International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 15 (2015) pp 36023-36029

Performance Comparison of Huffman Coding and Tunstall Coding

E.Thirunavukkarasu

Professor and Head Department of Computer Science and Engineering
Jairupaa College of Engineering Tiruppur, India.
Thirunavukkarasu465@gmail.com

Dr.G.Karuppusami

Dean – Research and Innovations Department of Mechanical Engineering
Sri Eshwar College of Engineering Coimbatore, India
Karuppusami.gks@gmail.com

Dr.P.Ezhilarasu

Associate Professor, Department of Computer Science and Engineering,
Hindusthan College of Engineering and Technology, Coimbatore - 641032, India
prof.p.ezhilarasu@gmail.com

Dr.N.Krishnaraj

Associate Professor, Department of Information Technology,
Valliammai Engineering College, Kattankulathur, Chennai 603203, India
drnkrishnaraj@gmail.com

Abstract
In this paper, performance of Huffman coding compared with Tunstall coding. Huffman coding produces variable length codes. Tunstall coding based on fixed length codes. Both the techniques are lossless. Encoding and Decoding performed for both data compression techniques. Its compression ratio, space savings, and average bits also calculated.

Keywords — Huffman Coding,Tunstall Coding, Encoding, Decoding**.**

INTRODUCTION

Data compression defined as the representation of data in such a way that, the storage area needed for target data is less than that of the size of the input data [1]. The actual data regenerated by decompression method. After decompression, if data quantity reduced, then the compression named as lossy compression. If data quantity retained without any damage, then the compression named as lossless compression. The Huffman coding and Tunstall coding comes under lossless compression. The former one is based on variable length coding method. The later one is based on fixed length coding method.

RELATED WORK

Shannon [1948] and Fano [1949] used the top down approach to generate optimal code for data compression [2 and 3]. Huffman, the student of Fano used the reversal approach (bottom up approach) to produce improved optimal code [4]. The Huffman code was better than Shannon-Fano code in terms of optimality. Both the coding produced variable length code to represent each unique character. Tunstall [1967] worked on the thesis related lossless data compression using top down approach [5]. It is based on assigning fixed length code for leaf nodes. The tree generated based on internal node with high probability, starting from the root node.

The performance of Tunstall techniques also analysed [6 and 7]. The Tunstall codes are accessible randomly [8]. Because, it is independent of prior blocks. The Huffman code is dependent on prior block. Therefore, they are not randomly accessible. The Huffman coding produces very nearly optimal code [9 and 10]. The algorithm based on Tunstall technique in sublinear time produced optimal code through input random variable [11 and 12].

HUFFMAN AND TUNSTALL CODING

The Huffman coding and Tunstall coding uses the probability of unique characters. The first technique starts from the two unique characters with least probability. The second technique splits the root node into unique characters. Then based on higher probability and number of leaf node they are broken into leaf nodes. Each node is having the probability which is based on the previous internal node probability.

The number of iterations used in Huffman coding is n-1. Where, n represents the number of unique characters. If, n = 11 then Huffman tree constructed after the 10^{th} iteration. In Tunstall coding, it creates n new leaf nodes for each iteration. The total number of leaf node calculated by the equation 1.

The total number of leaf node = (Total number of unique characters * number of iterations) – (number of iterations -1) (1)

For the first iteration the number of leaf nodes will be total number of unique characters.
If the given input is: "comparison between huffman coding and tunstall coding."
The unique characters and its occurrence derived from the input and shown in table 1.

Table.1. Occurrence of unique characters

S.No	Unique Character	Occurrence
1	n	7
2	space	6
3	o	4
4	a	4
5	i	3
6	e	3
7	t	3
8	d	3
9	c	3
10	m	2
11	s	2
12	u	2
13	f	2
14	g	2
15	l	2
16	p	1
17	r	1
18	b	1
19	w	1
20	h	1
21	.	1

The probability of each unique character calculated from the Table 1 and as shown in Table 2.

Table.2.Probability of unique characters

S.No	Unique Character	Occurrence	Probability
1	n	7	0.12963
2	space	6	0.111111
3	o	4	0.074074
4	a	4	0.074074
5	i	3	0.055556
6	e	3	0.055556
7	t	3	0.055556
8	d	3	0.055556
9	c	3	0.055556
10	m	2	0.037037
11	s	2	0.037037
12	u	2	0.037037
13	f	2	0.037037
14	g	2	0.037037
15	l	2	0.037037
16	p	1	0.018519
17	r	1	0.018519
18	b	1	0.018519
19	w	1	0.018519
20	h	1	0.018519
21	.	1	0.018519

The Huffman code derived from the Table 2 and as shown in Table 3

Table.3.Huffman code and length for the unique characters

S.No	Unique Character	Code	Length
1	n	011	21
2	space	100	18
3	o	0100	16
4	a	0101	16
5	i	1100	12
6	e	1101	12
7	t	1010	12
8	d	1011	12
9	c	1110	12
10	m	1111	8
11	s	00010	10
12	u	00011	10
13	f	00000	10
14	g	00001	10
15	l	00110	10
16	p	001010	6
17	r	001011	6
18	b	001000	6
19	w	001001	6
20	h	001110	6
21	.	001111	6
Total Length			225

Here the Tunstall code generated based on Huffman code. The factor taken is the maximum length of the Huffman code. The maximum length is 6. So, the number of permissible leaf node will be 2 powers 6 = 64. After each iteration, the number of leaf nodes checked. If it is less than the condition, then next iteration implemented. If the condition fails, then the previous iteration values are taken and the process stopped.
After the first iteration the root node creates 21 leaf nodes (Total number of unique characters), as given in the Table 4.

Table.4.Tunstall code – After the first iteration

S.No	Leaf Node	Probability
1	n	**0.12963**
2	space	0.111111
3	o	0.074074
4	a	0.074074
5	i	0.055556
6	e	0.055556
7	t	0.055556
8	d	0.055556
9	c	0.055556
10	m	0.037037
11	s	0.037037
12	u	0.037037
13	f	0.037037
14	g	0.037037

15	l	0.037037
16	p	0.018519
17	r	0.018519
18	b	0.018519
19	w	0.018519
20	h	0.018519
21	.	0.018519

After first iteration the probability of each leaf node calculated. The number of leaf nodes with the given condition checked. If the condition met(21<=64), then the leaf node(**'n' - 0.12963)** with the highest probability selected for the next iteration. After second iteration the number of leaf node using equation 1 calculated, as given in Table 5.

The total number of leaf node = (Total number of unique characters * number of iterations) – (number of iterations -1)
=(21-2)-(2-1)
=42-1
=41

Table.5.Tunstall code – After the second iteration

S.No	Leaf Node	Probability
1	**space**	**0.111111**
2	o	0.074074
3	a	0.074074
4	i	0.055556
5	e	0.055556
6	t	0.055556
7	d	0.055556
8	c	0.055556
9	m	0.037037
10	s	0.037037
11	u	0.037037
12	f	0.037037
13	g	0.037037
14	l	0.037037
15	p	0.018519
16	r	0.018519
17	b	0.018519
18	w	0.018519
19	h	0.018519
20	.	0.018519
21	nn	0.016804
22	nspace	0.014403
23	no	0.009602
24	na	0.009602
25	ni	0.007202
26	ne	0.007202
27	nt	0.007202
28	nd	0.007202
29	nc	0.007202
30	nm	0.004801
31	ns	0.004801
32	nu	0.004801
33	nf	0.004801
34	ng	0.004801
35	nl	0.004801
36	np	0.002401
37	nr	0.002401
38	nb	0.002401
39	nw	0.002401
40	hn	0.002401
41	n.	0.002401

After second iteration the probability of each leaf node calculated. The number of leaf nodes with the given condition checked. If the condition met(41<=64), then the leaf node**('space character' - 0.111111)** with the highest probability selected for the next iteration. After second iteration the number of leaf node using equation 1 calculated, as given in Table 6.
The total number of leaf node = (Total number of unique characters * number of iterations) – (number of iterations -1)
=(21-3)-(3-1)
=63-2
=61

Table.6.Tunstall code – After the third iteration

S.No	Leaf Node	Probability
1	o	0.074074
2	a	0.074074
3	i	0.055556
4	e	0.055556
5	t	0.055556
6	d	0.055556
7	c	0.055556
8	m	0.037037
9	s	0.037037
10	u	0.037037
11	f	0.037037
12	g	0.037037
13	l	0.037037
14	p	0.018519
15	r	0.018519
16	b	0.018519
17	w	0.018519
18	h	0.018519
19	.	0.018519
20	nn	0.016804
21	nspace	0.014403
22	no	0.009602
23	na	0.009602
24	ni	0.007202
25	ne	0.007202
26	nt	0.007202
27	nd	0.007202
28	nc	0.007202
29	nm	0.004801
30	ns	0.004801
31	nu	0.004801
32	nf	0.004801

33	ng	0.004801
34	nl	0.004801
35	np	0.002401
36	nr	0.002401
37	nb	0.002401
38	nw	0.002401
39	hn	0.002401
40	n.	0.002401
41	space n	0.014403292
42	space space	0.012345679
43	space o	0.008230453
44	space a	0.008230453
45	space i	0.00617284
46	space e	0.00617284
47	space t	0.00617284
48	space d	0.00617284
49	space c	0.00617284
50	space m	0.004115226
51	space s	0.004115226
52	space u	0.004115226
53	space f	0.004115226
54	space g	0.004115226
55	space l	0.004115226
56	space p	0.002057613
57	space r	0.002057613
58	space b	0.002057613
59	space w	0.002057613
60	space h	0.002057613
61	space .	0.002057613

After third iteration the probability of each leaf node calculated. The number of leaf nodes with the given condition checked. If the condition met(61<=64), then the leaf node**('o' - 0.074074)** with the highest probability selected for the next iteration. It will produce another 21 leaf node. The total number of leaf node after fourth iteration will be 81 (81 not less than or equal to 64). The condition fails. So, the process stops with the third iteration. The 61 leaf nodes are given fixed unique code, as shown in Table 7.

Table.7.Consolidated Tunstall code

S.No	Leaf Node	Code
1	o	000001
2	a	000010
3	i	000011
4	e	000100
5	t	000101
6	d	000110
7	c	000111
8	m	001000
9	s	001001
10	u	001010
11	f	001011
12	g	001100
13	l	001101
14	p	001110
15	r	001111
16	b	010000
17	w	010001
18	h	010010
19	.	010011
20	nn	010100
21	nspace	010101
22	no	010110
23	na	010111
24	ni	011000
25	ne	011001
26	nt	011010
27	nd	011011
28	nc	011100
29	nm	011101
30	ns	011110
31	nu	011111
32	nf	100000
33	ng	100001
34	nl	100010
35	np	100011
36	nr	100100
37	nb	100101
38	nw	100110
39	hn	100111
40	n.	101000
41	space n	101001
42	space space	101010
43	space o	101011
44	space a	101100
45	space i	101101
46	space e	101110
47	space t	101111
48	space d	110000
49	space c	110001
50	space m	110010
51	space s	110011
52	space u	110100
53	space f	110101
54	space g	110110
55	space l	110111
56	space p	111000
57	space r	111001
58	space b	111010
59	space w	111011
60	space h	111100
61	space .	111101

ENCODING

The given input is "comparison between huffman coding and tunstall coding.". The input symbols are replaced by the corresponding codes. After encoding using huffman coding the input will be as shown below.

111001001111001010010100101111000001001000111000010001101101000100111011101011100001110000110000000000111101010111001110010010111100011000011000101011

1011100101000011011000101010010100110001101001110010010111100011000010011 11
(length = 225)
21+18+16+16+12+12+12+12+12+8+10+10+10+10+10+6+6 +6+6+6+6 = 225

After encoding using Tunstall coding the input will be as shown below.
000111000001001000001110000010001111000011001001000001010101010000000100000101010001000100000100010101010010001010001011001011001000000010010101000111000001000110000011100001101100011011101111001010011110000101000010001101001101110001000001000110000011100001010011
(length = 264)
34(no of single letter leaf node)+10 (no of dual letter leaf node)=44*6(length of leaf node)=264.

DECODING

The decoding of Huffman to be implemented from left to right. In case of Tunstall coding as the length of the codes are fixed as 6 decoding implementation can be performed in any order. i.e left to right(or) right to left (or) randomly.
Huffman Decoding
(left to right)

Step 1
c0100111100101001010010111100000100100011100001000110110100010011101110101110000111000011000000000001111010101110011100100101111000110000110001010111

0111001010000110110001010100101001100011010011100100101111000110000100111 1

Step 2
co111100101001010010111100000100100011100001000110110100010011101110101110000111000011000000000001111010101110011100100101111000110000110001010111011100101000011011000101010010100110001101001110010010111100011000010011 11

Step 3
com0010100101001011110000010010001110000100011011010001001110111010111000011100001100000000000111101010111001110010010111100011000011000101011101110010100001101100010101001010011000110100111001001011110001100001001111

Step 4
comp0101001011110000010010001110000100011011010001001110111010111000011100001100000000000111101010111001110010010111100011000011000101011101110010100001101100010101001010011000110100111001001011110001100001001111

Step 5-54
comparison between huffman coding and tunstall coding.
Tunstall decoding
(left to right)

Step 1
c000001001000001110000010001111000011001001000001010101010000000100000101010001000100000100010101010010001010001011001011001000000010010101000111000001000110000011100001101100011011101111001010011110000101000010001101001101110001000001000110000011100001010011

Step 2
co001000001110000010001111000011001001000001010101010000000100000101010001000100000100010101010010001010001011001011001000000010010101000111000001000110000011100001101100011011101111001010011110000101000010001101001101110001000001000110000011100001010011

Step 3
com001110000010001111000011001001000001010101010000000100000101010001000100000100010101010010001010001011001011001000000010010101000111000001000110000011100001101100011011101111001010011110000101000010001101001101110001000001000110000011100001010011

Step 4
comp000010001111000011001001000001010101010000000100000101010001000100000100010101010010001010001011001011001000000010010101000111000001000110000011100001101100011011101111001010011110000101000010001101001101110001000001000110000011100001010011

Step 5-44
comparison between huffman coding and tunstall coding.

Tunstall decoding
(right to left)

Step 1
000111000001001000001110000010001111000011001001000001010101010000000100000101010001000100000100010101010010001010001011001011001000000010010101000111000001000110000011100001101100011011101111001010011110000101000010001101001101110001000001000110000011100001.

Step 2
000111000001001000001110000010001111000011001001000001010101010000000100000101010001000100000100010101010010001010001011001011001000000010010101000111000001000110000011100001101100011011101111001010011110000101000010001101001101110001000001000110000011ng.

Step 3
000111000001001000001110000010001111000011001001000001010101010000000100000101010001000100000100010101010010001010001011001011001000000010010101000111000001000110000011100001101100011011101111001010011110000101000010001101001101110001000001000110ing.

Step 4

0001110000010010000011100000100011110000110010010000010101010100000001000001010100010001000001000101010100100010100011001011001000000010010101000111000001000110000011100000110111000110111011110010100111100001010000100011010011011100010000010001101001101110001000001000110000011100001010011

Step 5-44

comparison between huffman coding and tunstall coding.

Tunstall decoding
(random place)

Step 1(bit 31-36 r)

0001110000010010000011100000010r000011001001000001010101010000000100000101010001000100000100010101010010001010001011001011001000000010010101000111000001000011000001110000110111000110111011110010100111100000100110000011100001101110001101110111100101001111000010010100011010011011100010000010001100000111000100101001

Step 2(bit 55-60 n and space)

0001110000010010000011100000010r000011001001000001n010000000100000101010001000100000100010101010010001010001011001011001000000010010101000111000001000110000011100001101100011011101111100101001111000010100010011010011011100010000010001100000111000100101001

Step 3(bit 259-264 .)

0001110000010010000011100000010r000011001001000001n010000000100000101010001000100000100010101010010001010001011001011001000000010010101000111000001000110000011100001101100011011101111100101001111000010100010011010011011100010000010001100000111000010100001.

Step 4(bit 229-234 space and c)

0001110000010010000011100000010r000011001001000001n010000000100000101010001000100000100010101010010001010001011001011001000000010010101000111000001000110000011100001101100011011101111100101001111000010100010011010011 c000001000110000011100001.

Step 5-44

comparison between huffman coding and tunstall coding.

RESULTS AND DISCUSSION

The performance comparison in terms of compression ratio, space savings and average bits calculated for the examples are as given in Table 8

Table.8.Performance Comparison

Performance aspects→ Compression Technique	Compression Ratio	Space Savings	Average Bits
Huffman Coding	432/225=1.92:1	(1-(1/1.92))*100 = 47.92%	4.17 bits per characters
Tunstall Coding	432/264=1.63:1	(1-(1/1.63))*100 =38.65%	6 bits per character

The Feature comparison in terms of encoding, decoding, approach, code nature given in the Table 9

Table.9.Feature Comparison

Performance aspects→ Compression Technique	Approach	Code nature	Tree construction	Encoding						Decoding		
				Steps	After encoding Message length	Order	Used Characters	Speed	Parallel Processing	Order	Parallel Processing	Speed
Huffman Coding	Bottom Up	Variable	n-1 steps (n- no of unique characters)	M m – length Of input	Less	Random	n	More	Allowed	Left To Right	Not Allowed	Less
Tunstall Coding	Top Down	Fixed	Based on the condition	<=m	More than Huffman code	First lengthy characters (leaf node) replaced	>=n	less	Allowed	random	Allowed	More If Parallel Processing used

CONCLUSION

The performance and features of Huffman and Tunstall coding compared with each other. The results shows that Huffman coding better than Tunstall coding in terms of performance through compression ratio, space savings, and average bits. In case of feature aspects Tunstall coding is better than Huffman coding in terms of decoding. Because Tunstall decoding allows random order decoding and parallel processing. But in encoding in many parameters Huffman coding beats Tunstall coding.

REFERENCE

[1] Ezhilarasu, P., N. Krishnaraj, and B. Dhiyanesh. "Arithmetic Coding for Lossless Data Compression–A Review." IJCST 3 (3), pp. 89-94, 2015.

[2] C. E. Shannon, "A Mathematical Theory of Communication", Bell System Technical Journal, Volume 27, pp. 398-403, 1948.

[3] R.M. Fano, "The transmission of information", Technical Report 65, Research Laboratory of Electronics, M.I.T, Cambridge, Mass, 1949.

[4] D.A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E, pp. 1098–1102, 1952.

[5] B. Tunstall, "Synthesis of noiseless compression codes," Ph.D. dissertation, Georgia Institute of Technology, 1967.

[6] J. Abrahams, "Code and parse trees for lossless source encoding," Communications in Information and Systems, vol. 1, no. 2, pp. 113– 146, Apr. 2001.

[7] I. Tabus, G. Korody, and J. Rissanen, "Text compression based on variable-to-fixed codes for Markov sources," in Proc., IEEE Data Compression Conf., Mar. 28–30, 2000, pp. 133–142.

[8] Y. Bugeaud, M. Drmota, and W. Szpankowski, "On the construction of (explicit) Khodak's code and its analysis," IEEE Trans. Inf. Theory , vol. IT-54, no. 11, pp. 5073–5086, Nov. 2008.

[9] R. M. Capocelli, R. Giancarlo & I. J. Taneja, "Bounds on the Redundancy of Huffman Codes," IEEE Trans. Inform. Theory, IT-32, pp. 854-857, 1986.

[10] R. G. Gallager, "Variations on a Theme by Huffman," IEEE Trans. Inform. Theory, IT-24, pp. 668-674,1978.

[11] J. Kieffer, "Fast generation of Tunstall codes," in Proc., 2007 IEEE Int. Symp. on Information Theory , June 24–29, 2007, pp. 76–80.

[12] Y. A. Reznik and A. V. Anisimov, "Enumerative encoding/decoding of variable-to-fixed-length codes for memoryless source," in Proc., Ninth Int. Symp. on Communication Theory and Applications, 2007.