DOI: 10.2478/ijasitels-2022-0001

In search for the simplest example that proves Huffman coding overperforms Shannon-Fano coding

Macarie BREAZU[1], *Daniel I. MORARIU*[1],
Radu G. CREȚULESCU[1], *Antoniu G. PITIC*[1],
Adrian A. BĂRGLĂZAN[1]

[1]*Computer Science and Electrical and Electronics Engineering Department, Faculty of Engineering, "Lucian Blaga" University of Sibiu, Romania*
{macarie.breazu, daniel.morariu, radu.cretulescu, antoniu.pitic, adrian.barglazan} @ulbsibiu.ro

Abstract

Shannon-Fano coding (SFC) and Huffman coding (HC) are classic and well-known algorithms, but still in use today. The search for the simplest example that proves HC overperforms SFC is still of interest. The problem is not as trivial as it looks like at first view because of several decisions that must be considered. We perform a full-search of the stream data space for a maximum stream length of 100. Depending on additional requests we impose, the simplest solution we found is {1,1,1,1,3} when we accept to select a specific cutting, {2,3,3,3,7} when we accept only deterministic (unique) cuttings and {4,5,6,7,14} when we also ask for different frequencies for symbols as well.

Keywords: Shannon-Fano, Huffman, coding, simplest example

1. Introduction

In our days, of Internet and Big Data, the need for compression is more than obvious. The data compression domain started with Claude Shannon's 1948 paper "A Mathematical Theory of Communication" [1], where he also proposed the (Shannon) entropy concept. He also proves, in his famous source coding theorem, that entropy represents an absolute mathematical limit on the performance of lossless data compression methods.

Until 1952 Shannon [1], Fano [2] and Huffman [3] proposed coding methods to (more) efficiently code symbols based on their probabilities. It was proven that the Huffman method is optimal for symbol-by-symbol coding methods [3]. But, for most simple examples, all the methods produce same results…

For binary coding (i.e., the alphabet of the code is binary) the entropy is maximized for equal (0.5-0.5) probability of code symbols, a result that was at the core of the proposed coding methods.

From the point of view of the way the statistic source model changes in time while coding (static, semistatic or dynamic), all the methods described above fall in the semistatic case: first we have to evaluate the probabilities (frequencies), then build the model (codes), and finally use the model unchanged for coding.

Although classic, the presented algorithms are still of interest, mainly as entropy coders in more advanced coding schemes ([7],[8]), but also in research ([4]).

2. Shannon-Fano coding

In [5] Krejči et. all explain: "Around 1948, both Claude E. Shannon [1] and Robert M. Fano [2] independently proposed two different source coding algorithms for an efficient description of a discrete memoryless source. Unfortunately, in spite of being different, both schemes became known under the same name Shannon–Fano coding.
There are several reasons for this mixup. For one thing, in the discussion of his coding scheme, Shannon mentions Fano's scheme and calls it "substantially the same" [1, p. 17]. For another, both Shannon's and Fano's coding schemes are similar in the sense that they both are efficient, but suboptimal prefix-free coding schemes with a similar performance."

Shannon's method starts by deciding the lengths of all the codewords based on the $\lceil -log_2 p_i \rceil$ formula, and then select prefix codes having lengths accordingly. Fano's method is based on recursively dividing the sorted set of symbols in subsets.

We will further consider, as Shannon-Fano Coding (**SFC**), the **Fano's implementation**, because it is, in our days, more popular by far and, as stated in [5], "Fano coding — while still suboptimal — usually performs slightly better than Shannon coding".

Shannon describes in [1] the method of Fano as follows: "His method is to arrange the messages of length N in order of decreasing probability. Divide this series into two groups of as nearly equal probability as possible. If the message is in the first group its first binary digit will be 0, otherwise 1. The groups are similarly divided into subsets of nearly equal probability and the particular subset determines the second binary digit. This process is continued until each subset contains only one message".

We will follow the classical implementation that follows the description above and, as usual, we will use a recursive approach for the two resulting subsets. For each prospective position for cutting (division) we compute the left sum and the right sum and select the cutting point according to the minimum absolute difference of that sums.

The approach is a **top-to-bottom** approach, the corresponding tree being built from root to leaves.

We consider the ascending order for sorting because, by using that order, we only exchange "0" and "1" in codes (performance remains the same), but the resulting

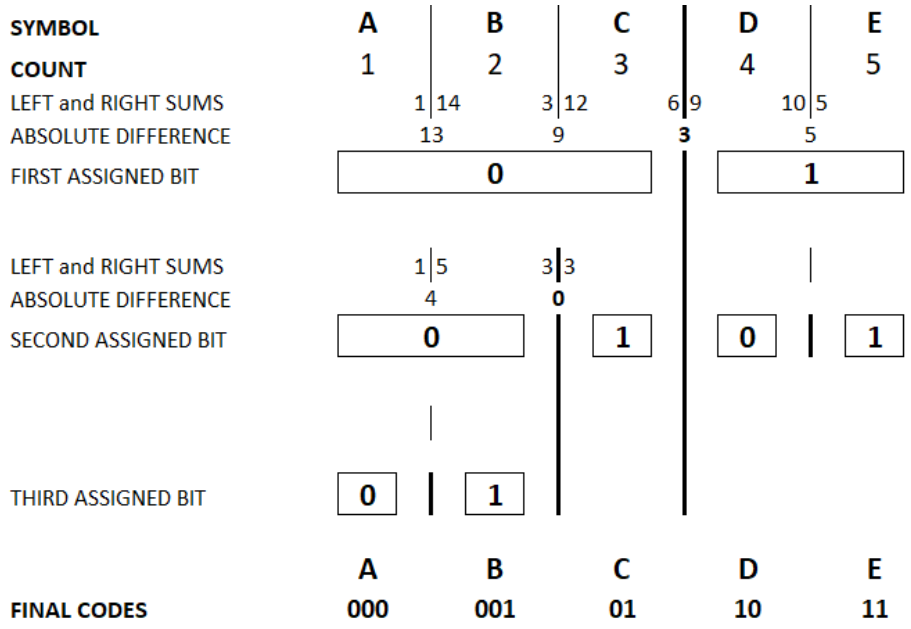


Figure 1. The {1,2,3,4,5} Shannon-Fano Coding (SFC) example

codes looks more like the ones generated by HC.

An example of applying the SFC on an input consisting of 1 A, 2 B's, 3 C's, 4 D's and 5 E's is presented in Fig. 1. The resulting codes are:

A="000", B="001", C="01", D="10" and E="11"

The total number of bits generated by coding is:

1*3+2*3+3*2+4*2+5*2 = 33 bits.

3. Huffman coding

In 1952 David A. Huffman, a student of Robert Fano, proposed "A Method for the Construction of Minimum-Redundancy Codes" [3].

Huffman coding (HC) method is briefly described as follows:

- Create a list containing nodes for each symbol (and its frequency). The list is sorted in ascending order of the frequencies.
- While the list contains more than one node:
  - Remove the first two nodes (i.e., with minimum frequencies) from the list.
  - Insert a new node having the two removed nodes as left child and right child and the frequency the sum of child frequencies.

At the end the last node from the list is the root of the coding tree. The code of each symbol is given by the path from root to leaf adding a "0" for going to the left child and "1" for going to the right child.

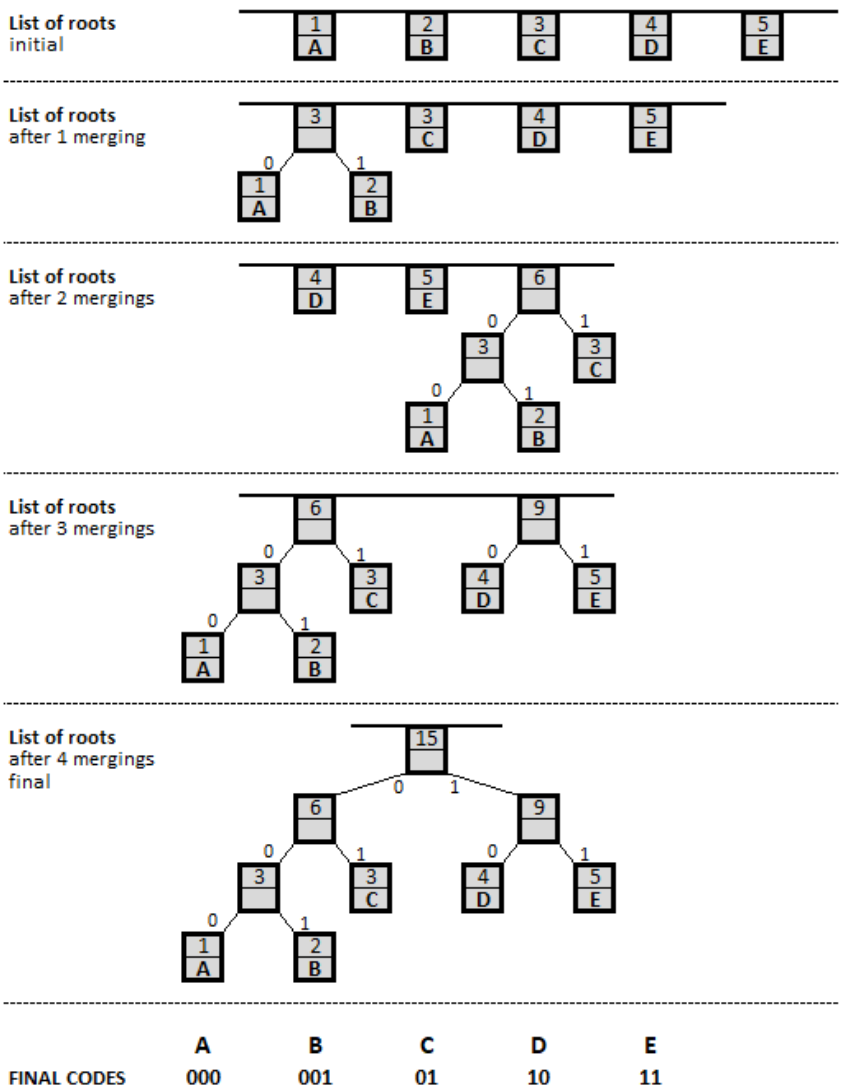


Figure 2. The {1,2,3,4,5} Huffman Coding (HC) example

The approach is a **bottom-to-top** approach, the corresponding tree being built from leaves to root.

The presented approach falls in the semistatic modelling case but is well-known (erroneously) as "Static Huffman", probably as opposite to the Dynamic Huffman method (which uses a true dynamic modelling, [6]).

An example of applying the HC on the same input consisting of 1 A, 2 B's, 3 C's, 4 D's and 5 E's is presented in Fig. 2. The resulting codes are:

A="000", B="001", C="01", D="10" and E="11" (same as SFC).

The total number of bits generated by coding is:

1*3+2*3+3*2+4*2+5*2 = 33 bits (same as SFC).

4. The research question and approach

As we can notice from the two previous examples, in simple cases SFC and HC give the same results (and performance). The general (theoretical) statement ([6]) is that HC overperforms SFC but without given examples. In [9] such an example is presented. The example presented there is {5,6,6,7,15} for the frequencies of symbols A, B, C, D, E. Technically, frequencies are presented in decreasing order, but we consider them in an increasing order, to be in line with the rest of our paper.

Therefore, a question of interest appears: "**What is the simplest example that proves that HC overperforms SFC?"**

In our approach the "**simplest**" means (in that order):

1. The **smallest number of different symbols** "**N**" and
2. The **shortest stream length "SSL"** (containing N different symbols).

Our approach was a full-search of the search space to find the answer. Certainly, we have to restrict the frequency of each symbol in a range depending on the **maximum stream length** "**MSL**" considered. Because in SFC the symbols are ordered increasingly, our main loop looks like (number of symbols not established yet):

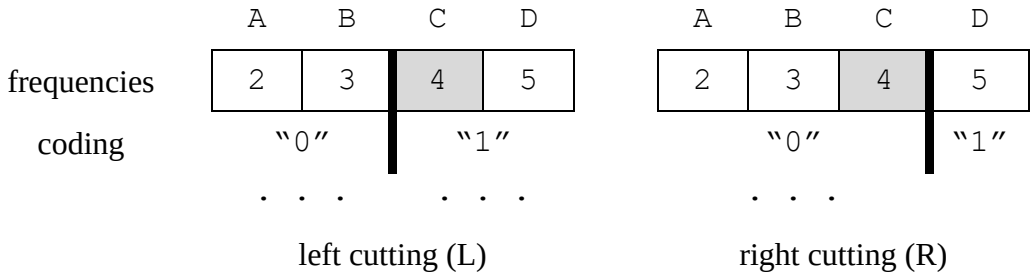
```
for(k1=1;k1<=MSL-(N-1);k1++)
  for(k2=k1;k2<=MSL-(N-2)-k1;k2++)
    for(k3=k2;k3<=MSL-(N-3)-(k1+k2);k3++)
      ...
        {
        evaluate SFC(k1, k2, k3, ...)
        evaluate HC( k1, k2, k3, ...)
        compare results
        }
```

where the symbols {A, B, C, ...} have the frequencies {k1, k2, k3, ...}.

Surprisingly, the problem is not as trivial as it looks like at first view because of more decisions that must be taken into consideration in implementation. Implementations are not fully deterministic (unique) because of the following:

- in HC, when one must choose the minimum 2 values, it is possible to have more symbols (roots) with the same frequency. Luckily, all the cases generate results that are equivalent from the compression point of view, so this is not a significant problem. But…
- in SFC, when one must divide the set in 2 subsets, it is possible to have two "division points" that give the same absolute difference. Unfortunately, the results for both cases are not always the same from the compression point of view, so they must be evaluated independently. When such a situation appears several times in a coding there are many versions of SFC that must be evaluated, to get a trustworthy result.

When we have the {2,3,4,5} (sub)set to be divided as in SFC we have two options:



left cutting (L) right cutting (R)

We label the two decisions L and R. But, as we go further with divisions (in a general case), we can have other divisions to be decided, so a final SFC case could be labeled with a string of L and R, according to the decisions that must be taken (in succession) for that specific case. More details and examples will be given in the next section.

5. Experimental results and interpretation

To determine the minimum value for N we have noticed that:

- For N=3 it is easy to prove that both SFC and HC will always obtain the same performance, codes having the length 2, 2 and 1 bits (for symbols in ascending order of frequency).
- For N=4 we obtain by simulation that there is no case where HC overperforms SFC (for MSL=100).
- For N=5 it is obvious that such a case exists (cited in section 4), so we focus on finding the **shortest stream length** for N=5.

When performing the full-search of the search space we have considered maximum stream length MSL=100.

To evaluate the result of HC and SFC we have used as criteria the total number of bits generated when coding the data stream with the codes obtained (as exemplified in chapters 2 and 3). Certainly, the average code length can be used but we consider that working with integer is simpler and also proves that the global advantage is very small (usually a single bit).

According to our approach for scanning the search space we test all possibilities in the following order:

```
{1,1,1,1,1},{1,1,1,1,2},...,{1,1,1,1,96},
{1,1,1,2,2},{1,1,1,2,3},...,{1,1,1,2,95},
{1,1,1,3,3},...,
...,
{20,20,20,20,20}
```

The total number of cases obtained is 757566. From these cases some of them must be analyzed in more variants, because of different possible cuttings for SFC. The obtained distribution is presented in Table 1.

An interesting situation are the 12 cases with 5 variants (described in Table 1). In fact, all are the same case {1,1,1,2,3} but multiplied with 1, 2, 3, . . ., 12 respectively.

Table 1. Number of variants distribution

Variants	Number of cases	Percentage	Cuttings		Number of cases with variants
With no variants	714942	94.3736%			714942
With 2 variants	41404	5.4654%	L,R		82808
With 3 variants	1130	0.1492%	L,RL,RR and LL,LR,R		3390
With 4 variants	78	0.0103%	LL,LR,RL,RR		312
With 5 variants	12	0.0016%	LL,LR,RL,RRL,RRR		60
Total	757566	100.0000%			801512

The answer to the main question ("What is the simplest example where HC overperforms SFC?") depends on what other restrictions we impose. Therefore:

- If we accept to **select a specific cutting** the simplest example is:

{1,1,1,1,3}+LL SFC=16 bits HC=15 bits

{1,1,1,1,3}+LR SFC=16 bits HC=15 bits

corresponding to a stream length of 7 symbols. Here we have also the cutting:

{1,1,1,1,3}+R SFC=15 bits HC=15 bits

where SFC and HC give the same result (Fig. 3).

The total number of examples where HC overperforms SFC is 6349, out of 801512 (0.79%).

- If we accept **only deterministic (unique) cuttings** (variants don't exist) the simplest example is:

{2,3,3,3,7} SFC=41 bits HC=40 bits

corresponding to a stream length of 18 symbols. In that case the symbols B, C and D can be interchanged, so, technically, one can still get different codes.

With the current restriction the total number of examples where HC overperforms SFC is 5884, out of 714942 (0.82%).

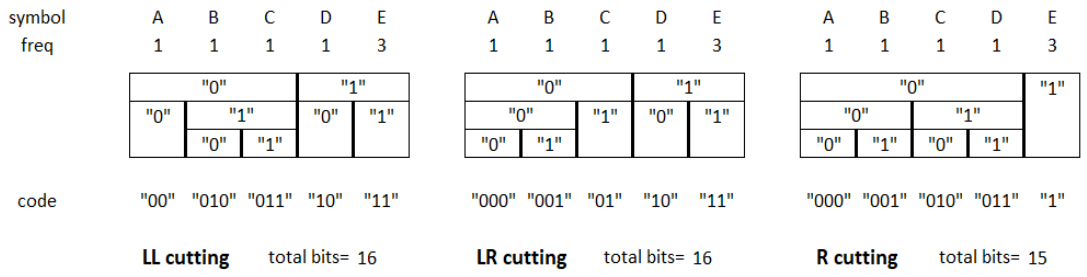


Figure 3. The {1,1,1,1,3} SFC case

- If we additionally accept **only different frequencies for symbols** (codes become technically unique) the simplest example is:

  {4,5,6,7,14} SFC=81 bits HC=80 bits

corresponding to a stream length of 36 symbols.

With the current restrictions the total number of examples where HC overperforms SFC is 1832, out of 463581 (0.39%).

6. Conclusions and future work

Even if SFC and HC are classic and well-known algorithms, the answer to the question: "What is the simplest example that proves HC overperforms SFC?" is still of interest. The problem is not as trivial as it looks like at first view because of several decisions that must be considered.

According to our experiments the simplest solution is:

`{1,1,1,1,3}`	when we accept to select a specific cutting (LL or LR).
`{2,3,3,3,7}`	when we accept only deterministic (unique) cuttings.
`{4,5,6,7,14}`	when we accept only deterministic (unique) cuttings and only different frequencies for symbols.

In our search space (N=5, MSL=100) the number of examples where HC overperforms SFC is very small (less than 1%).

Certainly, further research can be done, especially regarding:

- To prove also theoretically that for 4 symbols there is no case where HC overperforms SFC.
- To verify if, for 6 or more symbols, solutions with smaller stream length exist.

References

[1] Shannon, C. E. *A mathematical theory of communication*, The Bell System Technical Journal, Volume: 27, Issue: 3, page 379 – 423, October 1948
[2] Fano, R. M. *The transmission of information*, Research Laboratory of Electronics, Mass. Inst. of Techn. (MIT), Technical Report No. 65, Mar. 17, 1949
[3] Huffman, D. A. *A method for the construction of minimum-redundancy codes*, Proceedings of the IRE, vol. 40, no. 9, pp. 1098–1101, Sept. 1952
[4] Viraktamath, S. V., Koti, M. V., Bamagod, M. M. *Performance analysis of source coding techniques*, 2017 International Conference on Computing Methodologies and Communication (ICCMC), pp. 689-692, 2017
[5] Krajči S., Liu, C. -F., Mikeš, L., Moser, S.M. *Performance analysis of Fano coding*, 2015 IEEE International Symposium on Information Theory (ISIT), pp. 1746-1750, 2015
[6] Sayood K. *Introduction to Data Compression*, Morgan Kaufmann, ISBN 0128094745, 2017
[7] https://pkware.cachefly.net/webdocs/APPNOTE/APPNOTE-6.3.9.TXT
[8] https://en.wikipedia.org/wiki/Huffman_coding#Applications
[9] https://en.wikipedia.org/wiki/Shannon%E2%80%93Fano_coding