$ interior nodes is maximal.

- A simple property of the optimal parsing tree: Take the smallest probability of interior nodes

\[
P_{\text{min}} = P(w') = \min_{w \in \text{Int}(W)} P(w)
\]

(6)

Then necessarily all the leaves will have probabilities less or equal $P_{\text{min}}$.

The proof is simple: if a leaf $w$ has the probability larger than $P_{\text{min}}$, one can find another tree having $N$ leaves, where $w$ is an interior node and $w'$ is not an interior node anymore, the sum of probabilities of interior nodes in the new tree being higher than in the original tree.
Proving the Lemma 1

\[ p(w') \]

\[ p(w') p(0) p(w') p(1) \]

TREE W  TREE W'
**Tunstall algorithm**

- **Tunstall algorithm** The construction of the optimal parsing tree follows from the previous property:
  - Start with the elementary tree having only two leaves.
  - Until the number $N$ of leaves is reached, split the leaf having the largest probability.

- For our Bernoulli source with $p(1) = 0.2$ the Tunstall tree construction is the following:
  - Start with $W = \{0, 1\}$, where the leaf $w = 0$ has the largest probability, $p(0) = 0.8$. Split the leaf $w = 0$ to get the new leaves 00 and 01.
  - In the tree $W = \{00, 01, 1\}$ the leaf probabilities are 0.64, 0.16, 0.2. Split the leaf $w = 00$.
  - In the tree $W = \{000, 001, 01, 1\}$ the leaf probabilities are 0.512, 0.128, 0.16, 0.2. Split the leaf $w = 000$.
  - In the tree $W = \{0000, 0001, 001, 01, 1\}$ the leaf probabilities are 0.4096, 0.1024, 0.128, 0.16, 0.2. Split the leaf $w = 0000$.
  - In the tree $W = \{00000, 00001, 0001, 001, 01, 1\}$ the leaf probabilities are 0.3277, 0.0819, 0.1024, 0.128, 0.16, 0.1. Split the leaf $w = 00000$.
  - In the tree $W = \{000000, 000001, 00001, 0001, 001, 01, 1\}$ the leaf probabilities are 0.2622, 0.0655, 0.0819, 0.1024, 0.128, 0.16, 0.2. Split the leaf $w = 000000$. 
The tree $W = \{0000000, 0000001, 000001, 00001, 0001, 001, 01, 1\}$ has 8 leaves, and the leaf probabilities are 
0.2098, 0.0524, 0.0655, 0.0819, 0.1024, 0.128, 0.16, 0.2.

The average parsing length of the final tree is $EL = 3.8515$. The rate of the code is 
$R = \frac{\log_2 N}{EL} = \frac{3}{3.8515} = 0.7789$ bits per symbol, better than the Huffman code for blocks of two symbols, but worse than the Huffman code for blocks of three symbols).

Increasing $N$, the rate of the code will become closer to the entropy, as stated in the following

**Theorem** The rate of the variable to fix code converges to the entropy of the source

$$\lim_{N \to \infty} \frac{\log_2 N}{EL} = h$$  \hspace{1cm} (7)
Tunstall coding without a dictionary: Petry codes

- If the number of codewords in Tunstall code is taken large enough, the code rate is very close to the entropy.

- However, the codewords need to be stored in a data structure (we call it generically dictionary). The dictionary may have as many as $2^{32}$ entries in some applications. It requires a huge memory space.

- Petry codes relax the requirement of storing the codewords. Instead one finds the codeword while parsing the input string, by some arithmetic operations. It can be seen as a variant of arithmetic coding, with fixed size of codewords.

- Assume that $p$ is such that there exist positive integers $a$ and $b$ such that $(1-p)^b = p^a$. The integers $a$ and $b$ can be chosen as the numerator and denominator in the rational approximation of the ratio of logarithms:

$$\frac{\log(1-p)}{\log p} = \frac{a}{b} \quad (8)$$

- Define the sets $V(n)$ for $n = 0, 1, 2, \ldots$

$$V(n) = \{w \in \{0, 1\}^*|am_0(w) + bm_1(w) \geq n; am_0(w^{-1}) + bm_1(w^{-1}) < n\}$$
where \( m_0(w) \) is the number of 0’s in \( w \), \( m_1(w) \) is the number of 1’s in \( w \) and \( w^{-1} \) is the string \( w \) without its last symbol, and \( \{0, 1\}^* \) is the set of all binary strings (of all possible lengths).

- The sets \( V(n) \) are complete (every semi-infinite sequence must have a prefix in the set) and proper (no segment in the set can be a prefix of another segment in the set).

- Theorem: The sets \( V(n) \) are optimal (Tunstall) segment sets for the source with \( Pr\{X_i = 1\} = p \).

- Proof:

  - A Tunstall tree \( W \) has the property that there is a number \( \mu = \min_{w \in \text{Int}W} p(w) \) such that every leaf \( w \) and its parent node (denoted \( w^{-1} \)) satisfy the inequality \( p(w^{-1}) \geq \mu \geq p(w) \) (i.e. the probability of any leaf is smaller or equal to that of the smallest probability of an interior node).

  - For a Bernoulli source with the probability of one \( P(1) = p \), the probability of a string \( w \) with \( m \) symbols, having \( m_0 \) zeros and \( m_1 \) ones is \( p(w) = (1 - p)^{m_0} p^{m - m_0} \) and its logarithm is

\[
\log p(w) = m_0 \log (1 - p) + m_1 \log p = \log (1 - p) \left( m_0 + \frac{\log p}{\log 1 - p} m_1 \right) = \log (1 - p) \left( m_0 + \frac{b}{a} m_1 \right) = \frac{\log (1 - p)}{a} (am_0 + bm_1)
\]
Since every leaf in the Tunstall tree satisfies \( \log p(w^{-1}) \geq \log \mu \geq \log p(w) \), we have \( \frac{\log(1-p)}{a}(am_0(w^{-1}) + bm_1(w^{-1})) \geq \log \mu \geq \frac{\log(1-p)}{a}(am_0(w) + bm_1(w)) \) and finally by dividing by the negative number \( \frac{\log(1-p)}{a} \) we have

\[
am_0(w^{-1}) + bm_1(w^{-1}) \leq \frac{a}{\log(1-p)} \log \mu \leq am_0(w) + bm_1(w) \tag{10}
\]

Since \( am_0(w^{-1}) + bm_1(w^{-1}) \) is integer and is always strictly smaller than \( am_0(w) + bm_1(w) \), one can describe all leaves in a Tunstall-Petri tree as those leaves for which

\[
am_0(w^{-1}) + bm_1(w^{-1}) < n \leq am_0(w) + bm_1(w) \tag{11}
\]

where \( n \) is the same for all leaves. The parameter \( n \) uniquely determines the size of the Tunstall tree (attention! : \( n \) is not the size of the tree).

- Another property of the Tunstall tree: any subtree of a Tunstall tree is a Tunstall tree. This can be proven by observing first the decomposition of a Tunstall tree into three items: the root; the left tree; and the right tree. By construction, the left tree was extended always by splitting the node with highest probability, thus it is a Tunstall tree, and similarly the right tree is a Tunstall tree. Now by successively splitting the left and right trees it results that all subtrees are Tunstall trees.

- Thus a Tunstall tree can be seen as a tree containing as subtrees other Tunstall trees. Thus the cardinality (number of leaves) of a tree will be additively connected to the cardinalities of smaller Tunstall trees.
• Let \( c(n) = |V(n)| \) (the cardinality of the set \( V(n) \)). We have the recursion

\[
c(n) = c(n - a) + c(n - b)
\]

which can immediately be seen from the above mentioned decomposition of the leaves in the Tunstall tree as leaves in the left tree and leaves in the right tree.

With this recursion we can easily build all cardinalities of the sets \( V(n) \), which are necessary in order to build up the table of cardinalities used by encoder and decoder.

**Enumerative coding with Petry codes**

• Let \( a = 1 \) and \( b = 2 \), then \( (1 - p)^2 = p \), \( p = 0.382 \).

\[
V(0) = \emptyset, \quad V(1) = \{0, 1\}, \quad V(2) = \{00, 01, 1\}
\]

• The cardinalities will obey the recursion \( c(n) = c(n - 1) + c(n - 2) \), i.e they will form the Fibonacci string: \( 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 243, \ldots \). Thus, we have \(|V(9)| = 89, \ |V(10)| = 144, \ |V(11)| = 243, \ldots \)

• We will label each node of the Tunstall tree by the cardinality of the tree rooted in that node: see figure on next page.

• By enumerative coding, a leaf will be encoded by the index received when we label all the leaves of the tree, starting from the left-most to the right-most.
Consider encoding and decoding of the string 1001101. From the string, $a_0 n_0(w) + b_1 n_1(w) = 1 \times 3 + 2 \times 4 = 11$, while $a_0 (w^{-1}) + b_1 (w^{-1}) = 1 \times 3 + 2 \times 3 = 9$, so this is a leaf in the Tunstall tree $V(10)$. A part of the tree $V(10)$ is plotted on the figure on next page, specifically the part containing the path of the word to be encoded.

- **Encoding** The question is: by starting to count from the left to the right, what is the index of the leaf corresponding to the string 1001101? We see that in our path from the root to the leaf, we have on the left several subtrees, for which we know the cardinalities. Therefore it is easy to count how many leaves we have on the left of our string, by the rule: whenever in our string we have a bit 1, add the cardinality of the subtree remaining on the left. By adding the framed numbers, we get the index of the string to be $89 + 13 + 5 + 1 = 108$, which, in binary, is the code of our string.

- **Decoding** will be done by comparing the code to the cardinality of the left subtree: if the index is larger, then we decode a 1, otherwise we decode a 0. If a 1 was decoded, we subtract from the index the cardinality of the subtree on the left. The process continues until the index remains 0.

- Coding and decoding can be done using just a linear array (no tree is needed to be stored), as explained in the two pages provided in the course box on 4’th floor.
Path in the Tunstall tree for the parsing string 1001101
(not the entire Tunstall tree is represented).
Efficient implementations of Variable-to-fixed codes

History:

1. Tunstall
   - Schalkwijk, Antonino, Petry 1972. multidimensional array
   - Schalkwijk 1981. recursive, linear array
   - Tjalkens, Willems, 1987. recursive, linear array

2. BAC (Boncelet 1993) Block arithmetic coding

3. AB, (Teuhola, Raita 1994), Arithmetic block coding