What’s Wrong with Huffman?

The standard Huffman compression (also known as "static Huffman") has two major drawbacks:

1. the Huffman tree (or at least the symbol probabilities) must be transmitted along with the encoded data (or stored in the compressed file)
2. encoding requires two passes through the data:
    - one to compute the probabilities of the input symbols
    - one to encode the data

If we fixed the probabilities for all time, we could "solve" these problems:

- for example, if we knew we were always compressing English text, we could use some precomputed probabilities of English letters:

E	.124	N	.070	R	.061	M	.025	G	.020	V	.008		
T	.089	I	.067	D	.046	W	.023	Y	.020	K	.007	J	.001
A	.080	H	.065	L	.036	C	.022	P	.016	Q	.001	Z	.000
O	.076	S	.062	U	.027	F	.022	B	.013	X	.001		

Q: What’s wrong with this solution?

A:

Adaptive Huffman

Adaptive Huffman is a variation on static Huffman that:

1. does not require the probabilities to be transmitted with the encoded data
2. requires only one pass through the data for encoding
3. does not use fixed symbol probabilities

Cool! But how does it do that?

There are different versions of adaptive Huffman. The algorithms can be quite complicated, but the ideas are simple.

- start with a tree with only one (leaf) node: the "0 node"
- for each symbol in the input:
  - if the symbol does not appear in the tree
    - transmit the code for the 0 node
    - transmit the symbol
    - split the 0 node into two children: one becomes the new 0 node, the other is the code for the new symbol
  - if the symbol does appear in the tree
    - transmit the code for the input symbol
  - rebuild the tree with updated probabilities

Adaptive Huffman Encoding Example

Input:

a a b b b b c

Transmit (encoding):

Decoding Adaptive Huffman

Adaptive Huffman does not have to transmit the probabilities (or the Huffman tree) along with the data because as codes are transmitted, the decoder can update the probabilities and recompute the tree just like the encoder.

- start with a tree with only one (leaf) node: the "0 node"
- for each code received:
    - if the code is the code for the 0 node
        - split the 0 node into two children: one becomes the new 0 node, the other is the next symbol (S) received
        - output S
    - if the code is for a node (N) other than the 0 node
        - output the symbol at node N
    - rebuild the tree with updated probabilities

Adaptive Huffman Decoding Example

Input:

```
a  1  0 b  0 1  0 1  1  0 0 c
```

Output (decoding):

Another LZW Decoding Walkthrough

```
<65><256><257><82><71><72><33>
```

c	*s*
0	
1	
2	
. . .	. . .
33	!
. . .	. . .
65	A
66	B
. . .	. . .
71	G
. . .	. . .
255	

c	*s*
256	
257	
258	
259	
260	
261	
262	
263	
264	
265	
266	
267	
268	
269	
. . .	. . .

w	*k*	*output*

Your Favourite Compression Utilities

Utility	***Format***	***Compression***
pkarc (DOS) arc (Unix, Mac, etc.)	.arc, .ark	LZW
arj (DOS)	.arj	LZ77 + hashing, secondary static Huffman
Compuserve GIF	.gif	LZW
gzip	.gz	LZ77 + hashing, secondary static Huffman
lha, lharc	.lha, .lhz	LZ77 + tries, secondary static Huffman
squeeze (DOS)	.sqz	LZ77 + hashing
pkzip (DOS) zip (Unix) WinZip (Windows)	.zip	LZ77 + hashing, secondary static Huffman
zoo (DOS/Mac/Unix)	.zoo	LHA
freeze (Unix)	.F	LZ77 + hashing, secondary adaptive Huffman
yabba (Unix)	.Y	LZ78 variant
compress (Unix)	.Z	LZW

MPEG

- "Moving Picture Experts Group"
- compression for movies + audio
  - low resolution video
  - high quality audio

1. do irreversible compression on colour channels (not on shade channel)
2. for each block of 16x16 in a frame, try to find a "similar" block in a previous (*or future*) frame
3. store the differences between blocks instead of storing entire blocks
4. Huffman encode the whole thing

- the result: a sequence of frames
  - some frames (I frames) are original frames encoded as still images
  - some frames (P frames) are predicted from the most recent I or P frame
  - some frames (B frames) are predicted from the two closest I or P frames (in the past and future)

JPEG

- "Joint Photographic Experts Group"
- lossy compression
- full colour or gray scale images (not black and white)
- for pictures to be viewed by humans
  - humans are very good at judging light vs. dark
  - not as good at judging minor variations in colour
- ~ 100:1 compression to 3:1 (parameterized for quality vs. size)

1. do irreversible compression on colour channels
2. compute the *Discrete Cosine Transform* for each 8x8 block
3. "reduce" the DCT output: more reduction for less visible elements
4. Huffman encode the reduced output

- Lossless JPEG (~ 2:1 compression)
  1. predict the value of a pixel based on the pixels above it and to the left (for example, average them)
  2. record the difference between predicted value and actual value
  3. Huffman encode the differences