NUS National University of Singapore

EE5139/EE6139
Solution 4

Semester I, AY2021/2022
Marco Tomamichel

Exercise 4.1 Infinite alphabet optimal code [all]

Let X be an i.i.d. random variable with an infinite alphabet, $\mathcal{X} = \{1, 2, 3, \ldots\}$. In addition, let $P(X = i) = 2^{-i}$.

a.) What is the entropy of X?

Solution: By direct calculation,

$$H(X) = \sum_{i=1}^{\infty} -2^{-i} \log(2^{-i}) = \sum_{i=1}^{n} i2^{-i} = 2.$$

This is because

$$\sum_{i=1}^{\infty} ix^{i-1} = \frac{1}{(1-x)^2}$$

for all $|x| < 1$. This can be shown by differentiation of the identity $\sum_{i=1}^{\infty} x^i = \frac{1}{1-x}$.

b.) Find an optimal variable length code, and show that it is indeed optimal.

Solution: Take the code lengths to be $-\log(2^{-1}), -\log(2^{-2}), -\log(2^{-3}), \ldots$. Codewords can be

$$\begin{aligned} C(1) &= 0 \\ C(2) &= 10 \\ C(3) &= 110 \\ &\vdots \end{aligned}$$

Exercise 4.2 Codeword lengths for Huffmann codes [all]

(from 2019/2020 quiz)

Consider a random variable X which takes on four possible values with probabilities $(\frac{1}{3}, \frac{1}{3}, \frac{1}{4}, \frac{1}{12})$.

a.) (5 points) Construct a Huffman code for this source.

Solution: Applying the Huffman algorithm gives us the table above, which gives codeword lengths (1, 2, 3, 3) or (2,2,2,2) for the different codewords.

Code1	Code2	Symbol	Probability
0	00	1	1/3
11	01	2	1/3
101	10	3	1/4
100	11	4	1/12

b.) (5 points) Show that there exist two different sets of optimal lengths for the codewords, namely, show that codeword length assignments $(1, 2, 3, 3)$ and $(2, 2, 2, 2)$ are both optimal.

Solution: Both set of lengths $(1, 2, 3, 3)$ and $(2, 2, 2, 2)$ satisfy the Kraft inequality, and they both achieve the same expected length (2 bits) for the above distribution. Therefore they are both optimal.

c.) (5 points) Are there optimal codes with codeword lengths for some symbols that exceed the Shannon code length $\lceil \log \frac{1}{p(x)} \rceil$?

Solution: The symbol with probability 1/4 has an Huffman code of length 3, which is greater than $\left\lceil \log \frac{1}{p(x)} \right\rceil$. Thus the Huffman code for a particular symbol may be longer than the Shannon code for that symbol. But on the average, the Huffman code cannot be longer than the Shannon code.

Exercise 4.3 A better Morse code [EE5139]

In the previous exercise we designed a code for English letters that had slightly lower expected length than the Morse code. Now let us look at this problem a bit more closely. For the Morse code, the codeword symbol "-" requires 4 time units to send, whereas "." only requires 2 time units. The end-of-letter symbol "_" requires 3 time units. So what we actually want to optimise is not the codeword length but the the *time* require to send it, e.g. for the codeword ".-._" this would be 11.

a.) Devise an algorithm that produces an alternative Morse code that optimises the expected time for a source producing English letters.

Solution: An idea is to list all codeword within certain transmission time limit, and assign faster signals to letters with higher frequencies. An algorithm is listed below. We denote the frequency of a letter in Table 1 by $f(\cdot)$, e.g., $f(\mathrm{a}) = 0.084$.

Let $\{A_1, A_2, \cdots, A_{26}\} = \{\mathrm{a}, \mathrm{b}, \ldots, \mathrm{z}\}$, and $f(A_i) \geq f(A_{i+1})$ for each $i \in \{1, 2, \ldots, 25\}$;
Initialize $t \leftarrow 2$, $j \leftarrow 0$;
do
  $\mathcal{C}_t \leftarrow \{(s_1, \ldots, s_n, \_) | s_1, \ldots, s_n \in \{., \text{-}\}, \sum_{i=1}^{n} 2 \cdot \mathbf{1}\{s_i = .\} + 4 \cdot \mathbf{1}\{s_i = \text{-}\} = t, n \in \mathbb{N}\}$;
  for $i = 1, \ldots, |\mathcal{C}_t|$ **do**
    if $j < 26$ **then**
      Assign the i-th element of $\mathcal{C}_t$ to A_{i+j};
      $j \leftarrow i$;
    end
  end
  $t \leftarrow t + 1$;
while $j < 26$;

A code table is listed below

a	-_	b	....._	c	.-.._	d	-.._	e	._	f	-..._	g	--._
h	.-._	i	.-_	j	..-.._	k	....-_	l	--	m	..-._	n	...._
o	-._	p	.--_	q	.-..._	r	..._	s	..-_	t	.._	u	....._
v	...-._	w	...-_	x	-...._	y	-.-_	z	--.._				

b.) Compute the expected time of transmission of the above code. Compute the expected time of the code produced in Exercise 3.3a, where we treat 0 as '.' and 1 as '-'.

a	8.4%	b	1.5%	c	2.2%	d	4.2%	e	11.0%	f	2.2%	g	2.0%
h	6.0%	i	7.4%	j	0.1%	k	1.3%	l	4.0%	m	2.4%	n	6.7%
o	7.4%	p	1.9%	q	0.1%	r	7.5%	s	6.2%	t	9.2%	u	2.7%
v	0.9%	w	2.5%	x	0.1%	y	2.0%	z	0.1%				

Table 1: Statistical distribution of letters in the English language. Source: `https://en.wikipedia.org/wiki/Letter_frequency`, but normalized so that they add up to 100%.

Solution: Expected time of transmission of the above code: 9.712.
Expected time of transmission of the code in Exercise 3.3a: 9.872.
(Note that we included '_' into the transmission time.)

c.) Can you show that it is optimal?

Solution: Let vector $\mathbf{f} = (f(A_1), f(A_2), \ldots, f(A_{26}), 0, 0, \ldots)$. Let set $\mathfrak{A}$ denote the set of all finite sequences made up of '.' and '-', ending with '_', namely

$$\mathfrak{A} = \{(s_1, s_2, \ldots, s_n, \_) : s_i \in \{.,\text{-}\} \text{ for each } i, n \in \mathbb{N}.\}$$

For each $\mathbf{s} \in \mathfrak{A}$, let $t(\mathbf{s})$ denote the transmission time of $\mathbf{s}$. We order the set $\mathfrak{A} = \{\mathbf{s}_i : i \in \mathbb{N}\}$ s.t. $t(\mathbf{s}_i) \leq t(\mathbf{s}_{i+1})$ for all $i \in \mathbb{N}$. Define the vector $\mathbf{t} = (t(\mathbf{s}_1), t(\mathbf{s}_2), \ldots)$.

For any code described in this exercise, its expected transmission time can be expressed as

$$\sum_i \mathbf{f}_i \cdot t(\mathbf{s}_{\pi_i}) = \sum_i \mathbf{f}_i \cdot \mathbf{t}_{\pi_i}$$

for some permutation π of natural numbers. Since $\mathbf{f} \geq 0$, $\mathbf{t} \geq 0$, and one is non-increasing, and the other is non-decreasing,

$$\sum_i \mathbf{f}_i \cdot \mathbf{t}_{\pi_i} \geq \sum_i \mathbf{f}_i \cdot \mathbf{t}_i.$$

Since the right-hand side is the expected transmission time of the code we constructed in a), we have shown the optimality.

d.) Can you come up with a prefix code that attempts to minimise the expected transmission time? This code does not need the end-of-letter symbol. Your algorithm can be heuristic (you do not need to prove optimality). Compute the expected transmission time and compare it to the code from a.

Solution: We present a heuristic algorithm here. (The idea of the algorithm is to view a prefix code as a process to divide the alphabet recursively until each subset has only one element. Intuitively speaking, for prefix codes with two balanced symbols, it makes sense to have each division to be as balanced as possible so that each symbol reduces maximum amount of uncertainty. In our case, one of the symbol costs twice as that of the other symbol. Thus, it makes sense to design the division so that choosing the shorter symbol twice should reduce the same amount of uncertainty as choosing the longer symbol.)

To solve this problem exactly, we refer to Golin and Rote's 1998 paper "A dynamic programming algorithm for constructingoptimal prefix-free code with Unequal letter cost", in which they cleverly converted the problem of finding the optimal coding tree into finding the shortest path in a "signature tree". For your reference, we also include a MatLab implementation of their algorithm for our problem.

Listing 1: Unbalanced_prefix_code.m

```
A = 1:26;
p = [8.4, 1.5, 2.2, 4.2, 11.0, 2.2, 2.0, 6.0, 7.4, 0.1, 1.3, 4.0, 2.4, 6.7,
    7.4, 1.9, 0.1, 7.5, 6.2, 9.2, 2.7, 0.9, 2.5, 0.1, 2.0, 0.1];
p = p./sum(p);

P = {A, p};
q = (sqrt(5)-1)/2;

[C, X] = unbalanced_prefix_code (P, q);

```

```
C_sorted = cell(length(X),1);
for k = 1:length(X)
    C_sorted{X(k)} = C{k};
end
disp(C_sorted);

t = 0;
for x = 1:26
    t = t + p(x)*2*(length(C_sorted{x})+sum(C_sorted{x}));
end
disp(t);

function [C, X] = unbalanced_prefix_code (P, q)
    if length(P{1}) == 1
        C = {[]};
        X = P{1};
    else
        C = cell(length(P{2}),1);
        X = [];
        [Pl, Pr] = divide(P,q);
        [Cl, Xl] = unbalanced_prefix_code(Pl, q);
        [Cr, Xr] = unbalanced_prefix_code(Pr, q);
        X = [Xl, Xr];
        for k = 1:length(Xl)
            C{k} = [0,Cl{k}];
        end
        for k = 1:length(Xr)
            C{k+length(Xl)} = [1,Cr{k}];
        end
    end
end

function [Pl, Pr] = divide (P, q)
% We attempt to divide the normalized weighted set P={A,p} into Pl and Pr so
     that the total weight of Pl is as close to q as possible
% However, checking all subsets of A take exponential amount of time. Thus,
    here, we use a greedy algorithm.
    Al = [];
    pl = [];
    Ar = P{1};
    pr = P{2}/sum(P{2});
    % Al cannot be empty: Pick at least one element
    q_current = sum(pl);
    d_current = abs(q_current - q);
    j = 0;
    d = inf;
    for k = 1:length(Ar)
        if abs(q_current + pr(k) - q) < d
            d = abs(q_current + pr(k) - q);
            j = k;
        end
    end
```

```
    Al = [Al, Ar(k)];
    pl = [pl, pr(k)];
    Ar = [Ar(1:k-1), Ar(k+1:length(Ar))];
    pr = [pr(1:k-1), pr(k+1:length(Ar))];
   % Pick additional elements
    while (1)
        q_current = sum(pl);
        d_current = abs(q_current - q);
        if length(Ar) == 1
            break;
        end
        j = 0;
        d = inf;
        for k = 1:length(Ar)
            if abs(q_current + pr(k) - q) < d
                d = abs(q_current + pr(k) - q);
                j = k;
            end
        end
        if d < d_current
            Al = [Al, Ar(k)];
            pl = [pl, pr(k)];
            Ar = [Ar(1:k-1), Ar(k+1:length(Ar))];
            pr = [pr(1:k-1), pr(k+1:length(Ar))];
        else
            break;
        end
    end
    Pl = {Al, pl};
    Pr = {Ar, pr};
end
```

The code table generated by the above program looks like the below.

a	---	b	--..	c	--.-	d	-..-	e	-...	f	-.-.-	g	-.-..
h	-.--	i	..--	j	..-..-	k	..-...	l	..-.-	m	....-	n	.....
o	.....-	p	...-.-	q	...-..	r	...--	s	-.-.	t	.-..	u	.--.-
v	.--...	w	.--..-	x	-...-.	y	-...--	z	.----				

The expected time of transmission of the above code is 12.802.

Listing 2: Golin_Rote.m

```
% Here, we implement the algorithm in the 1998 paper by Golin and Rote for
% the special case r=2, and n = 26
n = 26;
p = [8.4, 1.5, 2.2, 4.2, 11.0, 2.2, 2.0, ...
     6.0, 7.4, 0.1, 1.3, 4.0, 2.4, 6.7, ...
     7.4, 1.9, 0.1, 7.5, 6.2, 9.2, ...
     2.7, 0.9, 2.5, 0.1, 2.0, 0.1]/100;
 P = p;
 A = {'a', 'b', 'c', 'd', 'e', 'f', 'g', ...
     'h', 'i', 'j', 'k', 'l', 'm', 'n', ...
     'o', 'p', 'q', 'r', 's', 't', ...
```

```
     'u', 'v', 'w', 'x', 'y', 'z'};
[p, idx] =  sort(p,'descend');
A_num = idx;
% Now we implement the shortest-path-in-signature-tree algorithm
cost = zeros(n+1,n+1,n+1);
pre = cell(n+1,n+1,n+1);
update = zeros(n+1,n+1,n+1);
for k = 1:length(cost(:))
    cost(k) = Inf;
end
cost(0+1,1+1,1+1) = 0;
update(0+1,1+1,1+1) = 1;

while (sum(update(:)) > 0)
    [M, L1, L2] = ind2sub([n+1,n+1,n+1],find(update));
    M = M - 1; L1 = L1 - 1; L2 = L2 - 1;
    update = zeros(n+1,n+1,n+1);
    for k = length(M)
        m = M(k);
        l1 = L1(k);
        l2 = L2(k);
        new_cost = cost(m+1, l1+1, l2+1);
        for t = m+1:1:n
            new_cost = new_cost + p(t);
        end
        for q = 0:1:l1
            m_next = m+l1-q;
            l1_next = l2+q;
            l2_next = q;
            if m_next+l1_next+l2_next <=n
                if new_cost < cost(m_next+1, l1_next+1, l2_next+1)
                    cost(m_next+1, l1_next+1, l2_next+1) = new_cost;
                    pre{m_next+1, l1_next+1, l2_next+1} = [m, l1, l2];
                    update(m_next+1, l1_next+1, l2_next+1) = 1;
                end
            end
        end
    end
end

expected_time = cost(n+1, 1, 1)*2;

% Now list the path from (0,1,1) to (n,0,0)
path = [n,0,0];
m = n; l1 = 0; l2 = 0;
while (~isempty(pre{m+1, l1+1, l2+1}))
    PRE = pre{m+1, l1+1, l2+1};
    m = PRE(1);
    l1 = PRE(2);
    l2 = PRE(3);
    path = [m, l1, l2; path];
end
```

```
% The first row of path must be (0,1,1).

% Create the prefix code graph from the path
signature = [0,1,1];
graph = {[0], [1]}; % We only need to keep track of leaves.
D = 0;
for k = 2:size(path,1)
    D = D + 1;
    next_signature = path(k,:);
    q = next_signature(3);
    % change q—number of depth—D leaves in graph into parents
    populated = 0;
    new_graph = {};
    for j = 1:length(graph)
        leaf = graph{j};
        depth = sum(leaf+1);
        if depth == D && populated < q
            new_graph{length(new_graph)+1} = [leaf,0];
            new_graph{length(new_graph)+1} = [leaf,1];
            populated = populated + 1;
        else
            new_graph{length(new_graph)+1} = leaf;
        end
    end
    graph = new_graph;
end
% Sort the leaves in the graph (n == length(graph))
time = zeros(1,n);
for k = 1:n
    time(k) = 2*sum(graph{k}+1);
end
[time, idx] = sort(time);
code = cell(1,n);
for k = 1:n
    codeword = graph{idx(k)};
    code{A_num(k)} = codeword;
end
for k = 1:n
    disp([A{k},': ',num2str(code{k}), ' (', num2str(P(k)), ')']);
end
disp(expected_time);
```

The code table generated by the above program looks like the below.

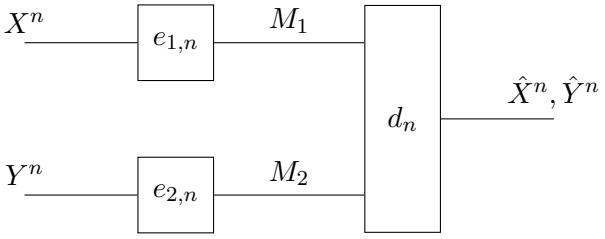
a	.--.	b	-.--	c	..-.-	d	.....-	e	.-...	f	..--.	g	.-..-
h	.....	i	-..-	j	....--	k	--.-	l	....-.	m	..-...	n	--..
o	-.-.	p	-...-	q	...-.-	r	-....	s	---	t	.-.-	u	...-..
v	.....-	w	...--	x	..-..-	y	.---	z	..---				

The expected time of transmission of the above code is 12.750.

Exercise 4.4 Converse for the Slepian–Wolf coding problem [all]

Let X and Y be a pair of jointly distributed random variables. (X is distributed on finite set $\mathcal{X}$, and Y is distributed on finite set $\mathcal{Y}$.) An $(n, 2^{nL_1}, 2^{nL_2})$-separately-encoded-jointly-decoded source code consists of a pair of encoders e_1, e_2, and a decoder d, where

- $e_1 : \mathcal{X}^n \to \{0,1\}^{nL_1}$,
- $e_2 : \mathcal{Y}^n \to \{0,1\}^{nL_2}$, and
- $d : \{0,1\}^{nL_1} \times \{0,1\}^{nL_2} \to \mathcal{X}^n \times \mathcal{Y}^n$.



The rate pair (R_1, R_2) is said to be achievable for DMS (X, Y) if there exists a sequence of $(n, 2^{nL_1}, 2^{nL_2})$-codes with encoders $e_{1,n}$, $e_{2,n}$ and decoder d_n such that

$$\lim_{n\to\infty} P\{(\hat{X}^n, \hat{Y}^n) \neq (X^n, Y^n)\} = 0$$

where

$$(\hat{X}^n, \hat{Y}^n) = d_n(M_1, M_2),\ M_1 = e_{1,n}(X^n),\ \text{and}\ M_2 = e_{2,n}(Y^n)$$

are the reconstructed source and codewords respectively.
Prove that, for any (R_1, R_2) achievable, it must hold that

$$R_1 \geq H(X|Y), \tag{1}$$
$$R_2 \geq H(Y|X), \tag{2}$$
$$R_1 + R_2 \geq H(X, Y). \tag{3}$$

Solution: By considering (M_1, M_2) as a single message, (3) follows directly from the proof in the lecture notes. We are only going to show the proof for (1) below. The proof for (2) follows symmetrically.
Suppose (R_1, R_2) is achievable. For any $\epsilon > 0$, there exists some $N \in \mathbb{N}$ such that for all $n > N$

$$P\{\hat{X}^n \neq X^n\} \leq P\{(\hat{X}^n, \hat{Y}^n) \neq (X^n, Y^n)\} < \epsilon.$$

By Fano's inequality, we have

$$I(\hat{X}^n : X^n) = nH(X) - H(\hat{X}^n|X^n) \geq nH(X) - \epsilon n \log|\mathcal{X}| - 1.$$

On the other hand, by how M_1 and M_2 are constructed, we have

$$\begin{aligned} I(\hat{X}^n : X^n) &\overset{(a)}{\leq} I(M_1, M_2 : X^n) \\ &\overset{(b)}{=} I(M_1 : X^n) + I(M_2 : X^n) \\ &\overset{(c)}{\leq} I(M_1 : X^n) + I(X^n : Y^n) \\ &\leq nR_1 + nI(X : Y) \end{aligned}$$

where (a) follows from the Markov chain $X^n - (M_1, M_2) - \hat{X}^n$, (b) follows from the Markov chain $M_1 - X^n - M_2$, and (c) follows from the Markov chain $M_2 - Y^n - X^n$. Thus,

$$nH(X) - \epsilon n \log |\mathcal{X}| - 1 \leq nR_1 + nI(X : Y),$$

which implies

$$R_1 \geq H(X|Y) - \epsilon \log |\mathcal{X}| - \frac{1}{n}.$$

Since above holds from all $\epsilon > 0$, and all $n > N_\epsilon$, one must have $R_1 \geq H(X|Y)$.