Huffman and Arithmetic Codes: Optimal Data Compression Methods
Compression Example

100k file, 6 letter alphabet:

File Size:
ASCII, 8 bits/char: 800kbits
$2^3 > 6$; 3 bits/char: 300kbits

Why?
Storage, transmission vs 5 Ghz cpu

<table>
<thead>
<tr>
<th>Letter</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>45%</td>
</tr>
<tr>
<td>b</td>
<td>13%</td>
</tr>
<tr>
<td>c</td>
<td>12%</td>
</tr>
<tr>
<td>d</td>
<td>16%</td>
</tr>
<tr>
<td>e</td>
<td>9%</td>
</tr>
<tr>
<td>f</td>
<td>5%</td>
</tr>
</tbody>
</table>
Compression Example

100k file, 6 letter alphabet:

File Size:

- ASCII, 8 bits/char: 800kbits
- $2^3 > 6$; 3 bits/char: 300kbits
- better: 2.52 bits/char $74\% \times 2 + 26\% \times 4$: 252kbits

Optimal?

<table>
<thead>
<tr>
<th>Letter</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>45%</td>
</tr>
<tr>
<td>b</td>
<td>13%</td>
</tr>
<tr>
<td>c</td>
<td>12%</td>
</tr>
<tr>
<td>d</td>
<td>16%</td>
</tr>
<tr>
<td>e</td>
<td>9%</td>
</tr>
<tr>
<td>f</td>
<td>5%</td>
</tr>
</tbody>
</table>

E.g.:

<table>
<thead>
<tr>
<th>Letter</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>00</td>
</tr>
<tr>
<td>b</td>
<td>01</td>
</tr>
<tr>
<td>c</td>
<td>100</td>
</tr>
<tr>
<td>d</td>
<td>10</td>
</tr>
<tr>
<td>e</td>
<td>101</td>
</tr>
<tr>
<td>f</td>
<td>110</td>
</tr>
</tbody>
</table>

Why not:

<table>
<thead>
<tr>
<th>Letter</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>00</td>
</tr>
<tr>
<td>b</td>
<td>01</td>
</tr>
<tr>
<td>c</td>
<td>100</td>
</tr>
<tr>
<td>d</td>
<td>10</td>
</tr>
<tr>
<td>e</td>
<td>101</td>
</tr>
<tr>
<td>f</td>
<td>110</td>
</tr>
</tbody>
</table>

1101110 = cf or ec?
Data Compression

Binary character code ("code")
  each k-bit source string maps to unique code word
  (e.g. k=8)
  “compression” alg: concatenate code words for
  successive k-bit “characters” of source

Fixed/variable length codes
  all code words equal length?

Prefix codes
  no code word is prefix of another (unique decoding)
Prefix Codes = Trees

```
1 0 1 0 0 0 0 0 1
f a b
```

```
1 1 0 0 0 1 0 1
f a b
```
Put most frequent under root, then recurse …
Greedy Idea #1

Top down: Put *most* frequent under root, then recurse

Too greedy: unbalanced tree

\[0.45*1 + 0.16*2 + 0.13*3 \ldots = 2.34\]
not too bad, but imagine if all freqs were \(\sim 1/6\):
\[(1+2+3+4+5+5)/6=3.33\]
Greedy Idea #2

Top down: Divide letters into 2 groups, with ~50% weight in each; recurse (Shannon-Fano code)

Again, not terrible
2*0.5 + 3*0.5 = 2.5

But this tree can easily be improved! (How?)
Greedy idea #3

Bottom up: Group least frequent letters near bottom

<table>
<thead>
<tr>
<th>Letter</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>45%</td>
</tr>
<tr>
<td>b</td>
<td>13%</td>
</tr>
<tr>
<td>c</td>
<td>12%</td>
</tr>
<tr>
<td>d</td>
<td>16%</td>
</tr>
<tr>
<td>e</td>
<td>9%</td>
</tr>
<tr>
<td>f</td>
<td>5%</td>
</tr>
</tbody>
</table>

Diagram:}

```
100
  /
 /   
25    14
  |    |
c:12  b:13
  |    |    |    |    |    |    |
f:5 e:9
```
\[.45 \times 1 + .41 \times 3 + .14 \times 4 = 2.24 \text{ bits per char}\]
Huffman’s Algorithm (1952)

Algorithm:

insert node for each letter into priority queue by freq
while queue length > 1 do
    remove smallest 2; call them x, y
    make new node z from them, with f(z) = f(x) + f(y)
    insert z into queue

Analysis: \( O(n) \) heap ops: \( O(n \log n) \)

Goal: Minimize \( B(T) = \sum_{c \in C} \text{freq}(c) \times \text{depth}(c) \)

Correctness: ???
Correctness Strategy

Optimal solution may not be unique, so cannot prove that greedy gives the only possible answer.

Instead, show that greedy’s solution is as good as any.

How: an exchange argument
Defn: A pair of leaves $x,y$ is an inversion if
\[ \text{depth}(x) \geq \text{depth}(y) \]
and
\[ \text{freq}(x) \geq \text{freq}(y) \]

Claim: If we flip an inversion, cost never increases.

Why? All other things being equal, better to give more frequent letter the shorter code.

\[
\begin{align*}
\text{before} & \quad (d(x) \cdot f(x) + d(y) \cdot f(y)) - (d(x) \cdot f(y) + d(y) \cdot f(x)) = \\
\text{after} & \quad (d(x) - d(y)) \cdot (f(x) - f(y)) \geq 0 
\end{align*}
\]

I.e., non-negative cost savings.
Lemma 1: “Greedy Choice Property”

The 2 least frequent letters might as well be siblings at deepest level

Let a be least freq, b 2\textsuperscript{nd}

Let u, v be siblings at max depth, f(u) ≤ f(v) (why must they exist?)

Then (a,u) and (b,v) are inversions. Swap them.
Lemma 2

Let (C, f) be a problem instance: C an n-letter alphabet with letter frequencies f(c) for c in C.

For any x, y in C, z not in C, let C' be the (n-1) letter alphabet C - {x,y} ∪ {z} and for all c in C' define

\[ f'(c) = \begin{cases} 
  f(c), & \text{if } c \neq x,y,z \\
  f(x) + f(y), & \text{if } c = z
\end{cases} \]

Let T' be an optimal tree for (C', f').

Then

\[ T' = \begin{array}{c}
  \text{T} \\
  \text{x} \quad \text{y}
\end{array} \]

is optimal for (C, f) among all trees having x, y as siblings.
Proof:

\[ B(T) = \sum_{c \in C} d_T(c) \cdot f(c) \]

\[ B(T) - B(T') = d_T(x) \cdot (f(x) + f(y)) - d_{T'}(z) \cdot f'(z) \]

\[ = (d_{T'}(z) + 1) \cdot f'(z) - d_{T'}(z) \cdot f'(z) \]

\[ = f'(z) \]

Suppose \( \hat{T} \) (having \( x \) & \( y \) as siblings) is better than \( T \), i.e.

\[ B(\hat{T}) < B(T) \].

Collapse \( x \) & \( y \) to \( z \), forming \( \hat{T}' \); as above:

\[ B(\hat{T}) - B(\hat{T}') = f'(z) \]

Then:

\[ B(\hat{T}') = B(\hat{T}) - f'(z) < B(T) - f'(z) = B(T') \]

Contradicting optimality of \( T' \)
Theorem: Huffman gives optimal codes

Proof: induction on $|C|$

Basis: $n=1,2$ – immediate

Induction: $n>2$

Let $x, y$ be least frequent
Form $C', f', z$, as above
By induction, $T'$ is opt for $(C', f')$

By lemma 2, $T' \rightarrow T$ is opt for $(C, f)$ among trees with $x, y$ as siblings

By lemma 1, some opt tree has $x, y$ as siblings
Therefore, $T$ is optimal.
Data Compression

Huffman is **optimal**.

**BUT** still might do better!

- Huffman encodes fixed length blocks. What if we vary them?
- Huffman uses one encoding throughout a file. What if characteristics change?
- What if data has structure? E.g. raster images, video,…
- Huffman is lossless. Necessary?

**LZW, MPEG, …**
David A. Huffman, 1925-1999
Arithmetic Coding

In some ways a generalization of Huffman coding
Can provide better compression (by relaxing some of the Huffman assumptions) approaching theoretical limit Algorithmically very different
Arithmetic Code

Shannon Bound letter i, Prob Pi

I ndp

need $- \Sigma P_i \times \log_2 P_i$ (bits per character)
Example

\( \{a, b, c\} \), \( P = \frac{1}{3} \)

Huffman

\[
\frac{1}{3} \cdot 1 + \frac{2}{3} \cdot 2 = 1.67
\]

b. Hints/Chew
\[ \text{Shannon} \]

\[-\sum_2^{\frac{1}{3}} \log_2 \frac{1}{3} = \log_2 3 \]

\[= 1.585 < 1.666 \]
An Idea: message: abach...
view as (01021...) (base 3)
In more detail

may \(0.01021\)

find interval for

\(0.01021x\) for all \(x\)

Send some

\(x \in [0.01021, 0.01021222222\ldots]\)

(any such \(x\) will do; might as well be the shortest one in binary)
What about \( \pm \) frequencies?

\[ \text{Ex: } P_a = \frac{1}{2}, P_b = \frac{1}{3}, P_c = \frac{1}{6} \]

Same idea, but unequal intervals. E.g. "a" maps to \( 1^{\text{st}} \) half; "ac" to last sixth of \( 1^{\text{st}} \) half.
abc → o + \frac{5}{6} \cdot \frac{1}{2} + [\frac{1}{2}]_5 \cdot \frac{1}{6} \cdot \frac{1}{6}
In general, if $i^{\text{th}}$ letter of the alphabet $a_i$ has frequency $p_i$, and $q_i = \sum_{j<i} p_i$

Associate an interval $(b, l) = \{ x \mid b \leq x < b+l \}$ with a string as follows:

empty string $\Rightarrow$ interval $(0, 1)$
if string $s$ $\Rightarrow$ interval $(b, l)$ then
string $sa_i$ $\Rightarrow$ interval $(b+q_i, l*p_i)$
How many bits?

may .01021
find interval fn
.01021x for all x

Send some

\( \forall x \in [0.01021, 0.0102122222...]) 

(any such \( v \) will do; might as well be the shortest one in binary)
Fact: interval \( \frac{1}{4} \leq 3 \leq \frac{1}{2} \) contains \( \frac{3}{4} \) for exactly one integer \( k \).
More generally, need \( \Gamma - \log_2 \leq 7 \) to encode a point in an interval of width \( \Sigma \).
Arithmetic Coding

$H_i = \sum p_i \log_2 \frac{1}{p_i}$

$\text{msg length} = n \times \text{expect}$

$n p_i$ for letter $i$
So: interval length
≈ \pi \sum p_i n\pi \approx \frac{7 \pi}{22}.

= Shannon
More:

patients

non-independent

adaptive

(But must be careful about arithmetic; # bits)