FUNDAMENTALS OF INFORMATION SCIENCE

Shandong University
2025 Spring

Lecture 2.4: Source Coding (Data Compression)

A Example

p_i	Code 1	Code 2
$1/2$	000	0
$1/4$	001	10
$1/8$	010	110
$1/16$	011	1110
$1/64$	100	111100
$1/64$	101	111101
$1/64$	110	111110
$1/64$	111	111111
El_i	3	2

$$H(X) = -\sum p_i \log p_i = 2\text{bits}$$

How to find the best code?

Source Codes

- Source code C for a random variable X is

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

$\mathcal{D}^*$: set of finite-length strings of symbol from D-ary alphabet $\mathcal{D}$

- Code length: $l(x)$

- Example: $C(\text{red}) = 00$, $C(\text{blue}) = 11$, $\mathcal{X} = \{\text{red, blue}\}$, $\mathcal{D} = \{0, 1\}$

Source Codes

Source coding applications

- Magnetic recording: cassette, hardrive, USB...
- Speech compression
- Compact disk (CD)
- Image compression: JPEG

Still an active area of research:

- Solid state hard drive
- Sensor network: distributed source coding

Source Codes

What defines a good code

- Non-singular:
$$x \neq x' \Rightarrow C(x) \neq C(x')$$

- non-singular enough to describe a single RV X

- When we send sequences of value of X, without "comma" can we still uniquely decode

- Uniquely decodable if extension of the code is nonsingular

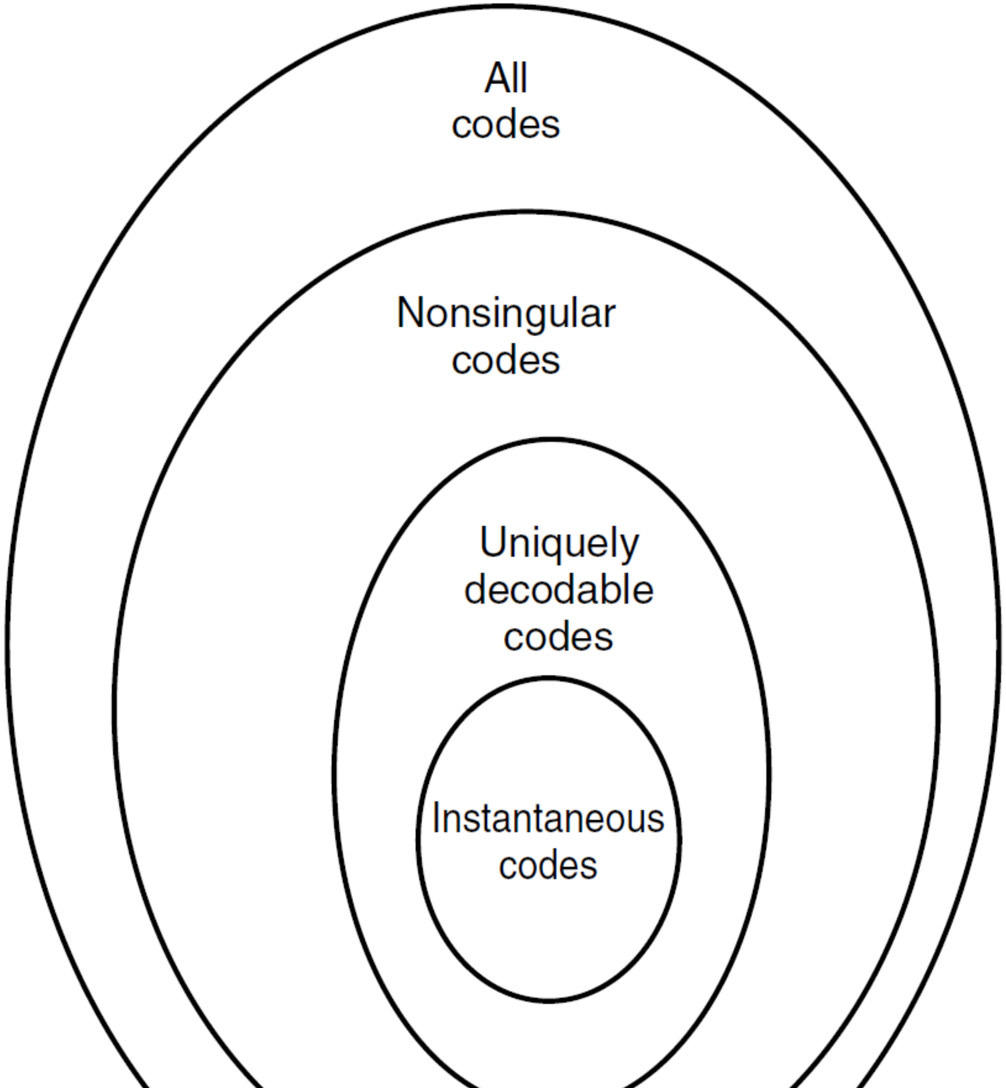
$$C(x_1)C(x_2)\cdots C(x_n)$$

Source Codes

X	Singular	Nonsingular not uniquely decodable	Uniquely decoable	Prefix
1	0	0	10	0
2	0	010	00	10
3	0	01	11	110
4	0	10	110	111

- Uniquely decodable if only one possible source string producing it
- However, we have to look at entire string to determine
- Prefix code (instantaneous code): no codeword is a prefix of any other code

Source Codes



Source Codes

Expected code length

- Expected length $L(C)$ of a source code $C(x)$ for X with pdf $p(x)$

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

- We wish to construct instantaneous codes of minimum expected length

Kraft Inequality

- By Kraft in 1949
- Coded over alphabet size D
- m codes with length $l_1, \ldots, l_m$
- The code length of all instantaneous code must satisfy Kraft inequality

$$\sum_{i=1}^{m} D^{-l_i} \leq 1$$

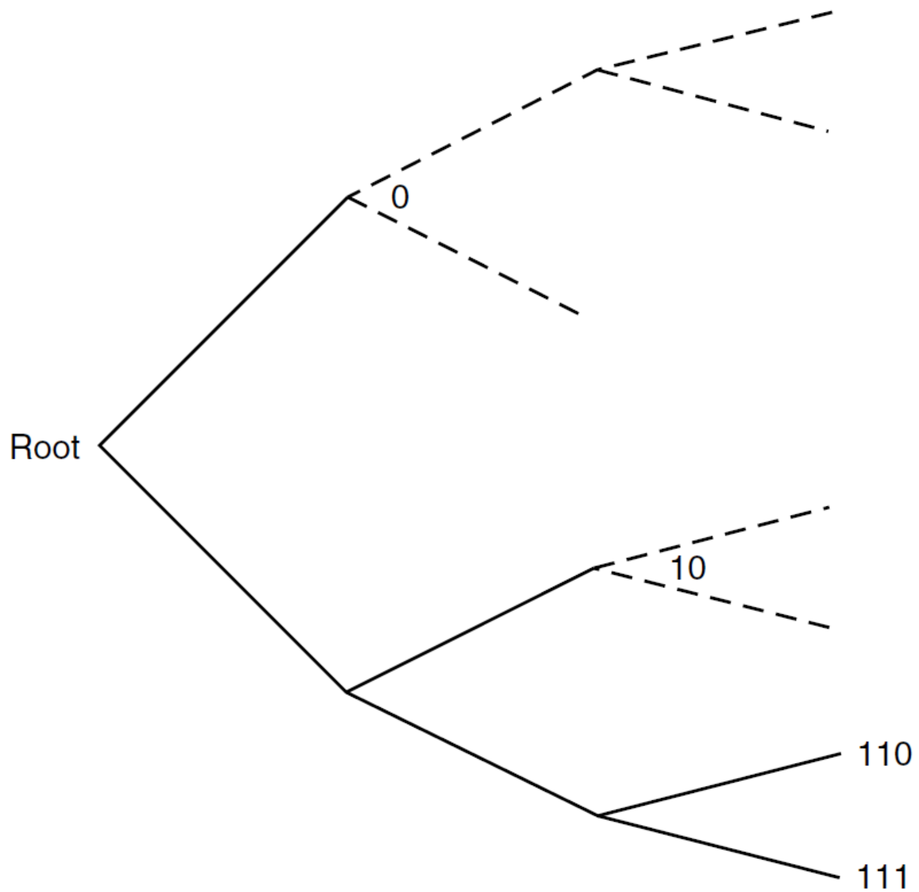
- Given $l_1, \ldots, l_m$ satisfy Kraft, can construct instantaneous code
- Can be extended to uniquely decodable code (McMillan inequality)

Kraft Inequality

Proof of Kraft inequality

- Consider D-ary tree
- Each codeword is represented by a leaf node
- Path from the root traces out the symbol
- Prefix code: no codeword is an ancestor of any other codeword on the tree
- Each code eliminates its descendants as codewords

Kraft Inequality



Kraft Inequality

- $l_{\max}$ be the length of longest codeword
- A codeword at level l_i has $D^{l_{\max}-l_i}$ descendants
- Descendant sets must be disjoint:

$$\sum D^{l_{\max}-l_i} \leq D^{l_{\max}}$$

$$\Rightarrow \sum D^{-l_i} \leq 1$$

- Converse: if $l_1, \ldots, l_{\max}$ satisfy Kraft inequality, can label first node at depth l_1, remove its descendants...
- Can extend to infinite prefix code $l_{\max} \to \infty$

Optimal Expected Code Length

- One application of Kraft inequality
- Expected code length of D-ary is lower bounded by entropy:

$$L \geq H_D(X)$$

Proof:

$$L - H_D(X) = \sum p_i l_i - \sum p_i \log_D \frac{1}{p_i}$$
$$= D(p||r) + \log_D \frac{1}{c} \geq 0$$
$$r_i = D^{-l_i} / \sum_j D^{-l_j}, \quad c = \sum D^{-l_i} \leq 1$$

Summary

- Nonsingular $>$ Uniquely decodable $>$ Instantaneous codes
- Kraft inequality for Instantaneous code
- Entropy is lower bound on expected code length

Lecture 2.5: Shannon Codes and Huffman Codes

Optimal Code Length

- Encode source X with pdf p_i
- Find code length l_i to minimize expected code length L
- Uniquely decodable code should satisfy Kraft-McMillan inequality

$$\begin{aligned} \text{minimize}_{l_i} \quad & \sum_{i=1}^{m} p_i l_i \\ \text{subject to} \quad & \sum_{i=1}^{m} D^{-l_i} \leq 1. \end{aligned}$$

Optimization with Constraints(KKT conditions)

$$\begin{aligned}&\text{minimize } f(x)\\&\text{subject to } h_1(x)=0, h_2(x)=0, \dots, h_m(x)=0\\&\qquad\qquad g_1(x)\le 0, g_2(x)\le 0, \dots, g_r(x)\le 0\end{aligned}$$

Its Lagrangian function

$$L(x,\lambda,u)=f(x)+\sum_{i=1}^{m}\lambda_{i=1}^{m}h_i(x)+\sum_{j=1}^{r}u_j g_j(x).$$

The optimal solution x* satisifies

$$\begin{aligned}&\nabla_x L(x^*,\lambda^*,u^*)=0,\\&u_j^*\ge 0,\quad j=1,\dots,r.\\&u_j^*=0,\quad \forall g_j(x)<0.\end{aligned}$$

Shannon Codes

- solution to optimal code length problem $l_i^* = -\log_D p_i$
- consider code length $l_i = -\lceil \log_D p_i \rceil$
- this satisfies Kraft inequality

$$\sum D^{-\lceil \log_D (1/p_i) \rceil} \leq \sum D^{\log_D p_i} = \sum_i p_i = 1$$

- we can construct a instantaneous code from Kraft

Shannon Codes

- expected code length of roof code

$$\sum p_i \lceil \log_D(1/p_i) \rceil < \sum p_i (\log_D(1/p_i) + 1) = H_D(X) + 1$$

- we have shown that $L \geq H_D(X)$ ($D(p||q) \geq 0$ and Kraft inequality)
- the expected code length of roof code at most one bit more than optimal code
- optimal code must be better than roof code.
- expected length of optimal code L^*

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

Shannon Codes

Sometimes "roof" can be quite bad

- code a biased coin flip with $p = 1/4$ using binary code
- $l_1 = \lceil \log_2 1/p \rceil = 2$, $l_2 = \lceil \log_2 1/(1-p) \rceil = \lceil 0.415 \rceil = 1$
- $C(1) = 01$, $C(2) = 1$, $L = 1.25$
- but obviously $C(1) = 0$ and $C(2) = 1$ is better, with $L = 1$
- if send 100 symbols using roof code, we will use $0.25 \times 100 = 25$ extra bits, that is 25% more than needed!

Huffman Codes (1952)

- The optimal (shortest expected length) prefix code for a given distribution
- $H(X) \leq L < H(X) + 1$

David Huffman, 1925 - 1999

(David Huffman, in term paper for MIT graduate class, 1951)

Huffman Codes

- Start from small probabilities
- Form a tree
- Assign 0 to higher branch, 1 to lower branch

Huffman Codes

- Binary alphabet $D = 2$
- Expected code length

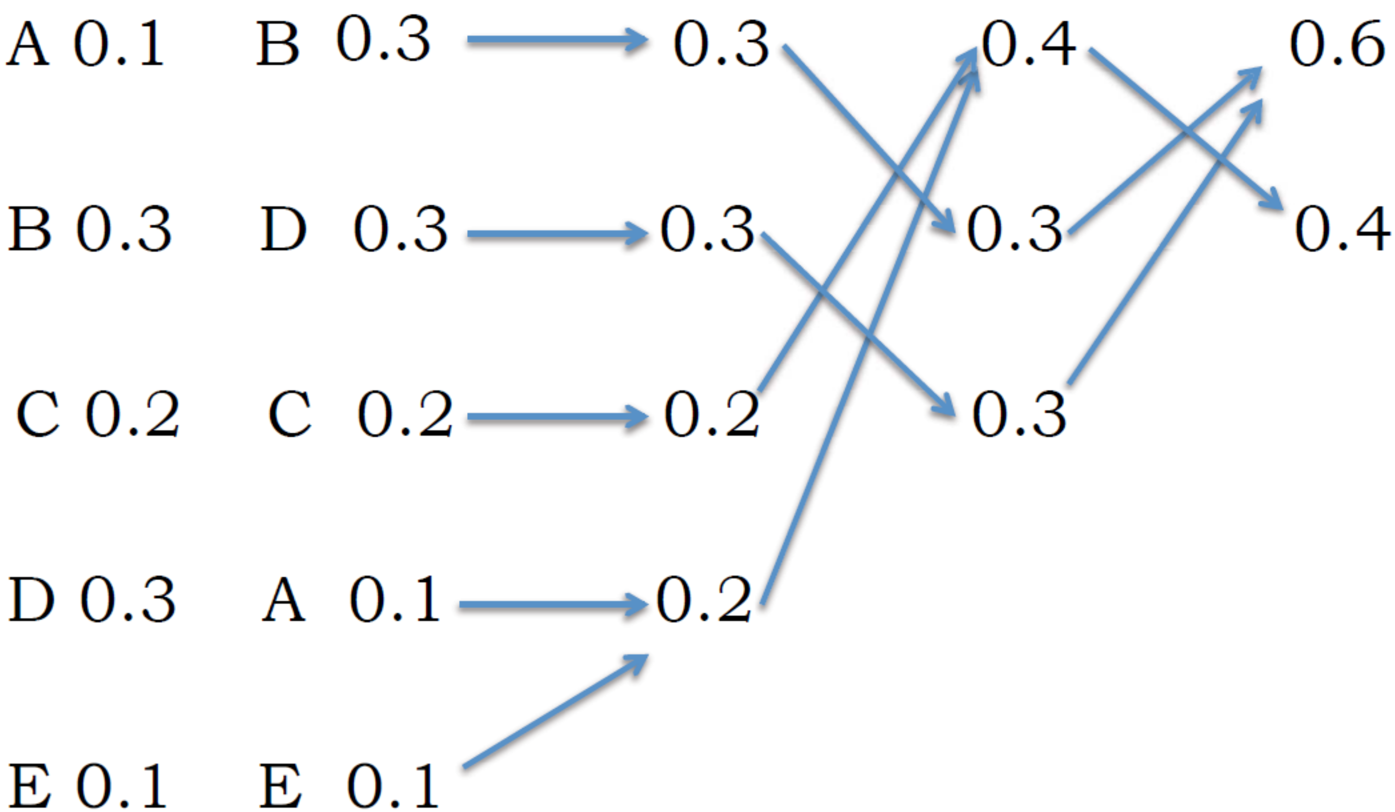
$$L = \sum p_i l_i = (0.25 + 0.25 + 0.2) \times 2 + 3 \times 0.3 = 2.3$$

- Entropy $H(X) = \sum p_i \log(1/p_i) = 2.3$ bits

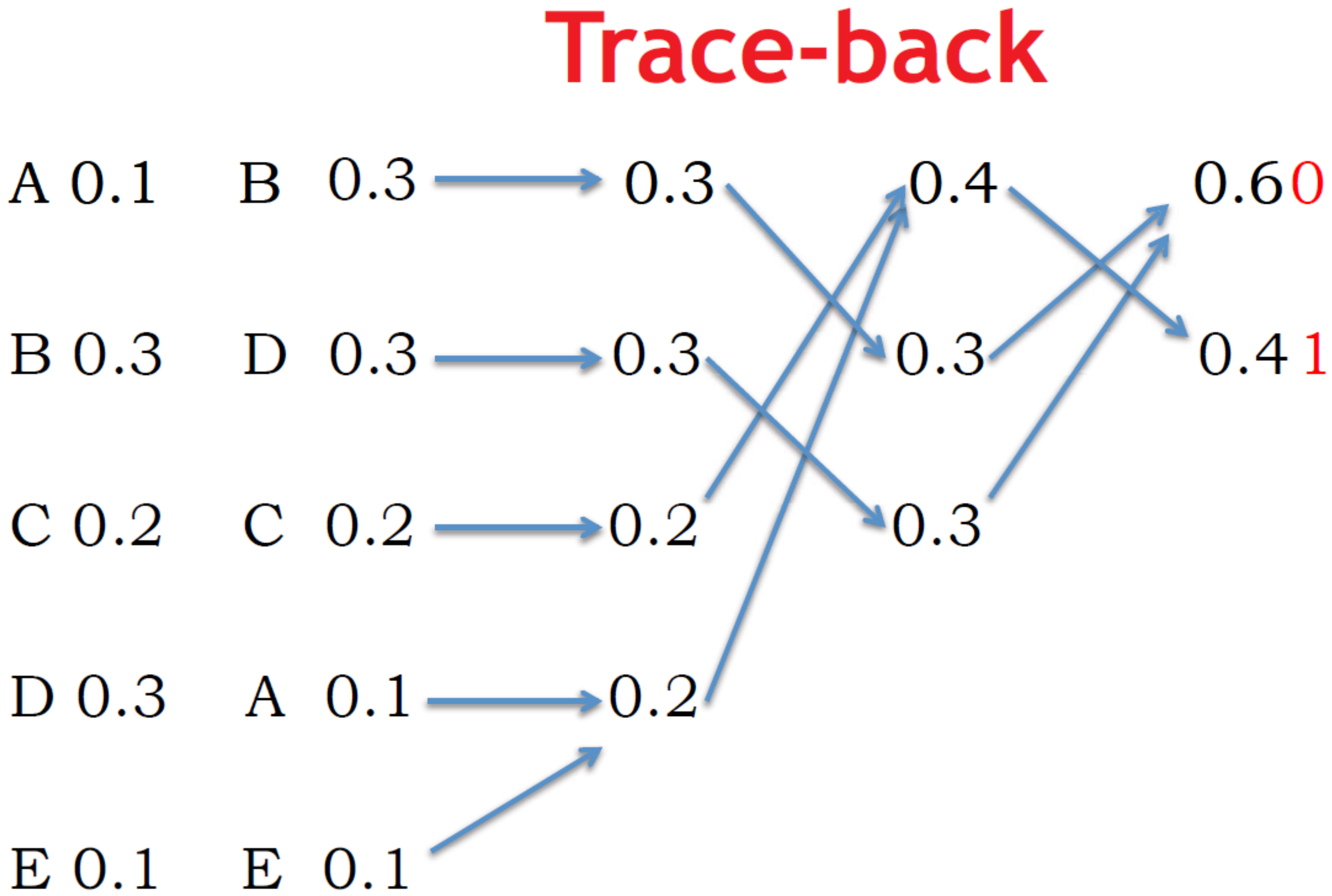
Codeword Length	Codeword	X	Probability				
2	01	1	0.25	0.3	0.45	0.55	1
2	10	2	0.25	0.25	0.3	0.45	
2	11	3	0.2	0.25	0.25		
3	000	4	0.15	0.2			
3	001	5	0.15				

Huffman Codes

Reduction

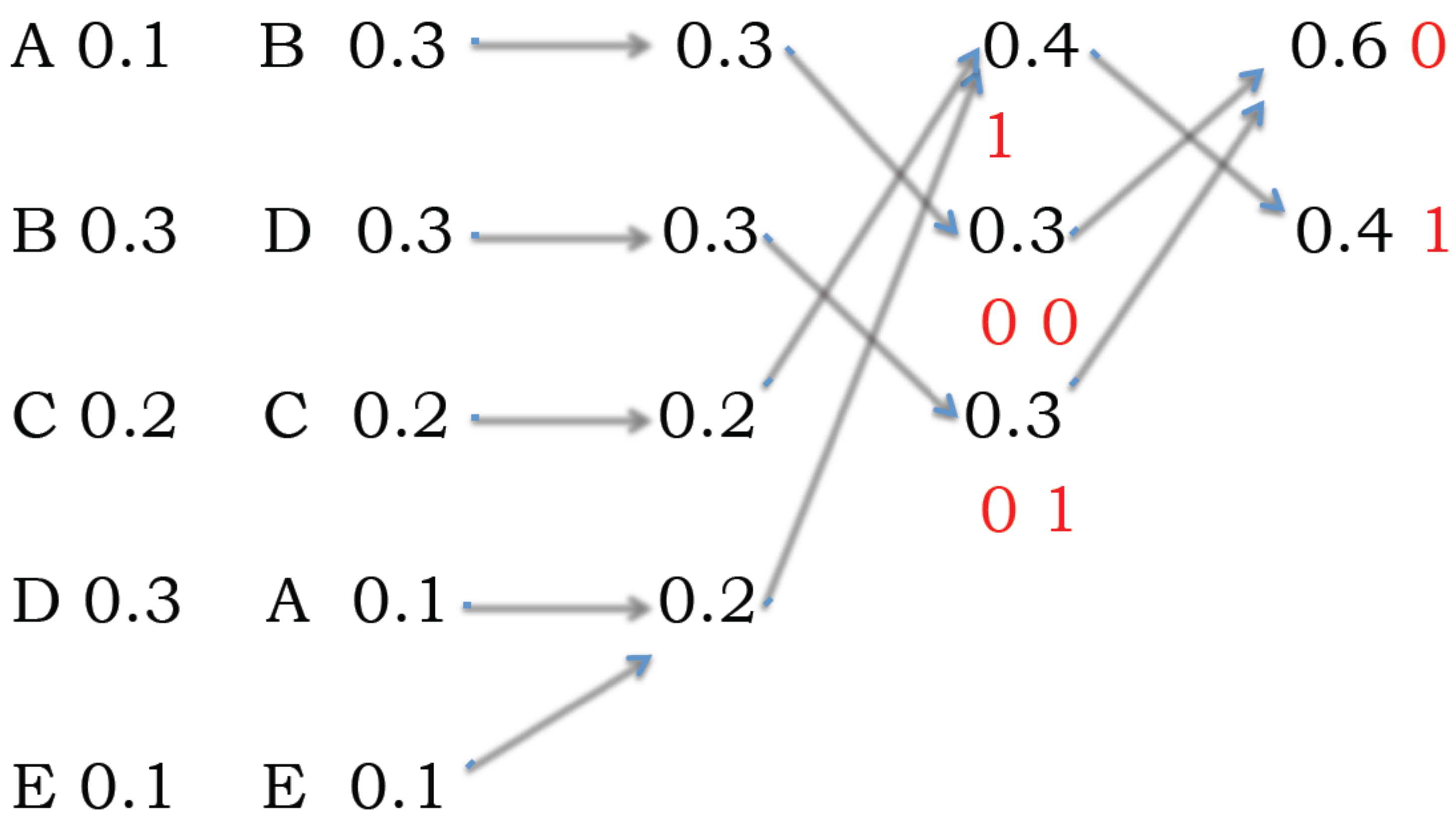


Huffman Codes

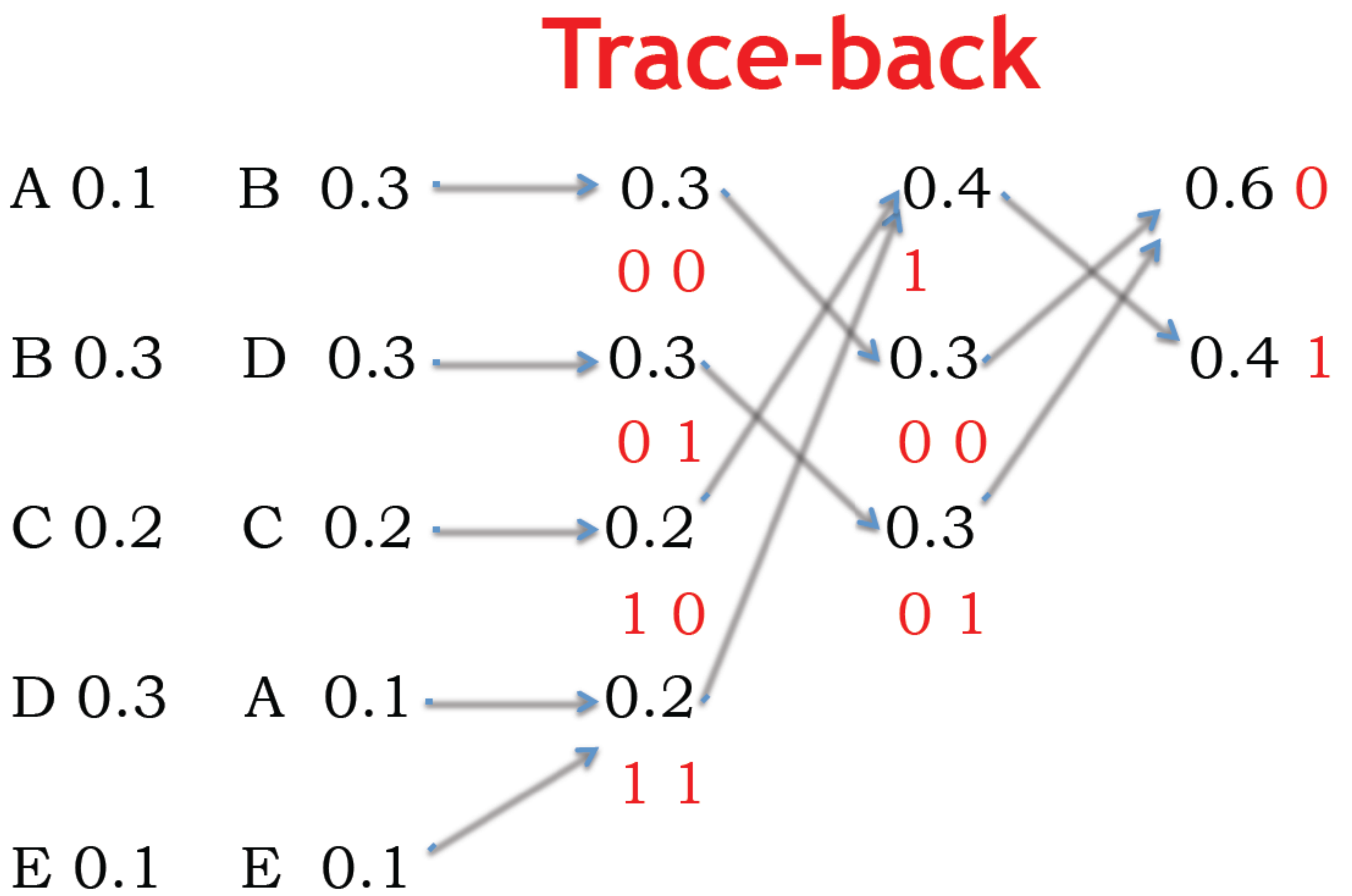


Huffman Codes

Trace-back

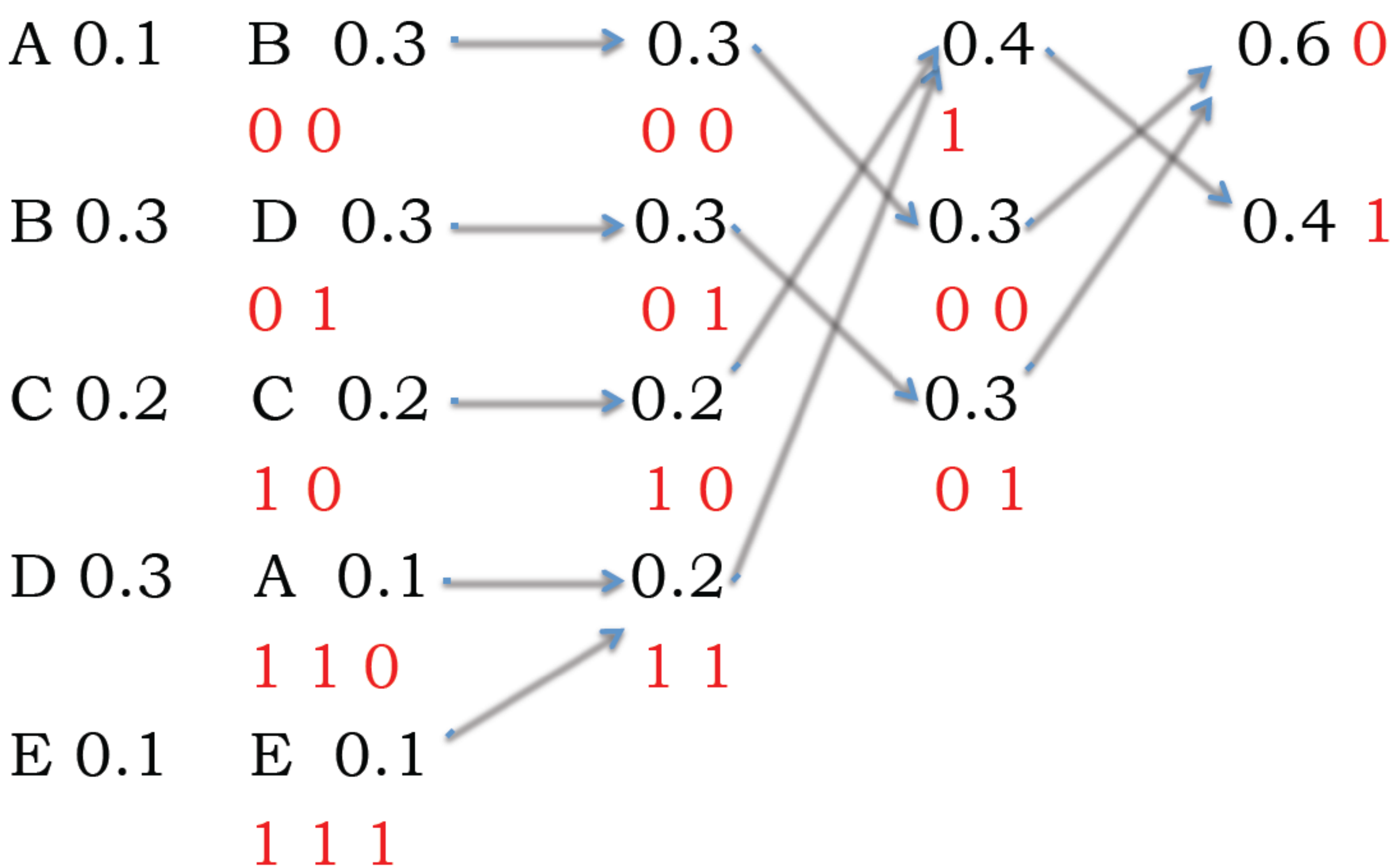


Huffman Codes



Huffman Codes

Trace-back



20 Questions

20 Questions

- Determine the value of a random variable X
- Know distribution of the random variable $p_1, \ldots, p_m$
- Want to ask minimum number of questions
- Receive "yes", "no" answer

20 Questions

index	1	2	3	4	5
p_i	.25	.25	.2	.15	.15

- Native approach

- Start with asking the most likely outcome:
  "Is $X = 1$"?
  "Is $X = 2$"?
  $\vdots$

- Expected number of binary questions $= 2.55$

20 Questions

- If we can ask any question of the form "is X $\in A$"
- Huffman code

index	1	2	3	4	5
p_i	.25	.25	.2	.15	.15
Code	01	10	11	000	001

- Q1: is X $=$ 2 or 3?
- Q2: if answer "Yes": is X $=$ 2; if answer "No": if X $=$ 1 and so on.
- $E(Q) = 2.3 = H(X)$

Huffman Codes and Shannon Codes

- Shannon code $l_i = \lceil \log 1/p_i \rceil$
- Shannon code can be much worse than Huffman code (last lecture)
- Shannon code can be shorter than Huffman code:

  (1/3, 1/3, 1/4, 1/12) result in Huffman code length (2, 2, 2, 2) or (1, 2, 3, 3); but $\lceil \log 1/p_3 \rceil = 2$
- Huffman code is shorter an average

$$\sum p_i l_{i,\text{Huffman}} \leq \sum p_i l_{i,\text{Shannon}}$$

but $l_{i,\text{Huffman}} \leq l_{i,\text{Shannon}}$ may not be true

Optimality of Huffman Codes

- Huffman code is not unique: investing the bits or exchanging two codewords of the same length

- Proof based on the following lemmas

(1) if $p_j \geq p_k$, then $l_j \leq l_k$
(2) Two longest codewords are of the same length
(3) Two longest codewords differ only in the last bit

Optimality of Huffman Codes

Proof idea

- Induction

- Consider we have found optimal codes for

$$C_m^*(p) = (p_1, \ldots, p_m)$$

$$C_{m-1}^*(p') = (p_1, \ldots, p_{m-2}, p_{m-1} + p_m)$$

Optimality of Huffman Codes

- First, $p' \to p$:

  expand the last codewords $C^*_{m-1}(p')$ for $p_{m-1} + p_m)$ by adding 0 and

  $$L(p) = L^*(p') + p_{m-1} + p_m$$

- Then, $p \to p'$:

  merging the codeswords for the two lowest-probability symbols

  $$L(p') = L^*(p) - p_{m-1} - p_m$$

- $L(p') + L(p) = L^*(p') + L^*(p)$, since $L^*(p') \le L(p')$, $L^*(p) \le L(p)$

  $$L^*(p') = L(p'), \quad L^*(p) = L(p)$$

Optimality of Huffman Codes

- Huffman code has shortest average code length in that

$$L_{\text{Huffman}} \leq L$$

  for any prefix code.

$$H(X) \leq L_{\text{Huffman}} < H(X) + 1$$

- Redundancy $=$ average Huffman codeword length - $H(X)$

SUMMARY

- Roof code: a simply construction, incurs at most 1 bit overhead per symbol
- Huffman code is a "greedy" algorithm that it combines two least likely symbols at each stage
- This local optimality ensures global optimality

Lecture 2.6: Universal Lossless Compression

Universal Source Coding

For many practical situations, the probability distribution underlying the source may be unknown

- First estimate the distribution, and then compress
- One-pass (or online) algorithm

Dictionaries for Compression

- Dates to invention of the telegraph, companies were charged by the number of letters used → codebooks for the frequently used phrases

- greetings telegrams that are popular in India

Example: “25:Merry Christmas” and “26:May Heaven’s choicest blessings be showered on the newly married couple.”

The idea of adaptive dictionary-based schemes were explored by Lempel and Ziv. Two distinct methods LZ77 and LZ78.

LZ 77

Finding the longest match within a window of past symbols and represents the string by a pointer to location of the match within and the length of the match.

- Compress $x_1, x_2, \ldots, x_n$, window size W

if find $x_j, x_{j+1}, \ldots, x_{j+k} = x_i, x_{i+1}, \ldots, x_{i+k}$ in window W, write

$$x_i, x_{i+1}, \ldots, x_{i+k} \rightarrow (1, \text{location}, \text{length})$$

else

$$x_i \rightarrow (0, x_i)$$

LZ 77 Example

String ABBABBABBBAABABA, W=4

The string is parsed:

A,B,B,ABBABB,BA,A,BA,BA

the sequence of “pointers”:

(0,A),(0,B),(1,1,1),(1,3,6),(1,4,2),(1,1,1),(1,3,2),(1,2,2)

A simple variant - asympotically optimal.

Lempel-Ziv-Welch (1977,’78,’84)

- Variant of LZ78

- Widely used, sometimes in combination with Huffman (gif, tiff, png, pdf, zip, gzip, …)

- Patents have expired --- much confusion and distress over the years around these and related patents

- Theoretical performance: Under appropriate assumptions on the source, asymptotically attains the lower bound H on compression performance

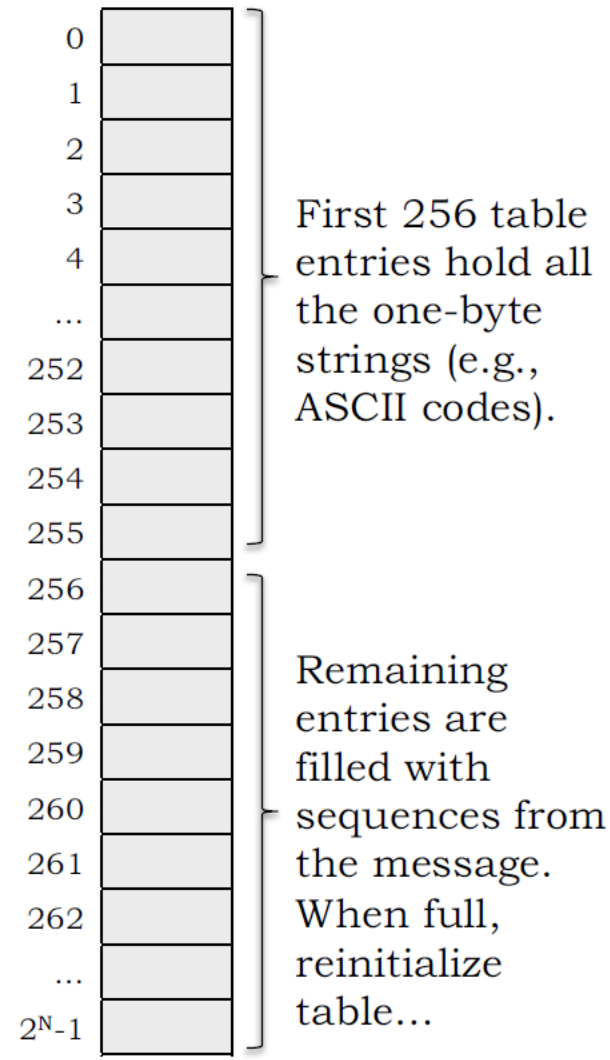
Lempel-Ziv-Welch (1977,' 78,' 84)

"Universal lossless compression of sequential (streaming) data by adaptive variable-length coding"

- Universal: doesn't need to know source statistics in advance. Learns source characteristics in the course of building a dictionary for sequential strings of symbols encountered in the source text
- Compresses streaming text to sequence of dictionary addresses --- these are the codewords sent to the receiver
- Variable length source strings assigned to fixed length dictionary addresses (codes)
- Starting from an agreed core dictionary of symbols, receiver builds up a dictionary that mirrors the sender's, with a one-step delay, and uses this to exactly recover the source text (lossless)
- Regular resetting of the dictionary when it gets too big allows adaptation to changing source characteristics

Lempel-Ziv-Welch (1977,’ 78,’ 84)

- Algorithm first developed by Ziv and Lempel (LZ88, LZ78), later improved by Welch.
- As message is processed, encoder builds a “string table” that maps symbol sequences to an N-bit fixed-length code. Table size = 2^N
- Transmit table indices, usually shorter than the corresponding string → compression!
- Note: String table can be reconstructed by the decoder using information in the encoded stream – the table, while central to the encoding and decoding process, *is never transmitted*!



Try out LZW on

abcabcabcabcabcabcabc

(You need to go some distance out on this to encounter the special case discussed later.)

LZW Encoding

S=string, c=symbol (character) of text

1. If S+c is in table, set S=S+c and read in next c.
2. When S+c isn't in table: send code for S, add S+c to table.
3. Reinitialize S with c, back to step 1.

Example Encoding

"abbbabbbab…"

256	ab
257	bb
258	bba
259	abb
260	bbab
261	
262	

1. Read a; string = a
2. Read b; ab not in table
   output 97, add ab to table, string = b
3. Read b; bb not in table
   output 98, add bb to table, string = b
4. Read b; bb in table, string = bb
5. Read a; bba not in table
   output 257, add bba to table, string = a
6. Read b, ab in table, string = ab
7. Read b, abb not in table
   output 256, add abb to table, string = b
8. Read b, bb in table, string = bb
9. Read a, bba in table, string = bba
10. Read b, bbab not in table
    output 258, add bbab to table, string = b

Encoder Notes

- The encoder algorithm is greedy – it’s designed to find the longest possible match in the string table before it makes a transmission.

- The string table is filled with sequences actually found in the message stream. No encodings are wasted on sequences not actually found in the input data.

- Note that in this example the amount of compression increases as the encoding progresses, i.e., more input bytes are consumed between transmissions.

- Eventually the table will fill and then be reinitialized, recycling the N-bit codes for new sequences. So the encoder will eventually adapt to changes in the probabilities of the symbols or symbol sequences.

LZW Decoding

```
Read CODE
STRING = TABLE[CODE] // translation table

WHILE there are still codes to receive DO
    Read CODE from encoder
    IF CODE is not in the translation table THEN
        ENTRY = STRING + STRING[0]
    ELSE
        ENTRY = get translation of CODE
    END
    output ENTRY
    add STRING+ENTRY[0] to the translation table
    STRING = ENTRY
END
```

(Ignoring special case in IF):

1. Translate received code to output the corresponding table entry E=e+R (e is first symbol of entry, R is rest)
2. Enter S+e in table.
3. Reinitialize S with E, back to step 1.

Concluding

- LZW is a good example of compression or communication schemes that "transmit the model" (with auxiliary information to run the model), rather than "transmit the data"

- There's a whole world of lossy compression!

SUMMARY

- LZW Code as an example of universal compression
- Lossy compression vs. Lossless compression