دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکدهٔ برق و کامپیوتر

مخابرات دیجیتال

گزارش تمرین کامپیوتری ۱

سید علیرضا جاوید

۸۱۰۱۹۸۳۷۵

استاد

دکتر ربیعی

۱۹ فروردین ۱۴۰۱

فهرست مطالب

۱ بدست آوردن نرخ آنتروپی منبع مارکف ایستان

در یک منبع با حافظه، برای نرخ آنتروپی آن می توان نوشت:

$$G_k = \frac{H(s_1) + H(s_2|s_1) + H(s_3|s_2, s_1) + ...}{k}$$

از خاصیت مارکف بودن می دانیم که حالت بعدی سیستم تنها به حالت فعلی آن وابسته است و به حالت های گذشته بستگی ندارد. همچنین از ایستان بودن منبع نیز استفاده کرده و رابطه قبل را به شکل زیر بازنویسی می کنیم:

$$G_k = \frac{H(s_1) + k \times H(s_2|s_1)}{k}$$

برای محاسبه نرخ آنتروپی کافی است تنها $H(s_1)$ و $H(s_2|s_1)$ را بدست آوریم. که در زیر توضیح داده می شود.

۱. احتمال هر استیت را بدست می آوریم:

$$P = \begin{bmatrix} p_1 \\ p_2 \\ ... \\ p_M \end{bmatrix} \qquad P = \phi^T P \quad \rightarrow \quad (\phi^T - I)P = 0$$

$$\sum_{i=1}^{M} p_i = 1$$

که با حل معادلات بالا ماتریس P را بدست می آوریم.

۲. محاسبه آنتروپی هر استیت:

$$H(s_1) = -\sum_{i=1}^{M} p(s = s_i) \log_2(p(s = s_i)) = -\sum_{i=1}^{M} p_i \log_2(p_i)$$

۳. محاسبه آنتروپی انتقال:

$$H(s_2|s_1) = -\sum_{i=1}^{M} p(s_1 = s_i) \sum_{j=1}^{M} p(s_2 = s_j|s_1 = s_i) \log_2(p(s_2 = s_j|s_1 = s_i))$$

$$H(s_2|s_1) == -\sum_{i=1}^{M} p_i \sum_{i=1}^{M} p_{ij} \log_2(p_{ij})$$

۴. محاسبه G_k از رابطه ابتدایی.

با پیاده سازی الگوریتم بالا در متلب کد زیر را می نویسیم:

```
function[Gk] = entropy(transition_states,k)
size = length(transition_states);
%step 1 : find P
ts_tp = transpose(transition_states);
% P = (phi)^T *P --> ((phi)^T - I) P = 0 --> Ax = 0
A = ts_tp - eye(size);
% add sigma(pi)= 1
ts = [A ; ones(1 , size)];
U = [zeros(size, 1) ;ones(1,1)] ;
%solve Ax = 0
P = linsolve(ts , U);
% step 2 : find Hs1
Hs = 0;
for i = 1:size
Hs = Hs - P(i)* log2(P(i));
end
% step 2 : find Hs2s1
Hs2s1 = 0;
for i = 1:size
Hx = 0;
for j = 1:size
Hx = Hx - transition_states(i,j)*log2(transition_states(i,j));
end
Hs2s1 = Hs2s1 + P(i,1) * Hx;
end
% step 3 : calculate Gk
Gk = (Hs + k*Hs2s1) / k;
end
```

۲ بدست آوردن متوسط طول کد هافمن یک رشته

می توان گفت حل این مسئله به ۴ بخش کلی تقسیم می شود:

۱. کد کردن رشته به صورت گروه های k تایی(super symbol): طبق خواسته سوال ابتدا باید رشته داده شده را به صورت گروه های k تایی در آوریم. ممکن است که طول رشته اصلی بر k بخش پذیر نباشد، اضافه کردن بیت به رشته اصلی تا زمانی که بر k بخش پذیر باشد می تواند نتایج نادرستی در پی داشته باشد پس بهتر است که بیت های اضافه رشته اصلی را از انتهای آن حذف کنیم.

۲. تولید سمبل برای هر گروه k تایی متناظر: برای محاسبه تکرار هر super symbol و تشکیل کد هافمن باید به هر کدام از گروه های k تایی یک سمبل یکتا نسبت بدهیم بصورتی که برای هر گروه یکتا باشد. سپس رشته سمبل های متناظر را بدست می آوریم

۳. بدست آوردن احتمال متناظر هر کدام از گروه های k تایی: تکرار های هر super symbol را در رشته سمبل که در بخش ۳ بدست آوردیم، محاسبه می کنیم و با تقسیم بر کل تعداد super symbol ها احتمال متناظر را بدست می آوریم.

۴. بدست آوردن طول متوسط رشته به کمک تابع huffmandict در متلب: با دادن ماتریس احتمال متناظر هر کدام از گروه های k تایی و رشته سمبل تولید شده در بخش ۲ طول متوسط رشته را بدست می آوریم.

با پیاده سازی الگوریتم بالا در متلب کد زیر را می نویسیم:

```
function [average_length_huffman] = average_length(chain , k)
%step1: make super-symbol with k-bit groups
res = mod(length(chain),k);
len = length(chain) - res;
normalize_chain = zeros(1, len);
for i = 1:(len)
normalize_chain(1,i) = chain(1,i);
end
group_num = length(normalize_chain)/k;
group_chain = reshape(normalize_chain, [k group_num]);
%step2: assigning symbols to k-bit sequences
symbol_chain = zeros(1, group_num);
for i = 1:group_num
separate_chain = group_chain(:,i);
symbol = 0;
for j = 1:k
symbol = symbol + separate_chain(j) * 2^(j);
end
symbol_chain(1,i) = symbol;
end
%step3: calculating symbol occurance probability
symbols = unique(symbol_chain);
symbol_count = histc(symbol_chain, symbols);
symbol_probability = zeros(1, length(symbols));
for i = 1 : length(symbols)
symbol_probability(1,i) = symbol_count(1, i) /length(...
 symbol_chain);
```

```
    end
    %step4: using huffmandict to find average length
    [¬,average_length_huffman] = huffmandict(symbols,...
    symbol_probability) ;
    end
```

۳ شبیه سازی منبع با حافظه و بدست آوردن پارامتر های آن

ابتدا منبع داده شده را به صورت تحلیلی بررسی می کنیم:

$$\phi = \begin{bmatrix} 0.5 & 0.5 \\ 0.8 & 0.2 \end{bmatrix}$$

$$P = \phi^T P \quad , \quad \sum_i p_i = 1$$

$$\rightarrow \quad P = \begin{bmatrix} 0.6154 \\ 0.3826 \end{bmatrix}$$

$$H(X_1) = -0.5 \times \log_2(0.5) \times 2 = 1 \quad , \quad H(X_2) = -0.2 \times \log_2(0.2) - 0.8 \times \log_2(0.8) = 0.7219$$

$$H(X) = \sum_{i=1}^{2} P_i \times H(X_i) = 0.6154 \times 1 + 0.3826 \times 0.7219 = 0.8930$$

برای تولید نمونه های تصادفی از تابع rand متلب کمک می گیریم و با مقایسه عدد تصادفی تولید شده و استیت فعلی، استیت بعدی و سمبل تولیدی را مشخص می کنیم.

```
temp  = rand;
switch chain(1,i)
case 1
if temp < transition_states(1,1)
chain(1,i+1) = 1 ;
else
chain(1,i+1) = 2 ;
end
case 2
if temp < transition_states(2,2)
chain(1,i+1) = 2 ;
else
chain(1,i+1) = 1 ;
end
end

```

رشته سمبل تولید شده در این بخش را با توابع بخش های قبل مورد آزمایش قرار داده و نتایج را تحلیل می کنیم.

۱.۳ نرخ آنتروپی (G_k)

در بخش قبل بدست آوردیم:

$$G_k = \frac{H(s_1) + k \times H(s_2|s_1)}{k}$$

از آنجایی که $H(s_2|s_1)$ همان آنتروپی منبع یا $H(X)$ است، انتظار داریم با افزایش k در نهایت به آن میل کند.

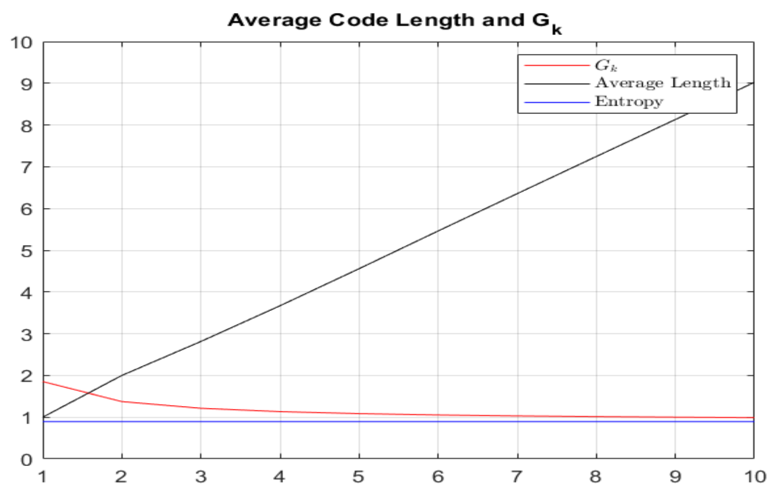
$$\lim_{k \to \infty} G_k = H(s_2|s_1) = H(X)$$

در این مسئله ما $H(X) = 0.8930$ بدست آوردیم.

۲.۳ متوسط طول کد هافمن

با افزایش k و تشکیل سمبل هایی با طول بزرگتر انتظار داریم که میزان redundancy کم تر شده و آنتروپی تخمین زده شده ($\hat{H}_k = \frac{\hat{l_k}}{k}$) به مقدار واقعی نزدیک تر شود.

در شبیه سازی مطابق شکل ۱ می توان دید با افزایش k طول متوسط کد به صورت تقریبا خطی افزایش یافته و در شکل ۲ نیز در ادامه مشاهده می شود که مقدار $\hat{H}_k$ به آنتروپی منبع نزدیک می شود.



شکل ۱: متوسط طول کد و G_k به ازای تغییرات k

۳.۳ بهره کد (Coding efficiency)

از محاسبات تئوری می دانیم:

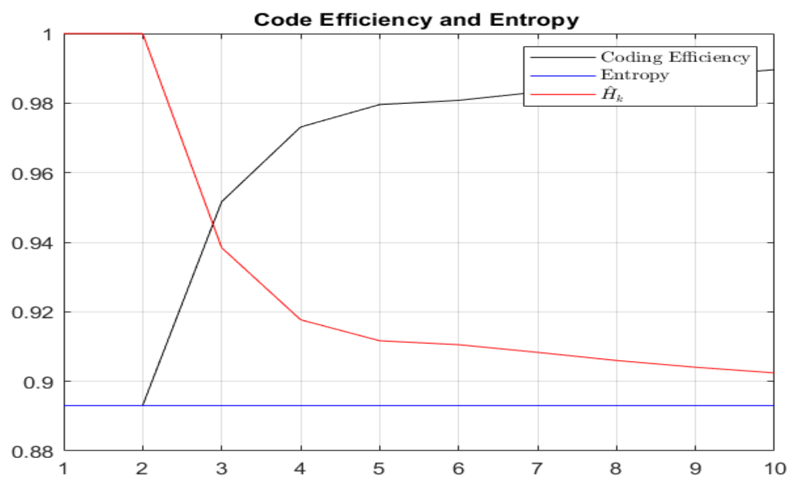
$$\hat{H}_k = \frac{\hat{l_k}}{k} \quad , \quad \eta_k = \frac{\hat{H}_k}{H(X)}$$

همچنین انتظار داریم با افزایش k بهره کد در نهایت به ۱ میل کند.

$$\lim_{k\to\infty} \hat{H}_k = H(X)$$

$$\lim_{k\to\infty} \eta_k = 1$$

در شکل ۲ می توان دید نتایج بدست آمده مطابق انتظار است.



شکل ۲: بهره کد و $\hat{H}_k$ به ازای تغییرات k

- فایل متلب مربوط به این بخش شامل جزئیات پیاده سازی آن با نام problem3.mlx پیوست شده است .

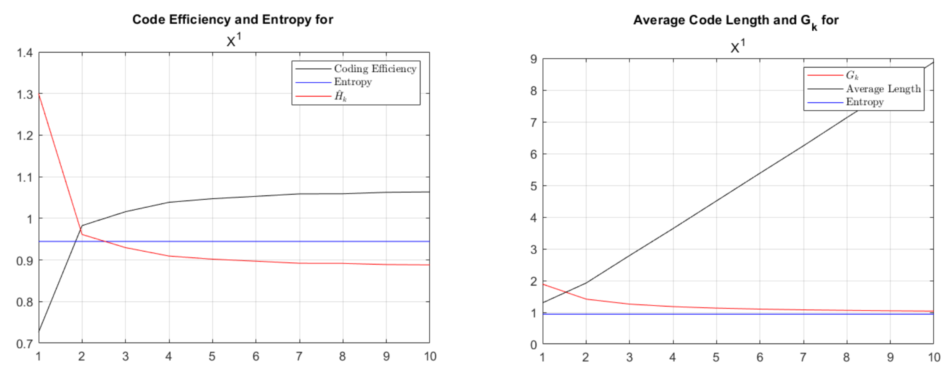
۴ شبیه سازی منبع بی حافظه و بدست آوردن پارامتر های آن

یک منبع بی حافظه را می توان به صورت یک منبع مارکف با تنها یک استیت مدل کرد که با احتمال های مختلف، سمبل های متفاوتی می سازد. پس روابط پیشین باز هم برقرار است. اما با تغییراتی همراه می باشد.

$$H(s_1) = H(X) \quad , \quad H(s_2|s_1) = H(s_2) = H(X)$$

$$G_k = \frac{H(s_1) + k \times H(s_2|s_1)}{k} = (\frac{k+1}{k})H(X)$$

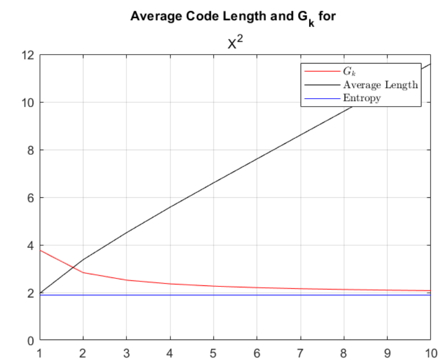
احتمال تولید سمبل برای منبع X^2 و X^3 از تابع korn در متلب استفاده می کنیم. برای تولید نمونه های تصادفی نیز از تابع randsrc با ورودی های مناسب استفاده می کنیم. لازم است تا تابع بخش یک را برای منبع بی حافظه ویرایش کنیم. با انجام این کار و همچنین استفاده از تابع بخش دوم در شبیه سازی نتایج زیر را به دست می آوریم.



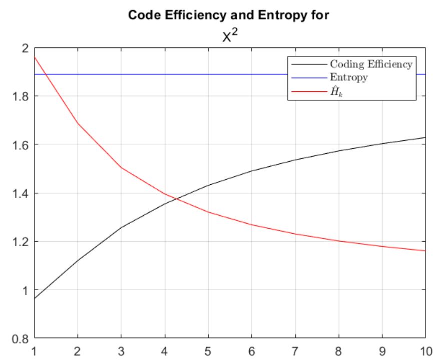
(آ) متوسط طول کد و G_k به ازای تغییرات k

(ب) بهره کد و $\hat{H}_k$ به ازای تغییرات k

شکل ۳: پارامتر های مبتع X^1

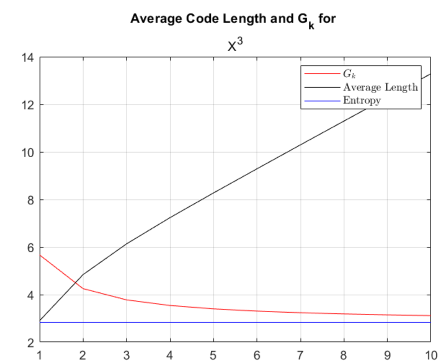


(آ) متوسط طول کد و G_k به ازای تغییرات k

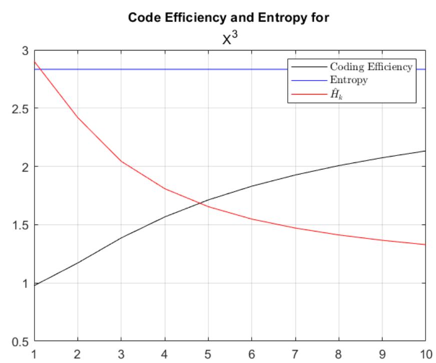


(ب) بهره کد و $\hat{H}_k$ به ازای تغییرات k

شکل ۴: پارامتر های مبتع X^2



(آ) متوسط طول کد و G_k به ازای تغییرات k



(ب) بهره کد و $\hat{H}_k$ به ازای تغییرات k

شکل ۵: پارامتر های مبتع X^3

به دلیل تقریب های لحاظ شده تا حدودی نمی توان نتایج دقیقی بدست آورد اما به صورت کلی داریم:

۱. طول متوسط کد هافمن منبع بصورت تقریبا خطی افزایش می یابد.

۲. مقدار G_k با افزایش k به میزان آنتروپی منبع نزدیک می شود.

۳. بهره کد منبع تقریبا مقدار ثابتی دارد و با افزایش k شاهد کمی افزایش است.

- فایل متلب مربوط به این بخش شامل جزئیات پیاده سازی آن با نام problem4.mlx پیوست شده است .

۵ مقایسه پارامتر های منبع با حافظه و بی حافظه

با تقریب نتایج زیر را می توان از مقایسه مشاهدات بخش ۳ و ۴ بدست آورد. مشاهدات برای k های کوچک تر دارای دقت بهتری می باشد اما برای k های زیاد به دلیل محدود بودن N نتایج از دقت کمتری برخوردار می باشد.

۱. در منبع بی حافظه به دلیل نبود redundancy ناشی از احتمال شرطی، با افزایش k شاهد تغییرات چشم گیری در بهره کد نیستیم و طول متوسط کد هافمن منبع نیز به صورت خطی افزایش می یابد. اما در منبع مارکف (با حافظه) با تشکیل سوپر سمبل از سمبل های ابتدایی، احتمال تشکیل برخی سمبل ها کاهش می یابد و میزان redundancy ناشی از احانال شرطی کم تر شده و بهره کد افزایش می یابد.

۲. منبع بدون حافظه به صورت چشم گیری دارای آنتروپی و بهره کد بیشتری نسبت به منبع با حافظه است که دلیل آن نایقینی بیشتر در تولید سمبل های تصادفی در منبع بی حافظه می باشد.

۳. در هر دو منبع با افزایش k مقدار G_k به صورت حدی به آنتروپی میل می کند و سرعت همگرایی در هر دو منبع سرعت همگرایی تقریبا یکسان می باشد.