Chapter 7: Source Coding

7.1. Basic of the coding theory

Decimal	Binary	BCD
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	1010	0001 0000
11	1011	0001 0001
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101

- A source generates one symbol from source alphabet at a time
- Normally, source alphabet is finite
  - S = {s1, s2, …, sq}   while q is |S| or number of symbols of source S
- Coding: use finite set of code symbols (code alphabet) to represent a source symbol
  - Generally, code alphabet is denoted by X = {x1, x2,…,xr}
  - r is |X| or number of different code symbols
    - Called base of code
    - r = 2 : binary code
    - r ≠ 2: r-ary code
- Number of source symbols of a source (source alphabet) must be higher than the number of coding symbols in code alphabet
  - Create combinations of code symbols (string of symbols) for coding an source symbol
    - Use rule for combination
    - Available combinations: generated from rule
    - E.g. Rule in BCD code: each sequence of 4 bits is available combination
  - Each available combination used to represent (coding) an source symbol
    - Each combination that has information called codeword (code, word)
    - Combination not used to represent any information called “don’t care combination”
    - E,g, in BCD code: 0 is encoding by “0000”, 1 is encoding “0001”…..
- Code without “don’t care combination”: full code

7.1. Basic of the coding theory (Cont.)

- Coding:
  - Assigns each source symbol to an available combination
  - Create a codeword.
  - Rule of coding is usually expressed in the form of a code table
    - Code table completely describes the source code rule by listing the codeword encoding of all the source symbols $\{s_i \rightarrow C(s_i)$: i =1,2,…,q} while s_i is one source symbol, $C(s_i)$ is one codeword used to code source symbol s_i
      - $C(s_i)$ = xj1….xjl while xji is one code symbol of code alphabet at position i in codeword, xj is one code symbols of code alphabet X
  - Length of codeword is number of code symbols in the codeword (denoted l)
    - If codewords have same length of l: fixed length code
    - If codewords have different length: variable length code
    - E.g: length of BCD code is 4 (fixed length code)
  - Code: set of codeword of a source
- Encoding process is to replace every source symbols of the message generated from the source with a codeword
    - After encoding process, a message is converted to string of code symbols
    - E.g. Message 23. After encoding: "00100011" (string of code symbols)
      - Source symbol " 2" encoded as "0010" (one codeword, also knows as, string of code symbols)
      - Source symbol " 3" encoded as "0011" (one codeword, also knows as, string of code symbols)

7.1. Basic of the coding theory (Cont.)

- Decoding process:
  - Detach received string of code symbols to codewords
  - Convert one codeword to one source symbol using code table
  - E.g: received message “00100011”
    - Detach to “0010 –0011 “
    - Convert “0010” to 2, “0011” to 3
    - Receive 23

7.1. Basic of the coding theory(Cont.)

- Types of code:
  - Singular code: code has distinct codewords
    - E.g: “01” is first codeword so that “01” must not be another codeword
    - Normally the codes are singular code
  - Uniquely decodable code:
    - Singular code
    - Results of decoding is the transmitted message
      - Detachable: from received string of code symbols, it exists only one way of detach it into codeword
        - Detachable code: each string of codeword does not coincide with another codeword string
      - Each codeword corresponds only to one source symbol (bijection between X and S)

7.1. Basic of the coding theory(Cont.)

- Types of code:
  - E.g. of detachable code:
  - Consider the source alphabet, S = {s1,s2,s3,s4}, and binary code, X= {0,1}.
  - The following are three possible binary source codes.

Source	Code A	Code B	Code C
s_1	0	0	00
s_2	11	11	01
s_3	00	00	10
s_4	11	010	11

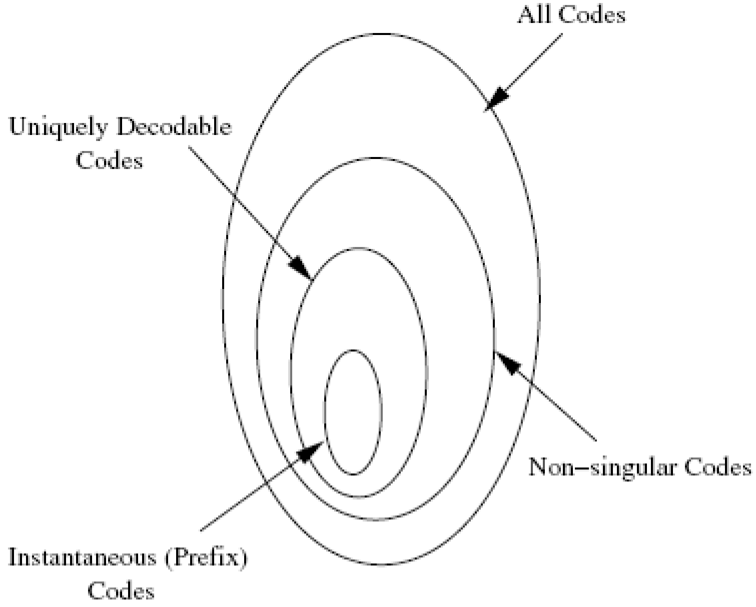
  - Code A: violate the singularity: “11” presented by both s2 and s4
  - Code B: violate the detachability: Message “s1s3” is encoded into “000” so that there are three detachable ways: “0-0-0”, “0-00”, and “0-00”. Results after decoding process are: “s1-s1-s1”, “s1-s3”, and “s3-s1”
  - Code C: OK

7.1. Basic of the coding theory(Cont.)

- Types of code:
  - Instantaneous code:
    - Uniquely code
    - After receiving the final symbol of codeword, the codeword can be detached immediately
    - Code needs to have “prefix”
      - Prefix of a codeword: Let C(S_i) = xj1xj2…xjl be a code word of length l. A sub-string of code characters from C(S_i), xj1xj2…xjm , where m < l, is called a prefix of the codeword C(S_i).
      - Prefix code: A codeword is not a prefix of another codeword
      - E.g. “10110” has prefixes: “1”, “10”, “101”, “1011”. These prefixes must not be any codeword in the code
      - E.g. of a prefix code: S = {s1,s2,s3,s4} has codewords “0”,”10”,”110”,”1110”

7.1. Basic of the coding theory(Cont.)

- Types of code:



7.1. Basic of the coding theory(Cont.)

- Construction of a prefix code:
  - Source has q symbols: need q codewords that has length {l1,l2,…,lq}.
  - To design the code: the codeword lengths are sorted in order of increasing length
  - The code words are derived in sequence such that at each step the current code word does not contain any of the other code words as a prefix
  - A systematic way to do this is to enumerate or count through the code alphabet.

7.1. Basic of the coding theory(Cont.)

- Construction of a prefix code:
  - Example 1. Need to construct an prefix binary code with length of 3,2,3,2,2
    - The lengths are re-ordered in increasing order as 2, 2, 2, 3, 3.
    - For the first three codewords of length 2, a count from 00 to 10 is used:

      00
      01
      10
    - For the next two code words of length 3, we count to 11, form 110
    - Then start counting from the right most symbol to produce the complete code:

      00
      01
      10
      110
      111

7.1. Basic of the coding theory(Cont.)

- Construction of a prefix code:
  - Example 2. Need to construct an prefix binary code with length of 2,3,1,1,2
    - The lengths are re-ordered in increasing order as 1, 1, 2, 2, 3
    - For the first two codewords of length 1:

      0
      1
    - For the next two codewords of length 2, we count from 2:

      20
      21
    - For the next codeword of length 3, we count from 22, form 220:

      0
      1
      20
      21
      220

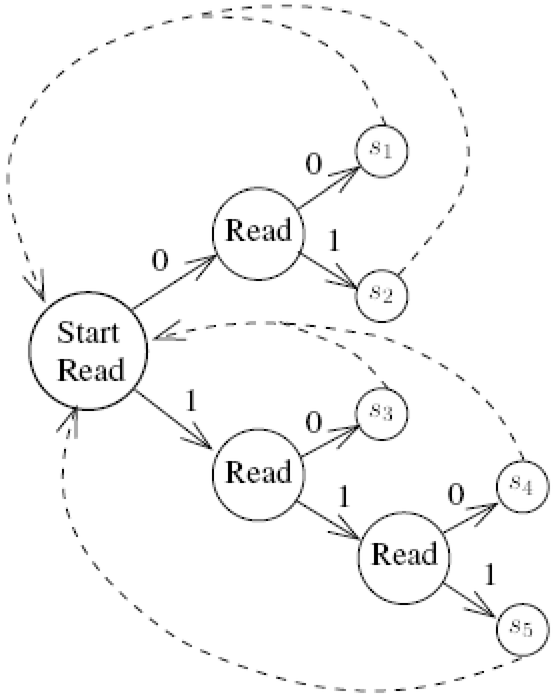
7.1. Basic of the coding theory(Cont.)

- Decoding prefix code:
  - From the first code symbol that is received:
    - Find a codeword
      - if found a final symbol of code word: detach a codeword without error
      - If not found: continue the next symbol
  - Normally, the used codes are prefix codes

7.1. Basic of the coding theory(Cont.)

- Decoding prefix code:

Source	Code
s_1	00
s_2	01
s_3	10
s_4	110
s_5	111



7.1. Basic of the coding theory(Cont.)

- Sensitivity to symbol errors:
  - Channel has noise: symbols are changed (symbol errors)
  - In case of fixed length code: to detach the codeword, only need to count the number of symbols (n).
    - If n = l: detach the codeword (no error in the detachable step)
  - In case of variable length code: if symbol are changed, one codeword may become a prefix of another codeword (error in the detachable step)
  - Prefix code with variable length code can not be used in the channel with noise
    - To overcome this issue: new special symbols are added in the end of each codeword
      - Special symbols are not used as any substring of codeword
      - Special symbols are difficult to changed to any substring of codeword
      - These special symbols are called detachable symbols
      - Code with these special symbols called code with detachable symbols
  - Remark:
    - Channel without noise uses prefix code
    - Channel with noise uses:
      - Fixed length prefix code
      - Code with detachable symbols

7.1. Basic of the coding theory(Cont.)

- Kraft Inequality:
  - Provides a limitation on the codeword lengths for the design of instantaneous codes.
  - A necessary and sufficient condition for the existence of an instantaneous code with alphabet size r and q codewords with individual codeword lengths of l1,l2…,lq is that the following inequality be satisfied:

$$\sum_{i=1}^{q} r^{-l_i} \leq 1$$

- *Conversely, given a set of code word lengths that satisfy this inequality, then there exists an instantaneous code with these word lengths.*

7.1. Basic of the coding theory(Cont.)

- Kraft Inequality:
  - Example: Consider binary codes (r = 2):

Source	Code A	Code B	Code C
s_1	0	0	0
s_2	100	100	10
s_3	110	110	110
s_4	111	11	11

  - Satisfied code?

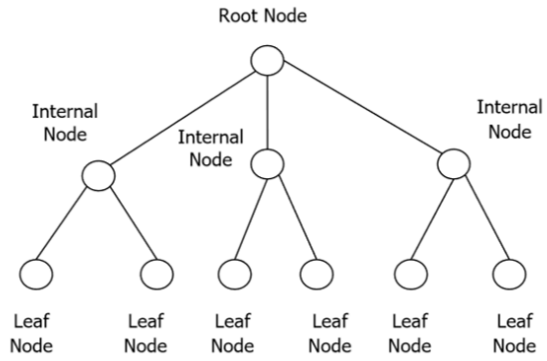
7.1. Basic of the coding theory(Cont.)

- Kraft Inequality:
  - Example: Consider binary codes (r = 2):

Source	Code A	Code B	Code C
s_1	0	0	0
s_2	100	100	10
s_3	110	110	110
s_4	111	11	11

- Code A satisfies because $\sum_{i=1}^{4} 2^{-l_i} = 2^{-1} + 2^{-3} + 2^{-3} + 2^{-3} = \frac{7}{8} \leq 1$
- Code B satisfies but not detachable because $\sum_{i=1}^{4} 2^{-l_i} = 2^{-1} + 2^{-3} + 2^{-3} + 2^{-2} = 1 \leq 1$
  - Prefix of codeword of s3 is codeword s4
  - Exist another codes which satisfies. Example: 0, 110, 111, 10
- Code C not satisfied because $\sum_{i=1}^{4} 2^{-l_i} = 2^{-1} + 2^{-2} + 2^{-3} + 2^{-2} = \frac{9}{8} > 1$

7.1. Basic of the coding theory(Cont.)



- Coding tree
  - A tree-shaped graph is used to represent the code
    - Start from root node (can also be called as 0-level node)
    - Each node emits at most r branches (r is code base)
    - Finish to leaf node
  - Each codeword is represented by a path from root node
    - Each branch represents a code symbol. The first code symbol of the codeword is the branch comes from root node
    - A node is the end of a branch that represents the last code symbol of the codeword called the last node of the codeword
  - Corollary
    - Prefix code: last node = leaf node
    - Fixed length code: leaf node in the same level
    - Full code: each node has r branches

7.2. Average length of codework and compact code

- Define $\{P_i: i = 1,2,\ldots,q\}$ as the individual source symbol probabilities for a source with q possible symbols.
- Define $\{l_i: i = 1,2,\ldots,q\}$ as the length of the corresponding codewords for a given source coding.
- The average length of the code, L, is given by:

$$L = \sum_{i=1}^{q} P_i l_i$$

  - Because source symbol s_i is coded using a codeword has length l_i
    → Probability of length l_i = probability of source symbol s_i (P_i)

7.2. Average length of codework and compact code (Cont.)

- Compact code:
  - Uniquely decodable code
  - Its average length is less than or equal to the average length of all other uniquely decodable codes for the same source and code alphabet.

Source	P_i	Code A	Code B
s_1	0.5	00	1
s_2	0.1	01	000
s_3	0.2	10	001
s_4	0.2	11	01

  - Code A has average length L_A= 2 bits / symbol (fixed length code)
  - Code B has average length L_B= (0.5)1 + (0.1)3 + (0.2)3 + (0.2)2 = 1.8 bits/symbol

→ Code B is better

7.3. Lower bound of average length

- Every instantaneous r-ary code of the source, S ={s1,s2,…,sq}, will have an average length, L, which is at least the entropy, H_r(S), of the source, that is:

$$L \geq H_r(S)$$

  - with equality when $P_i = r^{-l_i}$for i = 1,2,..,q where P_i is the probability of source symbol s_i
  - H_r(S) = $\frac{H(S)}{log_2 r}$ is the entropy of the source S using logarithms to the base r and H(S) is the entropy of the source S using logarithms to the base 2.

7.3. Lower bound of average length (Cont.)

- Code efficiency:

$$\eta = \frac{H_r(S)}{L} \times 100\%$$

  - Where if L = H_r(S) the code is 100% efficiency
    - Lmin = H_r(S) (code has minimum average length or average number of codes needed to encode a source symbol is minimal)
  - Special source: source with symbol probabilities {P_i : i= 1,2,…,q} such that {$log_r \frac{1}{P_i}$} are integers is a special source for r-ary codes since an instantaneous code with codeword lengths $l_i = log_r \frac{1}{P_i}$, for a code alphabet of size r can be designed which is 100% efficient with L = H_r(S)

7.3. Lower bound of average length (Cont.)

- Code efficiency:

$$\eta = \frac{H_r(S)}{L} \times 100\%$$

Source	P_i	**Code A**	**Code B**
s_1	0.5	00	1
s_2	0.1	01	000
s_3	0.2	10	001
s_4	0.2	11	01

7.3. Lower bound of average length (Cont.)

- Code efficiency:
  - Special source:
    - Example: 4-symbol source
      - Symbol probabilities are of the form $P_i = (\frac{1}{2})^{l_i}$
      - $l_1 = 3$, $l_2 = 2$, $l_3 = 1$ and $l_4 = 3$
      - A 100% efficient compact binary code can be designed because L = H(S) = 1.75
      - A code can be

Source A	P_i
s_1	0.125
s_2	0.25
s_3	0.5
s_4	0.125

Source A	Code A
s_1	110
s_2	10
s_3	0
s_4	111

7.4. Shannon’s noiseless coding theorem

- Let L be the average length of a compact r-ary code for the source S. Then:

$$H_r(S) \le L \le H_r(S) + 1$$

  - To obtain a efficient code, average length L must be minimal ($L = H_r(S)$)
  - Not special source: can not obtain the condition ($L > H_r(S)$
- The theorem states that the coding efficiency (which will always be less than 100% for compact codes that are not special) can be improved by coding the extensions of the source
  - Extensions of the source:
    - S^n: each symbol of S^n is a sequence of n symbols of source S
    - Entropy H(S^n) = n H(S)
  - Efficiency code for extension source has average length L_n.
  - Appling Shannon’s theorem for the S^n:

$$n\, H_r(S) \le L_n \le n\, H_r(S) + 1$$

  - Divide by n:

$$H_r(S) \le L_n/n \le H_r(S) + 1/n$$

  When n→ ∞: $L_n/n \rightarrow H_r(S)$: efficient code
- $H_r(S) \le L \le H_r(S) + 1$ considered as limit of the average length of the source code

7.4. Shannon's noiseless coding theorem (Cont.)

- Example:
  - binary coding scheme for a binary source

Source	P_i	Compact Code
s_1	0.8	0
s_2	0.2	1

- H(S) = -0.8 $log_2$0.8 – 0.2 $log_2$0.2 = 0.772 bits/symbol
- L = 1 bit/ symbol
- Efficiency = 77.2%

7.4. Shannon's noiseless coding theorem (Cont.)

- Example:
  - binary coding scheme for a binary source
    - H(S) = -0.8 $log_2 0.8$ – 0.2 $log_2 0.2$ = 0.772 bits/symbol
    - L = 1 bit/ symbol
    - Efficiency = 77.2%
  - second extension:
    - Each symbol of second extension source is sisj (si, sj ϵ S)
    - L_2 = (0.64)1 + (0.16)2 + (0.16)3 + (0.04)3 = 1.56 bits/ symbol
    - H(S^2)= -0.64 $log_2 0.64$ - 0.16 $log_2 0.16$ - 0.16 $log_2 0.16$ - 0.04 $log_2 0.04$ ≈ 1.444 bits/symbol
    - Efficiency = 1.444/1.56 ≈ 92.6%

Source	P_i	Compact Code
s_1	0.8	0
s_2	0.2	1

Source	P_i	Compact Code
$s_1 s_1$	0.64	0
$s_1 s_2$	0.16	10
$s_2 s_1$	0.16	110
$s_2 s_2$	0.04	111

7.6. Source coding with variable length code

- Source coding: use minimum number of code symbols to represent a source symbol which satisfies the limit of average length
  - Limit of average length (Shannon 's theorem): H_r(S) ≤ L ≤ H_r(S) +1
- Source code needs to have prefix
- Number of required code symbol (li) that is needed to code a source symbol is inversely proportional with probability of source symbol P(si)
  - Because a source symbol has amount of information = -log P(si)

→Algorithm to find a source code is algorithm to find all codewords that satisfies all three conditions

7.6. Source coding with variable length code

- Huffman code:
  - Proposed by Huffman
  - Assign each symbol a code word of length proportional to the amount of information conveyed by that symbol
  - Compact codes
    - Huffman algorithm produces a code with an average length, L, which is the smallest possible
  - Huffman algorithm: successively reducing a source with q source symbols to a source with r code symbols

7.6. Source coding with variable length code

- Huffman code:
  - Reduce source:
    1. Consider the source S with q source symbols: $\{s_i$: i = 1,2,..,q} with symbol probabilities {P(s_i): i = 1,2,…,q}
    2. Sort the symbols so that P(s_1) ≥ P(s_2) ≥ …. ≥ P(s_q)
    3. Replace last r-symbols of new sorted source symbols into one symbol S'_{q-r+1} with probability P(S'_{q-r+1}) = $\sum_{i=1}^{r} P(S_{q-r+i})$
    4. Obtained source after reducing denoted as S_1
    5. Successive reduced sources S_2 S_3 S_4… can be formed by a similar process of renumbering and combining until we are left with a source with only r symbols

7.6. Source coding with variable length code

- Huffman code:
  - It should be noted that we will only be able to reduce a source to exactly r symbols if the original source has q= r +α (r-1) symbols where α is a non-negative integer.
    - With binary code (r=2): the above condition always correct with any q$\geq$ 2
    - With non-binary code: if α = $\frac{q-r}{r-1}$ is not an integer value, then "dummy" source symbols with zero probability are appended to create a source with

$$q = r + \lceil \alpha \rceil \, (r-1)$$

where $\lceil \alpha \rceil$ is smallest integer greater than or equal to α

7.6. Source coding with variable length code

- Huffman code:
  - It should be noted that we will only be able to reduce a source to exactly r symbols if the original source has q= r +α (r-1) symbols where α is a non-negative integer.
    - With binary code (r=2): the above condition always correct with any q$\geq$ 2
    - With non-binary code: if α = $\frac{q-r}{r-1}$ is not an integer value, then "dummy" source symbols with zero probability are appended to create a source with

$$q = r + \lceil \alpha \rceil \, (r-1)$$

where $\lceil \alpha \rceil$ is smallest integer greater than or equal to α

7.6. Source coding with variable length code

- Huffman code:
  - In each reducing step, each source symbol in r-last symbols is assigned a code symbol
  - Each codework of each source symbol is a sequence of codework used to assign the source code itself and the symbols it contains.

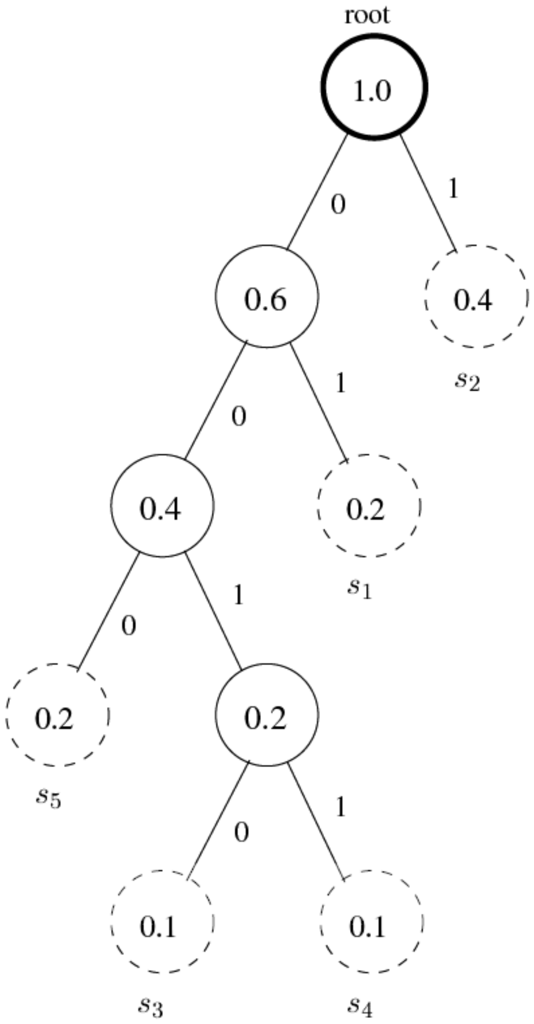
7.6. Source coding with variable length code (Cont.)

- Example:
  - Source has:

$$P(s_1) = 0.2 \quad P(s_2) = 0.4 \quad P(s_3) = 0.1 \quad P(s_4) = 0.1 \quad P(s_5) = 0.2$$

  - Apply Huffman algorithm for binary code:

	S	S_1	S_2	S_3
s_2	0.4 *1*	0.4 *1*	0.4 *1*	0.6 *0*
s_1	0.2 *01*	0.2 *01*	0.4 *00*	0.4 *1*
s_5	0.2 *000*	0.2 *000*	0.2 *01*	
s_3	0.1 *0010*	0.2 *001*		
s_4	0.1 *0011*			



7.6. Source coding with variable length code (Cont.)

- Example:
  - Source has q=11 :

    $P(s_1) = 0.16 \;\; P(s_2) = 0.14 \;\; P(s_3) = 0.13 \;\; P(s_4) = 0.12 \;\; P(s_5) = 0.10$

    $P(s_6) = 0.10 \;\; P(s_7) = P(s_8) = 0.06 \;\; P(s_9) = 0.05 \;\; P(s_{10}) = P(s_{11}) = 0.04$

  - Apply Huffman algorithm with r=4:
    - $\alpha = \frac{q-r}{r-1}$ =2.33 (not integer value) → $\lceil \alpha \rceil$ = 3
    - Get new q = 13 according to $q = r + \lceil \alpha \rceil (r-1)$
      - Add new dummy s12,s13 with P(s12) = P(s13) =0

7.6. Source coding with variable length code (Cont.)

- Example:
  - Original Source :

$$P(s_1) = 0.16 \;\; P(s_2) = 0.14 \;\; P(s_3) = 0.13 \;\; P(s_4) = 0.12 \;\; P(s_5) = 0.10$$

$$P(s_6) = 0.10 \;\; P(s_7) = P(s_8) = 0.06 \;\; P(s_9) = 0.05 \;\; P(s_{10}) = P(s_{11}) = 0.04$$

S		S_1		S_2		S_3	
0.16	*2*	0.16	*2*	0.25	*1*	0.45	*0*
0.14	*3*	0.14	*3*	0.16	*2*	0.25	*1*
0.13	*00*	0.13	*00*	0.14	*3*	0.16	*2*
0.12	*01*	0.12	*01*	0.13	*00*	0.14	*3*
0.10	*02*	0.10	*02*	0.12	*01*		
0.10	*03*	0.10	*03*	0.10	*02*		
0.06	*11*	0.08	*10*	0.10	*03*		
0.06	*12*	0.06	*11*				
0.05	*13*	0.06	*12*				
0.04	*100*	0.05	*13*				
0.04	*101*						
0.00	*102*						
0.00	*103*