

More Efficient Algorithms and Analyses for Unequal Letter Cost Prefix-Free Coding

Mordecai Golin¹ Jian Li²

¹Hong Kong University of Technology

²University of Maryland, College Park

September 21, 2007

Definition

- ▶ Alphabet: $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_t\}$.
- ▶ Words: $w = a_1 a_2 a_3 \dots a_k$ where $a_i \in \Sigma$.
- ▶ Σ^* : all words written using Σ .
- ▶ Code: $C = \{w_1, w_2, \dots, w_n\}$.
- ▶ Prefix: word $w \in \Sigma^*$ is prefix of word $w' \in \Sigma^*$ if $w' = wu$ for non-empty $u \in \Sigma^*$.
- ▶ Prefix-free code $C: w_i$ is not a prefix of w_j for all $i \neq j$.

Definition

- ▶ Cost of a letter σ_i : c_i
- ▶ Cost of a word $w = \sigma_{i_1}\sigma_{i_2}\dots\sigma_{i_l}$: $cost(w) = \sum_{k=1}^l c_{i_k}$.
- ▶ Cost of a Code C : $cost(C) = \sum_{i=1}^n cost(w_i)p_i$ where $p_i, 1 \leq i \leq n$ is the probabilities associated with word w_i .

Definition

Examples

$\Sigma = \{a, b\}$. , $cost(a) = 1$ and $cost(b) = 3$.

x	aaa	aab	ab	b
$cost(x)$	3	5	4	3

x	aaa	aab	ab	$aaba$
$cost(x)$	3	5	4	6

Figure: Code $\{aaa, aab, ab, b\}$ is prefix-free. Code $\{aaa, aab, ab, aaba\}$ is not prefix-free because aab is a prefix of $aaba$.

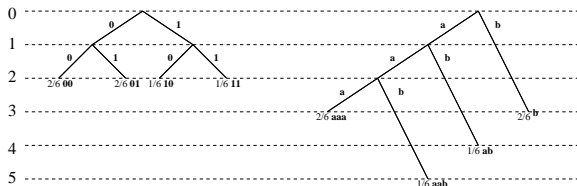


Figure: Tree representation of prefix-free code

History

- ▶ finite alphabet, equal letter cost:
 1. **Alphabetic code**(code words must be chosen in increasing alphabetic order): Dynamic Programming $O(tn^3)$ [I.Itai '76].
 2. **General code**: Huffman Code, $O(tn \log n)$ -time greedy algorithm [Huffman '52].
- ▶ finite alphabet, unequal letter cost:
 1. **Alphabetic code**: $O(tn^3)$
 2. **General code**:

History

► finite alphabet, unequal letter cost:

1. **General code:**

- 1.1 Not Known to be NP-hard
- 1.2 Exact algorithm by transforming the problem into an integer program. [Karp '61]
- 1.3 Dynamic Programming, $O(n^{c_t+O(1)})$ [Golin, Rote '98; Bradford, Golin, Larmore, Rytter '02]
- 1.4 Polynomial Time Approximation Scheme (PTAS) [Golin, Kenyon, Young '02]
- 1.5 Information entropy lower bound:

$$OPT \geq \frac{1}{c} H(p_1, p_2, \dots, p_n)$$

where $H(p_1, p_2, \dots, p_n) = -\sum_{i=1}^n p_i \log p_i$ and c is the root of $1 = \sum_{i=1}^t 2^{-cc_i}$ [Krause '62]

- 1.6 Many efficient algorithm with additive error
 $C(T) \leq \frac{1}{c} H(p_1, \dots, p_n) + f(C)$: [Krause '62, Csiszar '69, Cott '77, Mehlhorn '80, Altenkamp & Mehlhorn '80, ...]
for eg: in [Mehlhorn '80]

$$cC(T) - H(p_1, \dots, p_n) \leq (1 - p_1 - p_n) + cc_t$$

Motivation

Motivation for Unequal Letter Cost

Models the scenario where different character corresponds to different transmission times or storage spaces: for eg

- ▶ telegraph channel: $\Sigma = \{., -\}$ where $c(.) = 1$ and $c(-) = 2$;
- ▶ (a, b) -run-length-code in magnetic storage: $\Sigma = \{0, 1\}$, each **1** must be preceded by at least a , and at most b , **0**'s So, this can be modeled by using alphabet $\{0^k 1 : k = a, a + 1, \dots, b\}$ with associated cost $\{c_i = a + i - 1\}$
- ▶

Motivation

Motivation for **infinite alphabet**

Example

$$\mathcal{C} = \{1, 2, 3, \dots\}:$$

Construct a tree where the children of a node are stored in a linked list.

Taking m^{th} child corresponds to using σ_m whose cost is $c_m = m$

Motivation

Example

1-ended codes: $\mathcal{L} = \{w \in \{0,1\}^* \mid \text{the last letter in } w \text{ is a } 1\}$ Find a minimum prefix-free code in \mathcal{L} .

Note that $\mathcal{L} = \mathcal{Q}^*$ where $\mathcal{Q} = \{1, 01, 001, 0001, \dots\}$

Create an **new infinite alphabet** $\Sigma_{\mathcal{Q}} = \{\sigma_q \mid q \in \mathcal{Q}\}$ where the cost of σ_q is $\text{len}(q)$. Thus, $\mathcal{C} = \{1, 2, 3, \dots\}$

Motivation

Example

Σ' -ended unequal-letter-cost codes

$$\mathcal{L} = \Sigma^* \Sigma' = \{w \in \Sigma^* \mid \text{the last letter in } w \text{ is in } \Sigma'\}$$

We can see $\mathcal{L} = Q^*$ where $Q = (\Sigma - \Sigma')^* \Sigma'$ is prefix-free
Therefore, we create a **new infinite alphabet** $\Sigma_Q = \{\sigma_q \mid q \in Q\}$.

The **important observation** is

$$d_j = |\{w \in \Sigma_Q \mid \text{cost}(w) = j\}|$$

satisfies a linear recurrence relation. For Example.....

Motivation

Example

Σ' -ended unequal-letter-cost codes

For eg: $\Sigma = \{1, 2, 3\}$ with $\mathcal{C} = \{1, 1, 2\}$ and $\Sigma' = \{1\}$. Find minimum cost prefix-free code in $\mathcal{L} = \{1, 2, 3\}^* \{1\}$.

Now $\mathcal{L} = \mathcal{Q}^*$ where $\mathcal{Q} = \{2, 3\}^* \{1\}$. The d_i is

$$d_1 = 1, d_2 = 1, d_3 = 2, d_4 = 3, d_5 = 5, \dots, d_{i+2} = d_{i+1} + d_i$$

the Fibonacci number.

Motivation

Example

"**balanced**" words \mathcal{L} . i.e., all words which contain exactly as many **0**'s as **1**'s.

$\mathcal{L} = \mathcal{D}^*$ where

$\mathcal{D} = \{w \mid w \text{ is balanced but no prefix of } w \text{ is balanced}\}$.

Let $d_j = |\{w \in \mathcal{D} \mid \text{cost}(w) = j\}|$.

From standard generating function, we can show

- ▶ $d_j = 0$ for $j = 0$ and odd j
- ▶ $d_j = 2C_{j/2-1}$ for even $j > 0$, where $C_i = \frac{1}{i+1} \binom{2i}{i}$ is the i^{th} Catalan number.

The Generic Algorithm

A generalization of Shannon's original binary splitting algorithm
[Shannon '48]

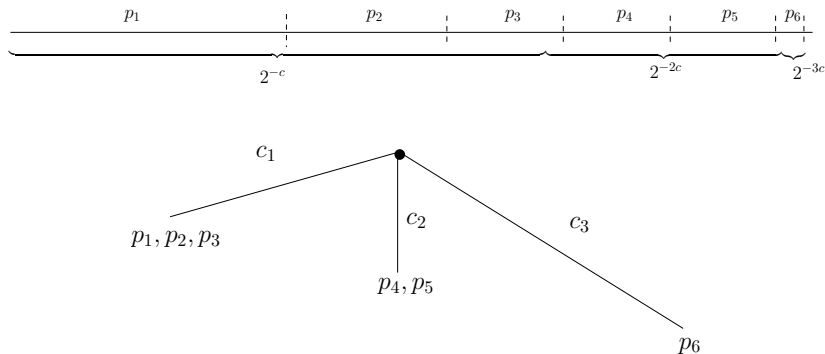


Figure: The first splitting step for a case when $n = 6$ $c_1 = 1$, $c_2 = 2$, $c_3 = 3$ and the associated preliminary tree.

The Generic Algorithm

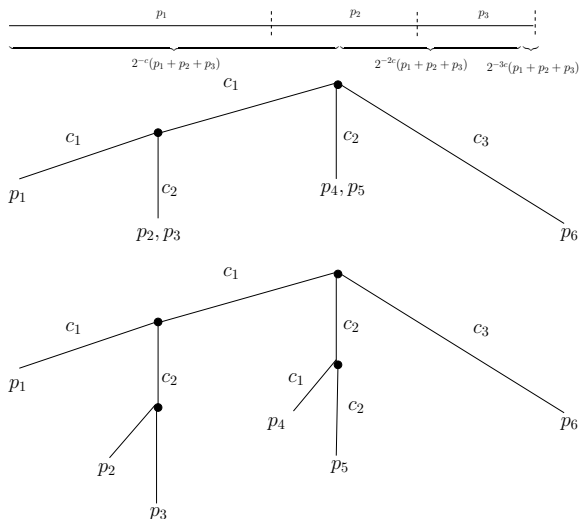


Figure: After two more splits, the final coding tree is constructed. The associated code is $\{\sigma_1\sigma_1, \sigma_1\sigma_2\sigma_1, \sigma_1\sigma_2\sigma_2, \sigma_2\sigma_1, \sigma_2\sigma_2, \sigma_3\}$

Our Algorithm

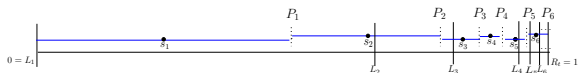


Figure: The splitting procedure.



Figure: The shift procedure.

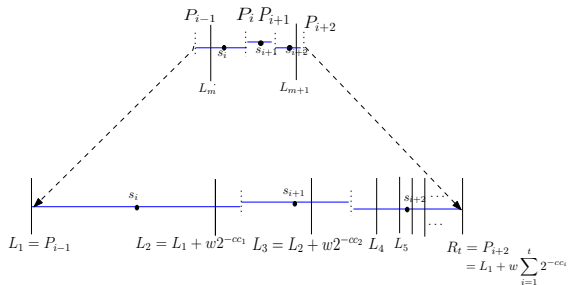


Figure: The recursive step.

Our Algorithm

```
CODE( $l, r, U$ );  
{ Constructs codewords  $U_l, U_{l+1}, \dots, U_r$  for  $p_l, p_{l+1}, \dots, p_r$ .  
   $U$  is previously constructed common prefix of  $U_l, U_{l+1}, \dots, U_r$ .}  
if  $l = r$   
  then codeword  $U_l$  is set to be  $U$ .  
else { Distribute  $p_i$ s into initial bins  $I_m^*$  }  
   $L = P_{l-1}; R = P_r; w = R - L$   
   $\forall m$ , let  $L_m = L + w \sum_{i=1}^{m-1} 2^{-cc_i}$  and  $R_m = L_m + w 2^{-cc_m}$ .  
  set  $I_m^* = \{k \mid L_m \leq s_k < R_m\}$  }  
   $M = 0; k = l;$   
  while  $k \leq r$  do  
     $M = M + 1;$   
     $l_M = k; r_M = \max(\{k\} \cup \{i > k \mid i \in I_M^*\});$   
     $k = r_M + 1;$   
  if  $r_1 = r$  then  $M = 2; r_1 = r - 1; l_2 = r_2 = r;$   
  for  $m = 1$  to  $M$  do  
    CODE( $l_m, r_m, U_{\sigma_m}$ );
```

Figure: Our algorithm.

Our Results

Definition

- ▶ **Redundancy** of Code T :

$$R(T; p_1, \dots, p_n) = C(T) - \frac{1}{c} H(p_1, \dots, p_n).$$

- ▶ **Normalized Redundancy**:

$$\text{NR}(T; p_1, \dots, p_n) = cR = cC(T) - H(p_1, \dots, p_n).$$

Our Results

We start by noting that, when $t \leq \infty$ our bound is never worse than 1 plus the Mehlhorn's bound of $(1 - p_1 - p_n) + cc_t$.

Theorem

If $t < \infty$ then $\text{NR} \leq 2(1 - p_1) + cc_t$.

Our Results

Definition

Set $\beta_m = 2^{cc_m} \sum_{i=m}^t 2^{-cc_i}$ and $\beta = \sup\{\beta_m \mid 1 \leq m \leq t\}$

Our Results

Our first improved bound:

Theorem

If $\beta < \infty$ then

$$\text{NR} \leq 2(1 - p_1) + \max(c(c_2 - c_1), 1 + \log \beta)$$

.

An improved bound for many finite cases:

if $t < \infty$, then $\beta_m = 2^{cc_m} \sum_{i=m}^t 2^{-cc_i} \leq t - m + 1$ so $\beta \leq t$.

Theorem

If t is finite then

$$\text{NR} \leq 2(1 - p_1) + \max(c(c_2 - c_1), 1 + \log t)$$

.

Our Results

Theorem

If $d_j = O(1)$, then $NR = O(1)$. In particular, if $\forall j, d_j \leq K$ then $\beta \leq \frac{2^c K}{1-2^{-c}}$ so,

$$NR \leq 2(1 - p_1) + \max \left(c(c_2 - c_1), 1 + c + \log \left(\frac{K}{1 - 2^{-c}} \right) \right).$$

Our Results

Theorem

If C is infinite and $\sum_{m=1}^{\infty} c_m 2^{-ccm} < \infty$, then, for every $\epsilon > 0$

$$R \leq \epsilon \frac{1}{C} H(p_1, \dots, p_n) + f(C, \epsilon)$$

where $f(C, \epsilon)$ is some constant based only on C and ϵ . Note that this is equivalent to stating that

$$C(T) \leq (1 + \epsilon)OPT + f(C, \epsilon)$$

Examples

Example

Let $t = 3$ with $c_1 = c_2 = 1$ and $c_3 = \alpha \geq 1$.

The old bounds: $\rightarrow \infty$ as $\alpha \rightarrow \infty$.

Our Bound is

$$NR_\alpha \leq 2(1 - p_1) + \max(c^{(\alpha)}(c_2 - c_1), 1 + \log t) \leq 3 + \log 3$$

and

$$R_\alpha = \frac{NR_\alpha}{c^{(\alpha)}} \leq 3 + \log 3$$

independent of α .

Examples

Example

Let $\mathcal{C} = (1, 2, 3, \dots)$. i.e., $c_m = m$.

The old bounds: $\rightarrow \infty$.

Our bounds:

$$\text{NR}_t \leq 2(1 - p_1) + \max(c(c_2 - c_1), 1 + \log t) \leq 4.388$$

and

$$R_t \leq 6.232.$$

Examples

Example

Let \mathcal{C} contain d copies each of $i = 1, 2, 3, \dots$, i.e., $c_m = 1 + \lfloor \frac{m-1}{d} \rfloor$. Note that $K = d$.

- ▶ If $d = 1$, i.e., $c_m = m$, then $c = K = 1$ and

$$R = \text{NR} \leq 2(1 - p_1) + 2.$$

- ▶ If $d > 1$ then $A(z) = \sum_{m=1}^{\infty} c_m z^m = \frac{dz}{1-z}$. The solution α to $A(\alpha) = 1$ is $\alpha = \frac{1}{d+1}$, so $c = -\log \alpha = \log(d+1)$.

$$\text{NR} \leq 2(1 - p_1) + \left(1 + \log \left(\frac{K}{1 - 2^{-c}} \right) \right) \leq 3 + \log(d+1)$$

and

$$R \leq 1 + \frac{3}{\log(d+1)}.$$

Examples

d_j are integral and satisfy a linear recurrence relation (Recall Σ' -ended Codes).

The generating function $A(z) = \sum_{j=1}^{\infty} d_j z^j = \sum_{m=1}^{\infty} z^{c_m}$ can be written as $A(z) = \frac{P(z)}{Q(z)}$ where $P(z)$ and $Q(z)$ are relatively prime polynomials. Let γ be a smallest modulus root of $Q(z)$. Let $c = -\log \alpha$.

Our algorithm creates a code T satisfying

$$C(T) - OPT \leq \epsilon OPT + f(C, \epsilon) = \epsilon OPT + \log_{\gamma/\alpha} 1/\epsilon + O(\log \log 1/\epsilon).$$

Examples

An interesting open question:

"**balanced**" words \mathcal{L} . i.e., all words which contain exactly as many **0**'s as **1**'s. $\mathcal{L} = \mathcal{D}^*$ where

$\mathcal{D} = \{w \mid \text{no prefix of } w \text{ is balanced}\}$.

Let $d_j = |\{w \in \mathcal{D} \mid \text{cost}(w) = j\}|$.

- ▶ $d_j = 0$ for $j = 0$ and odd j
- ▶ $d_j = 2C_{j/2-1}$ for even $j > 0$, where $C_i = \frac{1}{i+1} \binom{2i}{i}$ is the i^{th} Catalan number.

This \mathcal{C} does not satisfy any of our other theorems.

Questions