Test algorithms and Tunstall coding

Test algorithms

Assume manufacturing transistors of 6 types $u_1, \ldots, u_6$ with the probability distribution :

u	u_1	u_2	u_3	u_4	u_5	u_6
$f_U(u)$	0.30	0.20	0.20	0.20	0.05	0.05

Assume we have **seven** binary tests $T_1, T_2, \ldots, T_7$ to identify transistor types.

GOAL : Use minimal number of tests $T_1, T_2, \ldots, T_7$ to identify transistor types.

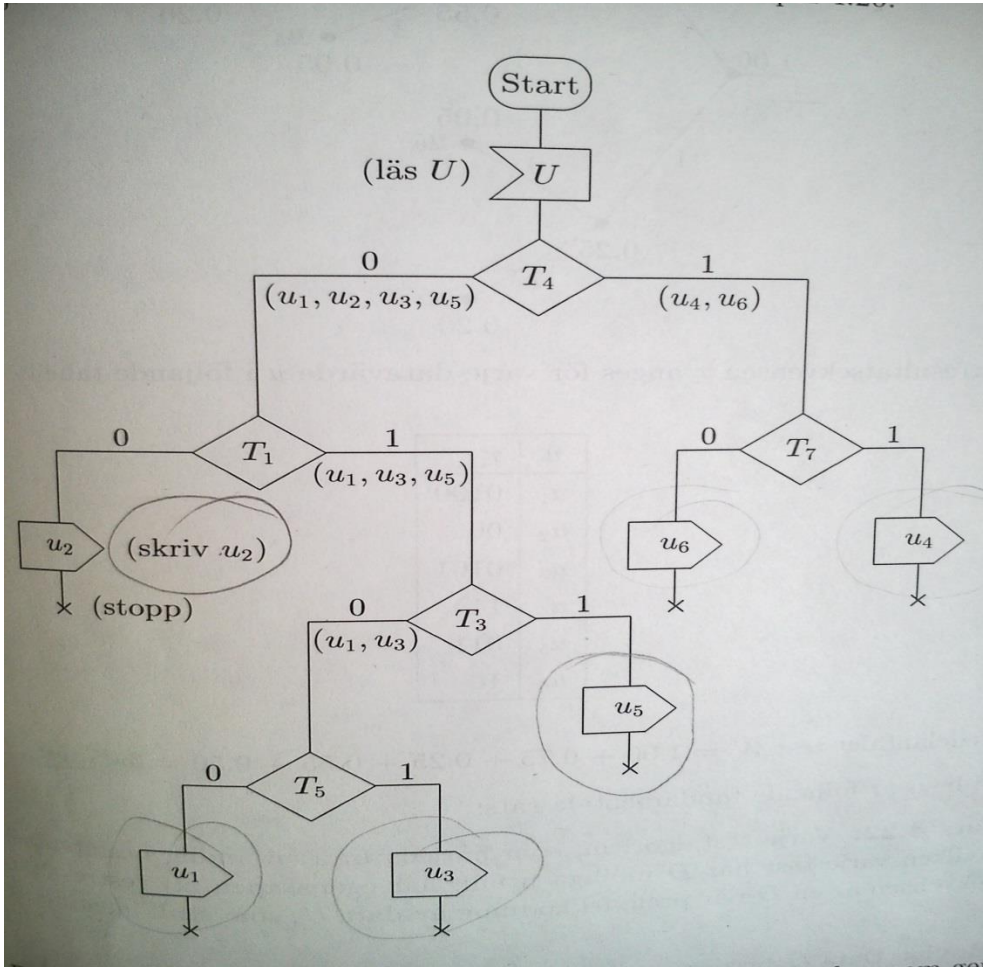
Test specification

	u_1	u_2	u_3	u_4	u_5	u_6	
T_1	1	0	1	0	1	0	
T_2	0	0	0	0	0	0	useless
T_3	0	0	0	1	1	1	
T_4	0	0	0	1	0	1	
T_5	0	0	1	1	0	1	complem. to T_6
T_6	1	1	0	0	1	0	complem, to T_5
T_7	1	0	0	1	0	0	important u_4 *vs.* u_6

- U can be identified IF AND ONLY IF all columns are DISTINCT !
- Test algorithm always uniquely identifies U
- Trivial test algorithm : use ALL available tests (in our case $T_1, \ldots, T_7$)

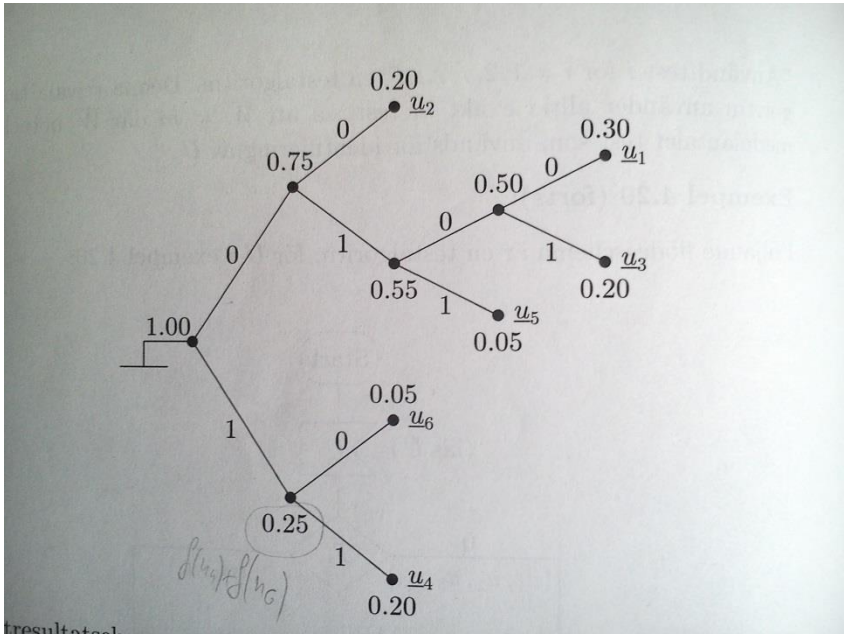
Test algorithm - example

- Can we do better ?



Test algorithms -comments

- Notice the absence of T_2 and T_6 !
- The "cost" of any test is the same - we are minimizing average number of tests.
- Encoding is straightforward via RV X - but tests are not spcified !



u	x
u_1	0100
u_2	00
u_3	0101
u_4	11
u_5	011
u_6	10

Average number of tests $\overline{W} = 1 + 0.75 + 0.25 + 0.55 + 0.5 = 3.05$

Main result

Theorem

Every test algorithm (which uniquely identifies U) with D possible outcomes has the property that **test result sequence** *is D-ary prefix-free coding of data U.*

Proof.

- Data U is uniquely determined by test result sequence $X_1 X_2 \ldots X_W$!
- If result is $X_1 X_2 \ldots X_i$ then stop, no other codeword with prefix $X_1 X_2 \ldots X_i$ can be constructed !

□

Hufman coding bounds

- Test algorithm is a prefix-free coding of U - use our theory !
- Easy to compute $H(U) = 2.345$ for the source U.
- In general,

$$\overline{W} \geq \frac{H(U)}{\log_2 D} = H(U) = 2.345.$$

- Can we find better bound ? YES, Huffman coding for U gives :

u	x
u_1	10
u_2	00
u_3	01
u_4	110
u_5	1110
u_6	1111

$$\overline{W}_{Huf} = 2 \times (0.3 + 0.2 + 0.2) + 3 \times 0.2 + 4 \times (0.05 + 0.05) = 2.40.$$

First order optimal test (FOOT) algorithms

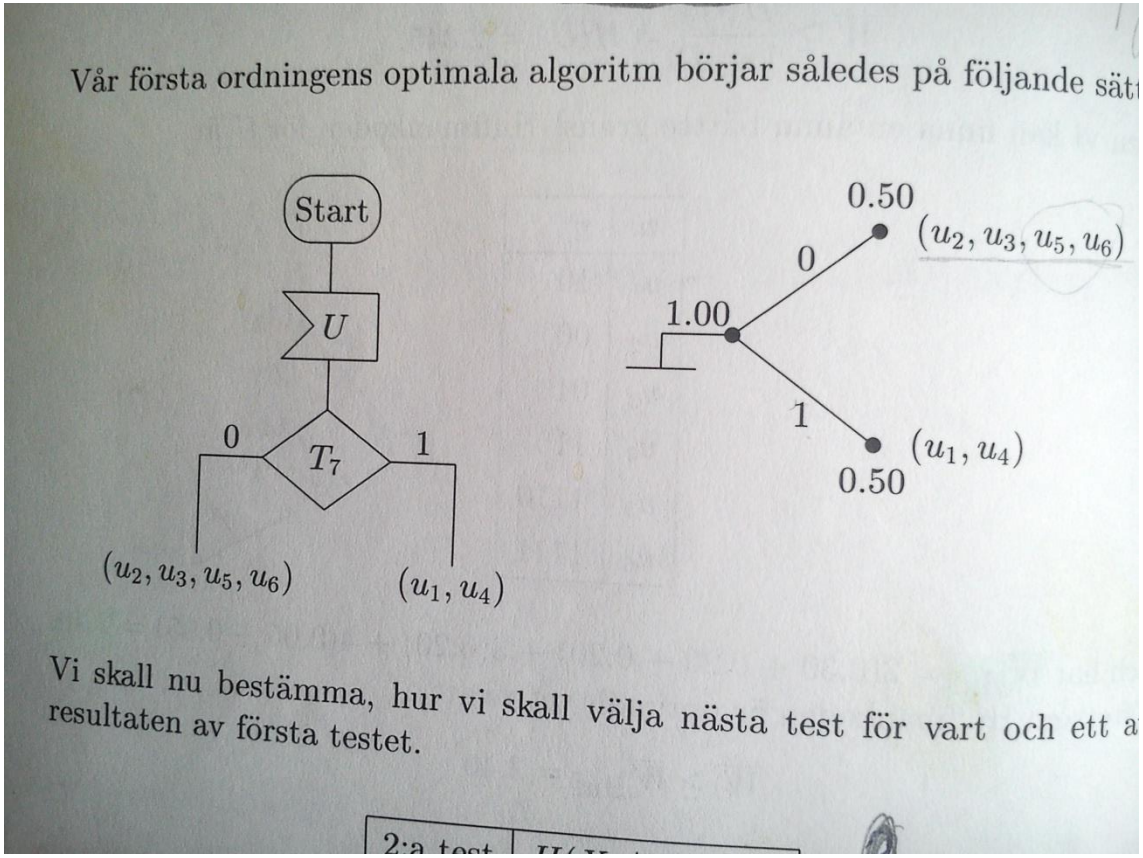
- Can we use Huffman coding for the previous scheme ?
- **NO**, for instance a test is needed which gives result 0 for u_2 and u_3 and 1 for remaining types. No such test exists !
- Huffman coding given above is not unique - but any Huffman coding for this U requires some nonavailable test !
- Can we get close to Huffman average length somehow ?

YES, the answer is **first order optimal test algorithm**.

In each step choose the test that maximizes the entropy - maximizes the information obtained by the next test !

FOOT algorithm - example

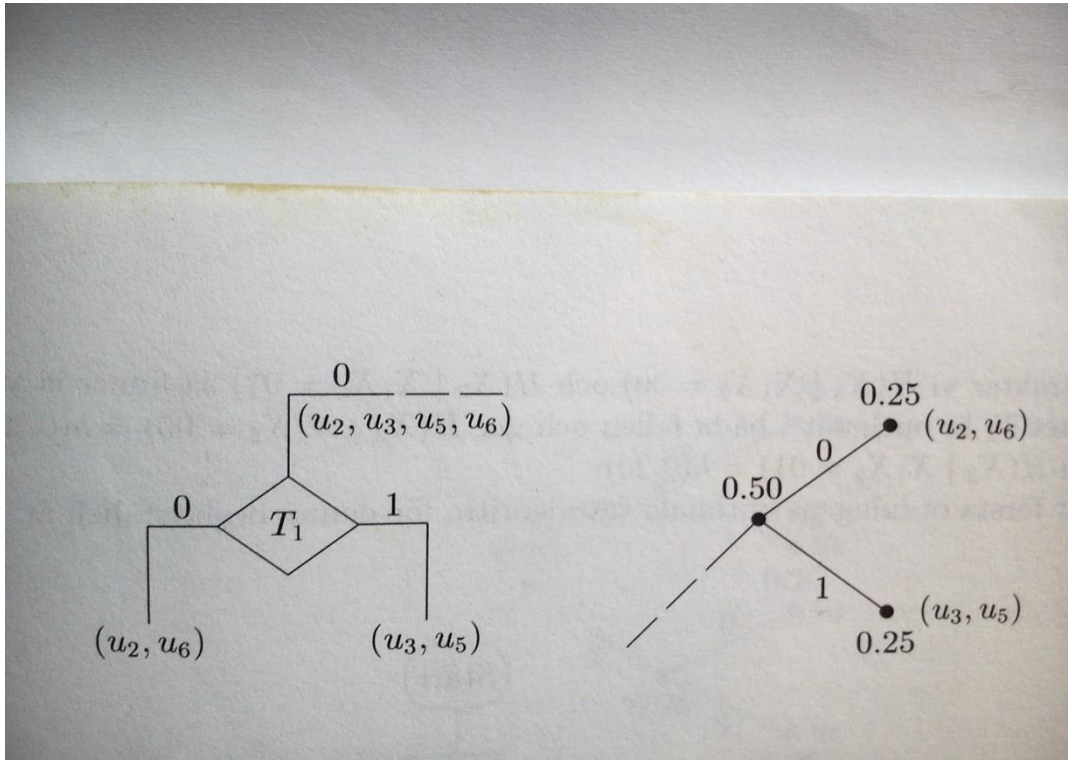
1st test	$H(X_1)$
T_1	$h(0.45)$
T_3	$h(0.30)$
T_4	$h(0.25)$
T_5	$h(0,45)$
T_7	$h(0.50)$



Choosing second test

- Compute the entropy for all remaining tests conditioned on the results of the first test.

2nd test	$H(X_2 \mid X_1 = 0)$
T_1	$h(0.50)$
T_3	$h(0.20)$
T_4	$h(0.10)$
T_5	$h(0,45)$



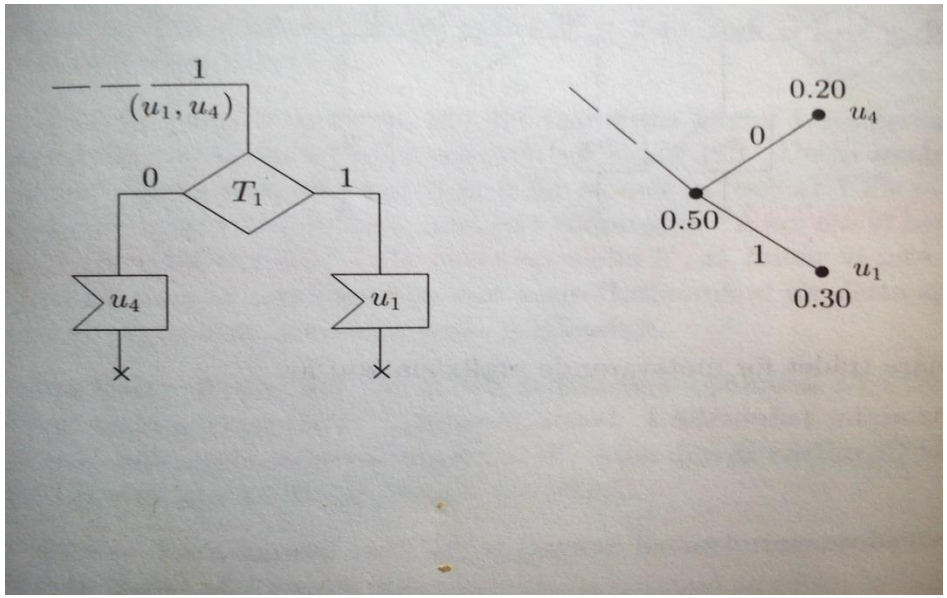
Either choose T_1 or T_5, we take T_1 !

Choosing second test II

We consider now $H(X_2|X_1 = 1)$.

2nd test	$H(X_2|X_1 = 1)$
T_1	$h(0.40)$
T_3	$h(0.40)$
T_4	$h(0.40)$
T_5	$h(0.40)$

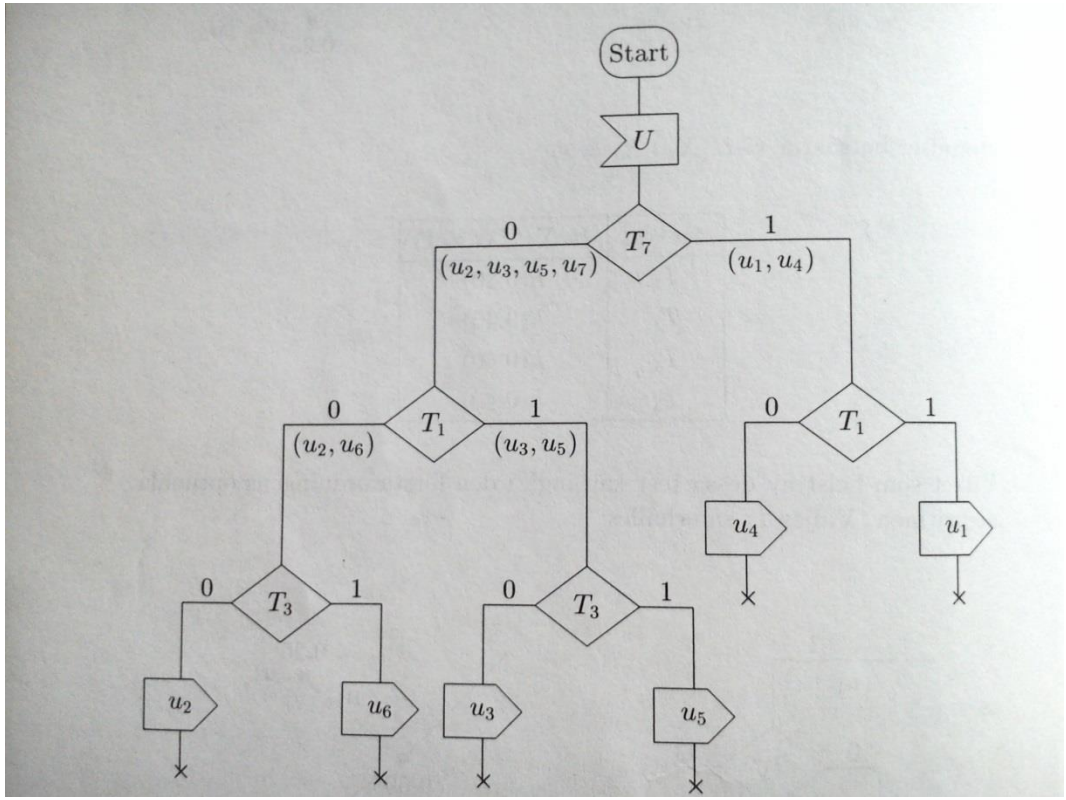
Choose any of the tests, we take T_1 !



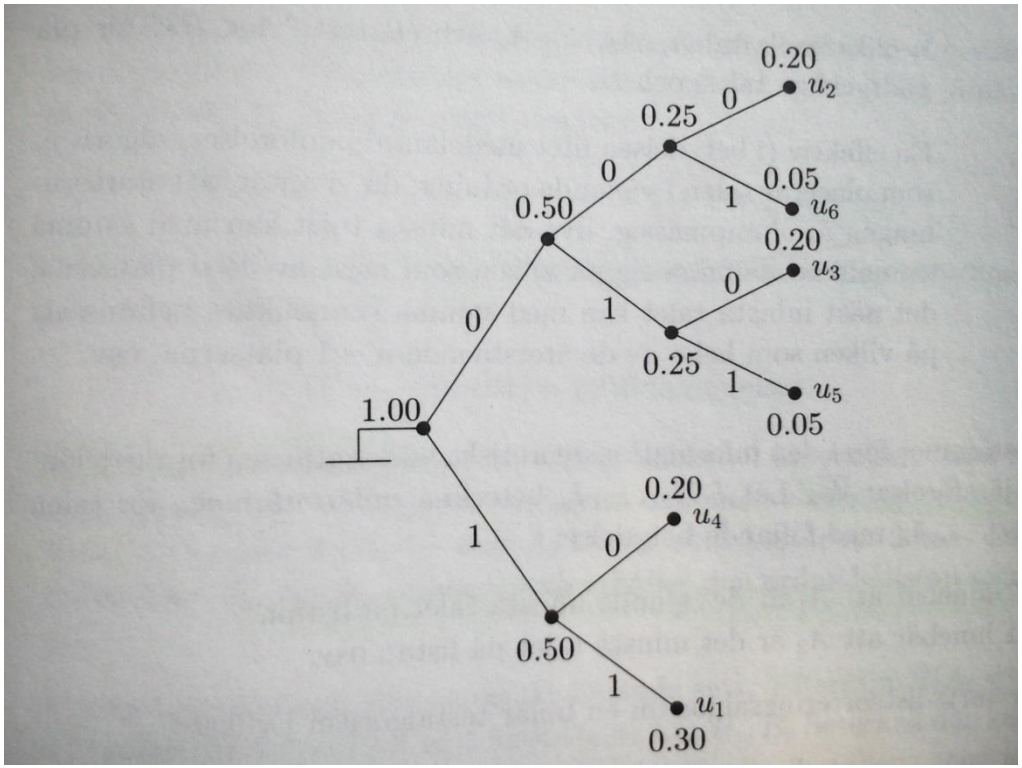
Selecting third test

If we now consider $H(X_3|X_1X_2 = 00)$ and $H(X_3|X_1X_2 = 01)$ we find that T_3 is optimal in both cases and

$$H(X_3|X_1X_2 = 00) = H(X_3|X_1X_2 = 01) = h(0.20).$$



Coding and average nmb. of tests



$$\overline{W} = 1 + 0.5 + 0.5 + 0.25 + 0.25 = 2.50!!$$

Very close to Huffman lower bound 2.40 !

Variable-to-fix length coding

Previously we only considered the situation where we encoded N source symbols into variable length code sequences for a fixed value of N. We could call this "'fixed length to variable length"' encoding. But another possibility exists. We could encode variable length source sequences into fixed or variable length code words.

Example 1: Consider the DMS source $\{A, B\}$ with probabilities $(.9, .1)$ and the following code book

Source Sequences	Codewords
B	00
AB	01
AAB	10
AAA	11

Average length of source phrase

$= 1 \times .1 + 2 \times .09 + 3 \times (.081 + .70) = 2.75$

Average # of code symbols/source symbols $= \frac{2}{2.71} = 0.738$

Variable-to-fix length coding II

The idea for variable-to-fix source coding has several motivations:

- The codewords are of the same length (no need for buffering)
- The source encoder is split into two parts : Parser and Encoder

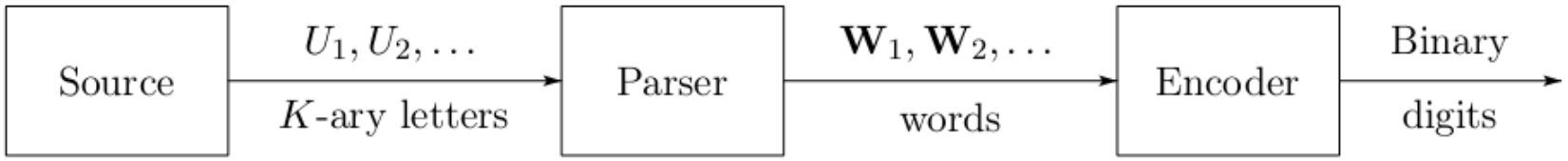
Consider the sequence

$$abaacbaabacaaabc...$$

A **parser** segments the sequence of letters from a source into a concatenation of words.

For instance, $ab, aac, b, aab, ac, aaa, b, c, ...$ may be the valid words after parser.

Block diagram



- Parser does its job with the help of a dictionary of size M.
- Such a dictionary is a collection of words $\mathbf{w}_1, \ldots, \mathbf{w}_M$, where each $\mathbf{w_m}$ is a sequence of source letters
- We assume $M \leq 2^b$ and then map each dictionary entry into a binary b-tuple

Given a dictionary of strings $\mathbf{w}_1, \ldots, \mathbf{w}_M$, the parser picks the longest prefix of the source sequence $U_1, U_2, \ldots$ that is in the dictionary (say, $U_1, \ldots, U_L$) and then continues with $U_{L+1}, U_{L+2} \ldots$.

Tunstall codes

Tunstall Codes are U.D. Variable- to Fixed- length codes with binary code words

Basic Idea – Encode into binary code words of fixed length L, make 2^L source phrases that are as nearly equally probable as we can

We do this by making the source phrases as leaves of a tree and always splitting the leaf with the highest probability.

Tunstall code - example

Example 2: (A, B, C, D) with $(p_1, p_2, p_3, p_4) = (.5, .3, .1, .1)$

Source Symbol	Code word	Source Symbol	Code word
D	0000	C	0001
BB	0010	AB	1001
BC	0011	AC	1010
BD	0100	AD	1011
BAA	0101	AAA	1100
BAB	0110	AAB	1101
BAC	0111	AAC	1110
BAD	1000	AAD	1111

Average length of source phrase = Sum of probabilities of internal nodes) $= 1 + .5 + .3 + .25 + .15 = 2.2$

Average number of code symbols/source symbols $= 4/2.2 = 1.82$

Improved Tunstall coding

- Since the phrases are not equally probable, one can use a *Huffman Code* on the phrases.
- The result is encoding a variable number of source symbols into a variable number of code symbols.

Example 3: Back to Example 1 with (A, B) with $(.9, .1)$

We have seen that Tunstall alone $I = 2.71$ $Av = \frac{2}{2.71} = .738$

Source Phrases	Tunstall Code	Probability	Improved Tunstall (Huffman)
AAA	11	0.729	0
B	00	0.1	11
AB	01	0.09	100
AAB	10	0.081	101

Average # of code symbols per source symbol

$$= \frac{(1+.271+.171)}{(1+.9+.81)} = \frac{1.442}{2.71} = .532$$

Summary of results for Example 1

All of the following use 4 code words in coding table:

1. Huffman Code, $N = 2$

$$\begin{aligned} AA &\longrightarrow 0 \\ AB &\longrightarrow 11 \\ BA &\longrightarrow 100 \\ BB &\longrightarrow 101 \end{aligned}$$

2. Shannon-Fano Code $N = 2$

$$\begin{aligned} AA &\longrightarrow 0 \\ AB &\longrightarrow 10 \\ BA &\longrightarrow 110 \\ BB &\longrightarrow 111 \end{aligned}$$

Summary of results for Example 1 cont.

1. Tunstall Code

$$
\begin{aligned}
B &\longrightarrow 00 \\
AB &\longrightarrow 01 \\
AAB &\longrightarrow 10 \\
AAA &\longrightarrow 11
\end{aligned}
$$

2. Tunstall/Huffman

$$
\begin{aligned}
B &\longrightarrow 11 \\
AB &\longrightarrow 100 \\
AAA &\longrightarrow 0 \\
AAB &\longrightarrow 101
\end{aligned}
$$

Valid and prefix-free dictionaries

Example: Assume the alphabet $\mathcal{U} = \{a, b\}$ and the dictionary $w_1 = a, w_2 = b, w_3 = bba, w_4 = bbb$. Then, the sequence

$$abbababba...$$

is parsed as

$$a, bba, b, a, bba, \ldots$$

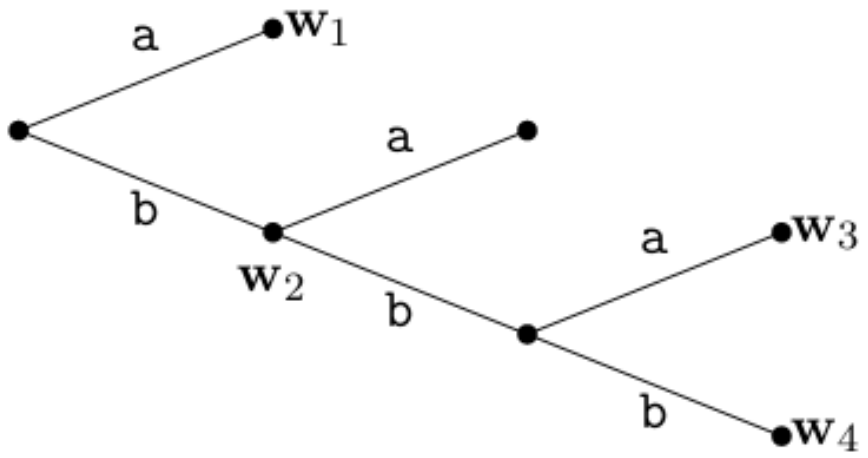
Assuming e.g. encoding $a = 00$, $b = 01$, $bba = 10$ and $bbb = 11$.

Definition

A dictionary $\mathbf{w}_1, \ldots, \mathbf{w}_M$ *is* **valid** *if every infinite sequence of source letters has a prefix in the dictionary.*

Valid and prefix-free dictionaries II

If the source alphabet U contains K letters, we can associate a unique K-ary tree with a dictionary $\mathbf{w}_1, \ldots, \mathbf{w}_M$:



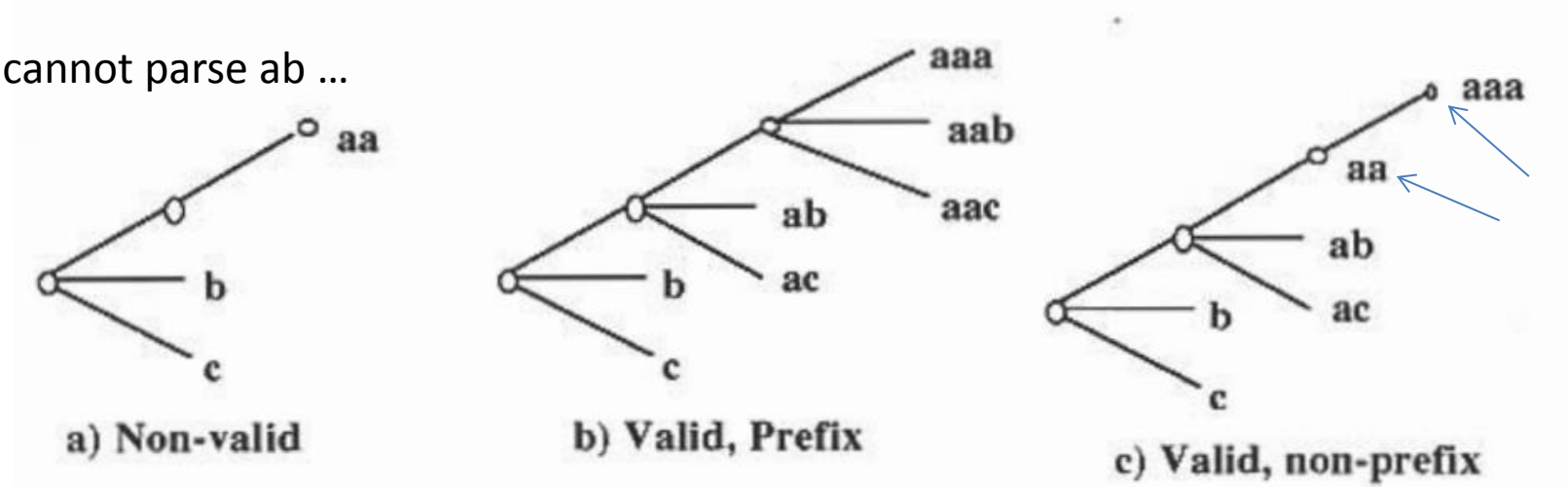
Valid and prefix-free dictionaries III

From the definition, we see that a dictionary is **valid**, if and only if there is at least one dictionary word on every path that connects the root to a leaf.

Definition

A dictionary is **prefix-free**, *if no dictionary word is a prefix of another. Such a dictionary is also called* **instantaneously encodable**.

cannot parse ab …

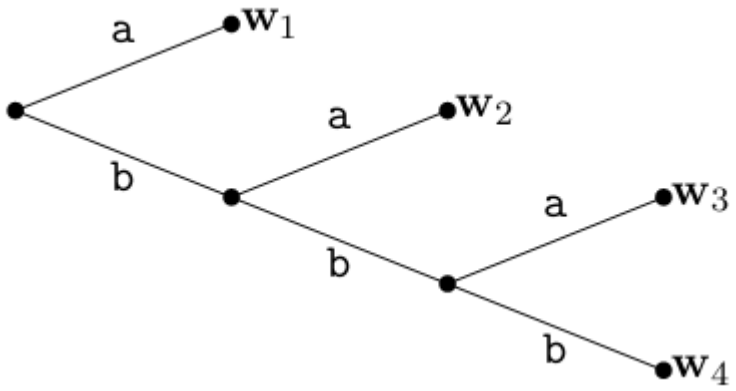


Example

Example 2: The dictionary in our Example is not prefix-free. On the other hand, the dictionary consisting of the words

$$\mathbf{w}_1 = a, \mathbf{w}_2 = ba, \mathbf{w}_3 = bba, \mathbf{w}_4 = bbb$$

is valid and prefix-free.



From the definition we see that a words of a valid and prefix-free dictionary are all the leaves (and only the leaves) of a K-ary tree.

Building a tree for valid prefix-free dictionary

- Codewords (after encoder) are of fixed length b we want to maximize $E[W]$ of dictionary words for given M
- Need to build up the tree of a valid prefix-free dictionary of size M.
- For a K-ary source alphabet each time we convert a leaf to an intermediate node, we replace it with K leaves that are its children.
- But we lose this node, so $K - 1$ leaves are added each time a new intermediate node is made.

This gives a relation

$$M = 1 + q(K - 1)$$

where 1 corresponds to the root, and q is the depth of the tree.

Computing parameters

To use a valid, prefix-free dictionary, and represent dictionary words with binary strings of b bits, we choose q such that

$$M \leq 2^b, \quad M = 1 + q(K-1),$$

i.e., we make M as large as possible without exceeding 2^b.

If M is given then we can find smallest b and q !

Example

What is the depth of the tree when $M = 5$ for $K = 3$ and $K = 2$, respectively ?

Compute $b = 3$ so that $2^b \geq M$ and $M = 1 + q(K-1)$ is satisfied.

This gives $q = 2$ when $K = 3$. For $K = 2$ (binary source alphabet) we get $q = 4$.

Tunstall coding

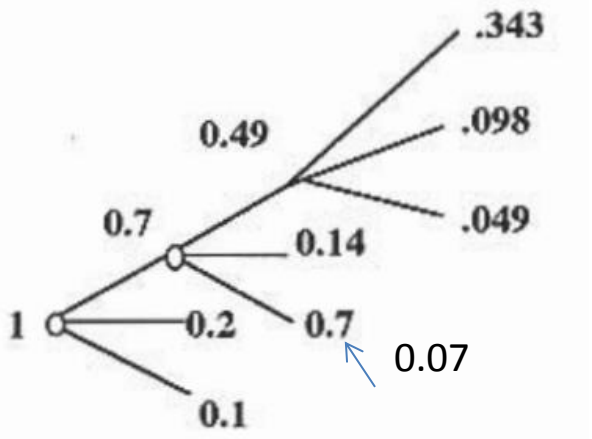
By maximizing $E[W]$ we minimize $b/E[W]$, the number of binary digits per average number of source letters encoded !!

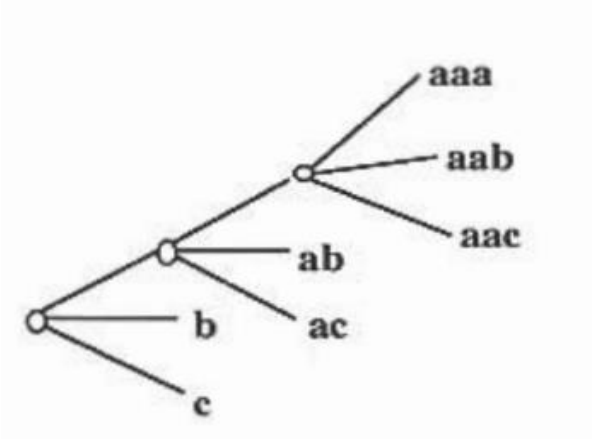
Example

Assume alphabet is $\{a, b, c\}$ and $P(a) = 0.7$, $P(b) = 0.2$ and $P(c) = 0.1$. Our valid prefix-free dictionary of $M = 7$ words **will be**

$$\mathcal{W} = \{b, c, ab, ac, aaa, aab, aac\}$$

It gives $E[W] = 1 + 0.7 + 0.49 = 2.19$.





Tunstall algorithm

How did we get this tree ? Answer: Tunstall algorithm !!

Tunstall algorithm:

1) Start with the root as an intermediate node and all level 1 nodes as leaves.
2) Pick the highest probability leaf, make it intermediate, and grow K leaves on it.
3) If the number of leaves < M, goto step 2 else stop.

Overview for source compression

- **Data compression:**
  - Minimize the size of information representation.
  - Reduce the *redundancy* of the original representation.
- **Purposes:**
  - Save storage space.
  - Reduce transmission time.
- **Basic approaches:**
  - *Lossless* compression: decompression into exactly the original form (typical for text).
  - *Lossy* compression: decompression into approximately the original form (typical for signals and images).

Overview II

- **Phases of data compression:**
  - *Modelling* of the source
  - *Source encoding* (called also *entropy coding*), using the model
- **Other viewpoints:**
  - Speed of compression / decompression
  - Size of the model
- **Classification by lengths of coding units:**
  - *Fixed-to-fixed* coding
  - *Variable-to-fixed coding*
  - *Fixed-to-variable coding*
  - *Variable-to-variable coding*

Examples of models

1. Character distribution

Char	Prob
A	0.10
B	0.05
C	0.08
D	0.06
E	0.15
…..	…..

2. Successor distribution

Char	Succ	Prob
A	A	0.01
A	B	0.20
A	C	0.10
A	D	0.25
…..	…..	……
B	A	0.15
B	B	0.02
B	C	0.01
B	D	0.01
…..	…..	…..

3. Dictionary

Word	Prob
ALL	0.02
ALWAYS	0.01
ARE	0.05
AS	0.03
AT	0.02
BASIC	0.01
BEGIN	0.01
…..	…..

Overview III

- **Main classes of text compression methods:**
  - *Dictionary* methods
  - *Statistical* methods
- **Classification based on availability of the source:**
  - *Off-line methods*
  - *On-line methods*
- **Classification based on the status of the model:**
  - *Static methods*
  - *Semiadaptive methods*
  - *Adaptive methods*
- **Measurement of compression efficiency:**
  - Compression ratio: Source size / compressed size
  - Bits per source symbol (character, pixel, etc.)

Illustration of a static method

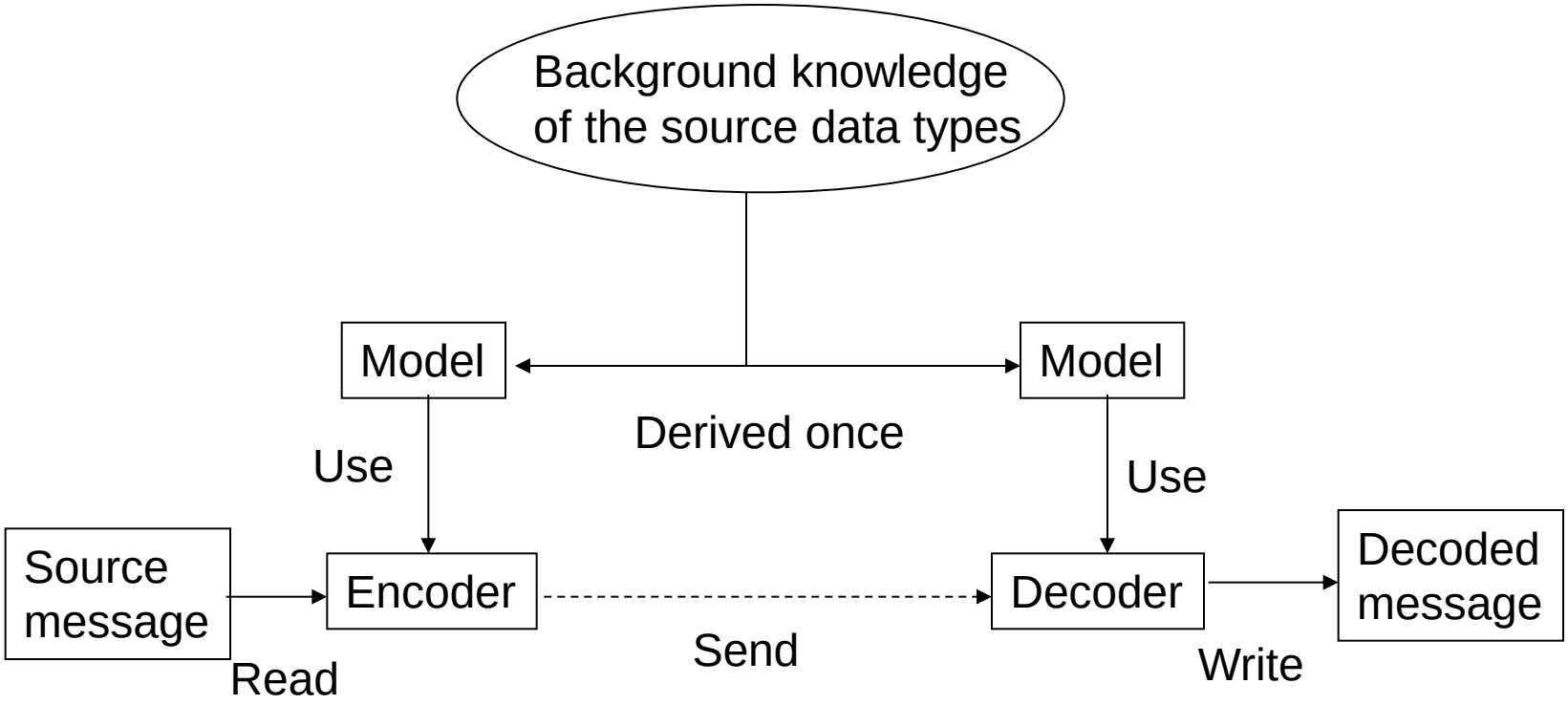


Illustration of a semiadaptive method

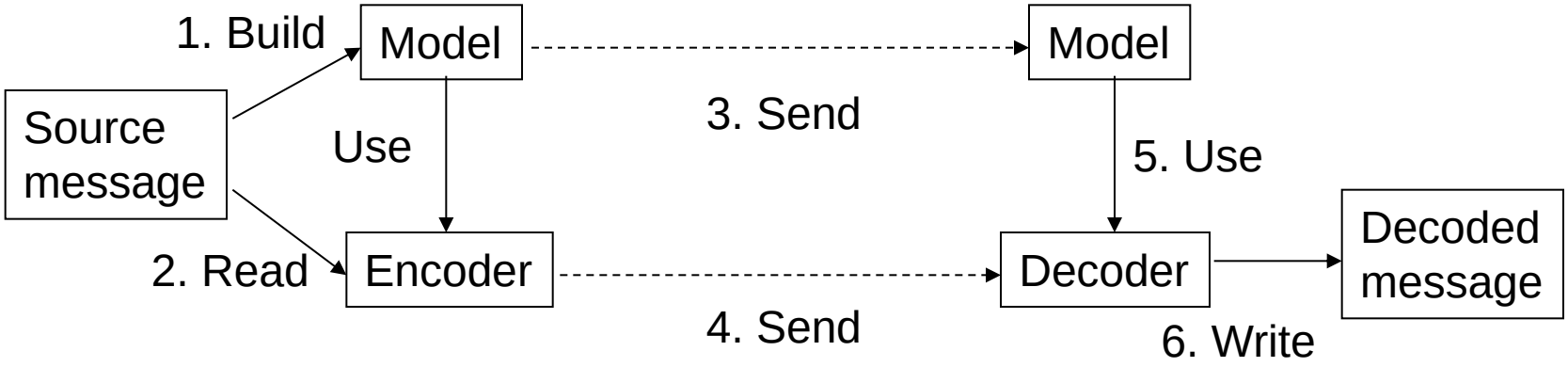


Illustration of an adaptive method

Models are updated dynamically, based on the already processed part of the source, known to both encoder and decoder.