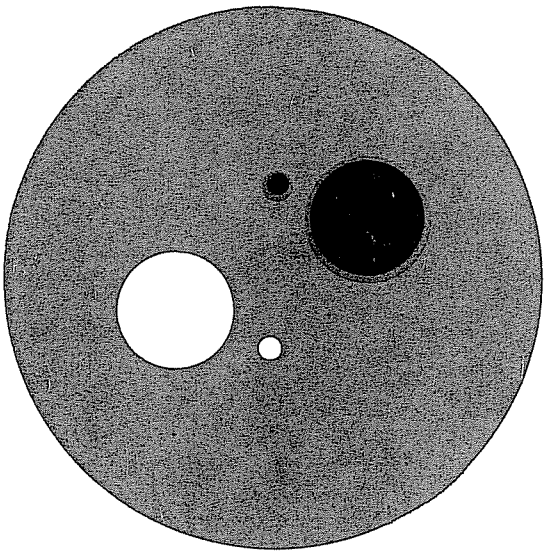
COMPUTER SCIENCES
DEPARTMENT

University of Wisconsin-
Madison



PARTITIONING FILES FOR HUFFMAN ENCODING

by

Michael L. Scott
Raphael A. Finkel

Computer Sciences Technical Report #540

April 1984

Partitioning Files for Huffman Encoding

Michael L. Scott
Raphael A. Finkel

Department of Computer Sciences
University of Wisconsin - Madison
1210 W. Dayton
Madison, WI 53706

April 1984

ABSTRACT

When the relative frequencies of characters vary significantly within a single file, it may be desirable to use a different Huffman code on different phases of the file. This note presents an heuristic algorithm for determining the boundaries between phases.

1. Introduction

Huffman encoding [3] is a well-known technique for data compression. The "standard" algorithm works off-line. Its first pass assigns each character a bit-string code whose length depends on the number of times the character appears. Common characters receive shorter codes. The second pass re-writes the data, using the new encoding.

Gallager [2] has developed an on-line adaptive algorithm. At a given point during execution, the coding to be used for the next character is the coding the off-line algorithm would have used for the portion of the data seen so far. The code used for the first few characters is less than optimal; on the other hand there is no need to prepend a translation table. We have found the on- and off-line algorithms to work equally well on almost all disk files.

The amount of compression achievable from either algorithm depends on the entropy of the data as a whole [1]. Consequently a file composed of several very different phases may not compact nearly as well as the individual phases would themselves. An example of such a file is a document in English containing large tables of numerical data.

If we could identify the phases of a file we could use a different encoding for each. Ideally we would like an on-line algorithm that determines when the local entropy of a file is significantly lower than the global entropy. We present a algorithm that, though theoretically less accurate, nonetheless agrees well with our intuitive notion of a phase and performs well in practice.

2. The Algorithm

We assign a *weight* to each byte in our file. The first occurrence of a distinct character c is assigned weight 1, the second weight 2, the third weight 3, and so on. For example

file :	a b c c a d b c e
weights :	1 1 1 2 2 1 2 3 1
total weight :	14

As we read through the file we add up the weights of all the bytes so far. We also add up the weights of the most recent n bytes. From these two totals we compute the average weight per byte so far and the average weight of the last n bytes. When the local average falls below the global

This work was supported in part by NSF grant number MCS-8105904, by Arpa contract number N0014/82/C/2087, and by a Bell Telephone Laboratories Doctoral Scholarship.

average we know that most of the recent characters are not common in the beginning of the file. We look back through the last n bytes for a maximum in the global average (it must be falling) and announce a phase change at that point. We then re-start the algorithm on the remainder of the file.

If $r_{N,c}$ is the number of occurrences of the character c in the first N bytes of the file, then the global average weight is

$$\frac{1}{N} \sum_{\substack{\text{distinct} \\ \text{characters} \\ c}} \binom{r_{N,c}+1}{2} = \frac{1}{2} + \frac{1}{2N} \sum_c r_{N,c}^2$$

and the local average weight is

$$\frac{1}{n} \sum_c \binom{r_{N,c}+1}{2} - \binom{r_{N-n,c}+1}{2} = \frac{1}{2} + \frac{1}{2n} \sum_c r_{N,c}^2 - r_{N-n,c}^2 .$$

We announce a phase change when

$$\frac{1}{2N} \sum_c r_{N,c}^2 > \frac{1}{2n} \sum_c r_{N,c}^2 - r_{N-n,c}^2$$

that is, when

$$\frac{1}{N} \sum_c r_{N,c}^2 < \frac{1}{N-n} \sum_c r_{N-n,c}^2 .$$

By way of comparison, a true entropy-measuring algorithm would announce a phase change whenever

$$\frac{1}{N} \sum_c r_{N,c} \log \frac{r_{N,c}}{N} < \frac{1}{N-n} \sum_c r_{N-n,c} \log \frac{r_{N-n,c}}{N-n} .$$

3. Practical Considerations

The "standard" off-line Huffman algorithm is not efficient for small amounts of data, due to the size of the prepended translation table. By using the adaptive algorithm on individual phases we can minimize the size of the smallest phase worth keeping. We prefix each phase of the compacted file with a simple count of the number of bytes it contains.

We can tune the partitioning algorithm by varying n. We can also vary the number of bytes passed over in the beginning of each phase to allow the averages to stabilize. We have obtained good results by requiring the global average to involve at least $2n$ bytes before comparing it to the local average. Allowing fewer than $2n$ bytes tends to result in a large number of very small phases. Requiring more than $2n$ bytes does not seem to change the behavior of the algorithm significantly.

4. Experimental Results

We have implemented the partition-compact algorithm under Berkeley UNIX 4.2* on a VAX. Our program spends less than 20% of its time discovering phases and more than 80% of its time compacting them.

---

* UNIX is a trademark of Bell Telephone Laboratories

It is easy to create files on which the program eliminates 10% more of the original length of the file than does the standard algorithm. Results are also impressive for most executable files, with additional compactions ranging between 5% and 10%.

When n is small (say 30 bytes) it is easy to find files on which the partition-compact program performs significantly worse than the standard algorithm. When n is as large as 512 such pathological behavior disappears.

Pascal programs show no significant improvement, nor do English documents. We have noticed 2% or 3% additional compaction for technical papers containing a moderate number of numerical tables.

5. Conclusion

Given that no one really wants to compress executable files, and given the modest gains for most other file types, we would recommend multi-phase compacting for unusual applications only – ones in which the relative frequencies of characters vary significantly over time. For such applications the partitioning algorithm presented above is a simple, effective, and efficient means of identifying phases.

Acknowledgment

Our interest in Huffman encoding originated in Sam Bent's seminar on trees.

References

[1] Gallager, R. G., *Information Theory and Reliable Communication*, Wiley, New York (1968).

[2] Gallager, R. G., "Variations on a Theme of Huffman," *IEEE Transactions on Information Theory IT-24*, 6, pp. 668-674 (November 1978).

[3] Huffman, D. A., "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings of the IRE 40*, 9, pp. 1098-1101 (September 1952).