Technical Note. Loss-less Compression

A long time ago and in a galaxy far, far away, we once used a compression algorithm in a balloon flight to reduce detector spectra data volume for a long-duration flight. The compression was performed by an 8086 processor.

The basic scheme is to run a bit compressor over an input stream of data, converting every n-bit pattern into a resultant m-bit pattern. If the choice of n-patterns which occur often translates to a m-bit pattern of much less size, then the information volume will be reduced. However, if the normal input pattern expands during compression, then the inverse of the input pattern will reduce in size. So, the compression is performed twice, and the shorter of the two sequences is chosen. The "key" is a 1-bit value which defines which compression (positive true or inverse) was used.

Each step of the compression is as follows:
[0] Shift in 3 bits;
[1] Compute a 1 to 5 bit value using logic described
[2] Jam the output shift register
[3] Compute a 3 bit shift count, values from 1 to 5
[4] Shift the register using the shift count.

| Input i0i1i2 | Result r0-r4 | Shift s2s1s0 |
|---|---|---|
| 000 | 0 | 001 |
| 001 | 100 | 011 |
| 010 | 101 | 011 |
| 011 | 110 | 011 |
| 100 | 11100 | 101 |
| 101 | 11101 | 101 |
| 110 | 11110 | 101 |
| 111 | 11111 | 101 |

Logic:
r0 = i0 OR i1 OR i2
r1 = NOT ( i2 OR (i1 OR i0))
r2 = i0 OR (NOT i0)

Shift Length:
s0 = 1
s1 = NOT ( i2 AND NOT (i1 OR i0))
s2 = i2

The basic strategy can be extrapolated to 4-bit inputs as follows. The input-output translation can be optimized for a given type of data. In a non-adaptive Huffman, we need to make some rational decisions and measurements of the input data set to select the encoding scheme. The following table gives some example schemes.

**Huffman 4 Encoding.**

| Input | 4A | 4B | 4C | 4D |
|---|---|---|---|---|
| 0 | 0 | 00 | 0 | 0 |
| 1 | 1000 | 01 | 100 | 100 |
| 2 | 1001 | 10 | 101 | 101 |
| 3 | 1010 | 1100 | 110 | 110 |
| 4 | 1011 | 1101 | 11100 | 1110 |
| 5 | 1100 | 1110 | 11101 | 111100 |
| 6 | 1101 | 111100 | 11110 | 111101 |
| 7 | 1110 | 111101 | 1111100 | 111110 |
| 8 | 111100 | 111110 | 1111101 | 11111100 |
| 9 | 111101 | 11111100 | 1111110 | 11111101 |
| 10 | 111110 | 11111101 | 111111100 | 11111110 |
| 11 | 11111100 | 11111110 | 111111101 | 1111111100 |
| 12 | 11111101 | 1111111100 | 111111110 | 1111111101 |
| 13 | 11111110 | 1111111101 | 11111111100 | 1111111110 |
| 14 | 111111110 | 1111111110 | 11111111101 | 11111111110 |
| 15 | 111111111 | 1111111111 | 11111111110 | 11111111111 |

**Compression Analysis Using Cluster Data.**
Cluster Data was used to determine the performance and then select the best of the Huffman 4-bit options above. The input spectra of nibbles is shown in the first two columns. Each is 4 bit raw input. The Huffman 4A encodes as A.out for a total number of output bits in A.Total. Similarly, the B, C and D conversions are shown in subsequent columns.

**Ordered**. The first attempt, used a simple ordering of nibbles, 0000 thru 1111, making the shortest to longest output codes. This ordering would work best on particle count data where the data is a steep falling curve. It did not achieve terribly exciting results.

| 4-Bit Nibble | Count | IN | IN Total | A.Out | A.Total | B.Out | B.Total | C.Out | C.Total | D.Out | D.Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 44576 | 4 | 178304 | 1 | 44576 | 2 | 89152 | 1 | 44576 | 1 | 44576 |
| 1 | 7176 | 4 | 28704 | 4 | 28704 | 2 | 14352 | 3 | 21528 | 3 | 21528 |
| 2 | 7026 | 4 | 28104 | 4 | 28104 | 2 | 14052 | 3 | 21078 | 3 | 21078 |
| 3 | 181 | 4 | 724 | 4 | 724 | 4 | 724 | 3 | 543 | 3 | 543 |
| 4 | 6902 | 4 | 27608 | 4 | 27608 | 4 | 27608 | 5 | 34510 | 4 | 27608 |
| 5 | 1683 | 4 | 6732 | 4 | 6732 | 4 | 6732 | 5 | 8415 | 6 | 10098 |
| 6 | 253 | 4 | 1012 | 4 | 1012 | 6 | 1518 | 5 | 1265 | 6 | 1518 |
| 7 | 8 | 4 | 32 | 4 | 32 | 6 | 48 | 7 | 56 | 6 | 48 |
| 8 | 6752 | 4 | 27008 | 6 | 40512 | 6 | 40512 | 7 | 47264 | 8 | 54016 |
| 9 | 353 | 4 | 1412 | 6 | 2118 | 8 | 2824 | 7 | 2471 | 8 | 2824 |
| 10 | 1677 | 4 | 6708 | 6 | 10062 | 8 | 13416 | 9 | 15093 | 8 | 13416 |
| 11 | 91 | 4 | 364 | 8 | 728 | 8 | 728 | 9 | 819 | 10 | 910 |
| 12 | 256 | 4 | 1024 | 8 | 2048 | 10 | 2560 | 9 | 2304 | 10 | 2560 |
| 13 | 6 | 4 | 24 | 8 | 48 | 10 | 60 | 11 | 66 | 10 | 60 |
| 14 | 23 | 4 | 92 | 9 | 207 | 10 | 230 | 11 | 253 | 11 | 253 |
| 15 | 17 | 4 | 68 | 9 | 153 | 10 | 170 | 11 | 187 | 11 | 187 |
| | 76980 | | 307920 | | 193368 | | 214686 | | 200428 | | 201223 |
| | | | | | **1.59** | | **1.43** | | **1.54** | | **1.53** |

Ordered sequence, assuming that low values are expected.

**Bit-Wise.** In examining the data to make a second attempt, the nibble spectrum was ordered according to its frequency of occurrence and the result showed that the first five most likely codes involved mostly zero bits. Codes were 0000, 0001, 0010, 0100, and 1000, in that order. Huffman 4C works ok on the first four, but expands the 1000 code. Huffman 4D stalls the output encoding to allow 1000 to be a 4 bit output. Option B is the clear loser, no matter what data was put to it. It seems there has to be a 4-to-1 reduction to make this work.

| 4-Bit Nibble | Count | IN | IN Total | A.Out | A.Total | B.Out | B.Total | C.Out | C.Total | D.Out | D.Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 44576 | 4 | 178304 | 1 | 44576 | 2 | 89152 | 1 | 44576 | 1 | 44576 |
| 1 | 7176 | 4 | 28704 | 4 | 28704 | 2 | 14352 | 3 | 21528 | 3 | 21528 |
| 2 | 7026 | 4 | 28104 | 4 | 28104 | 2 | 14052 | 3 | 21078 | 3 | 21078 |
| 4 | 6902 | 4 | 27608 | 4 | 27608 | 4 | 27608 | 3 | 20706 | 3 | 20706 |
| 8 | 6752 | 4 | 27008 | 4 | 27008 | 4 | 27008 | 5 | 33760 | 4 | 27008 |
| 5 | 1683 | 4 | 6732 | 4 | 6732 | 4 | 6732 | 5 | 8415 | 6 | 10098 |
| 10 | 1677 | 4 | 6708 | 4 | 6708 | 6 | 10062 | 5 | 8385 | 6 | 10062 |
| 9 | 353 | 4 | 1412 | 4 | 1412 | 6 | 2118 | 7 | 2471 | 6 | 2118 |
| 12 | 256 | 4 | 1024 | 6 | 1536 | 6 | 1536 | 7 | 1792 | 8 | 2048 |
| 6 | 253 | 4 | 1012 | 6 | 1518 | 8 | 2024 | 7 | 1771 | 8 | 2024 |
| 3 | 181 | 4 | 724 | 6 | 1086 | 8 | 1448 | 9 | 1629 | 8 | 1448 |
| 11 | 91 | 4 | 364 | 8 | 728 | 8 | 728 | 9 | 819 | 10 | 910 |
| 14 | 23 | 4 | 92 | 8 | 184 | 10 | 230 | 9 | 207 | 10 | 230 |
| 15 | 17 | 4 | 68 | 8 | 136 | 10 | 170 | 11 | 187 | 10 | 170 |
| 7 | 8 | 4 | 32 | 9 | 72 | 10 | 80 | 11 | 88 | 11 | 88 |
| 13 | 6 | 4 | 24 | 9 | 54 | 10 | 60 | 11 | 66 | 11 | 66 |
| | 76980 | | 307920 | | 176166 | | 197360 | | 167478 | | 164158 |
| | | | | | **1.75** | | **1.56** | | **1.84** | | **1.88** |

Reordered based upon probability of occurrence, where 0's dominate the data.

**Conclusion**. Although the real data could be different, the logic of the 4D sequence seems pretty effective for field data. It would even work well on low valued particle data (0,1,2,4) as well.