On the Hardness of Finding Optimal Multiple Preset Dictionaries

Michael Mitzenmacher *
Harvard University

Abstract

Preset dictionaries for Huffman codes are used effectively in fax transmission and JPEG encoding. A natural extension is to allow multiple preset dictionaries instead of just one. We show, however, that finding optimal multiple preset dictionaries for Huffman and LZ77-based compression schemes is NP-hard.

1 Introduction

Preset dictionaries are often used to improve compression. For example, with standard two-pass Huffman coding, one generally sends a table describing the encoding, or a *dictionary*, that allows the decoder to determine the appropriate code words for each alphabet symbol. Instead, if similar transmissions occur on a repeated basis, a preset dictionary can be set in advance to avoid the cost of computing and transmitting an explicit dictionary each time. Avoiding memory and computation costs for dictionary computation may be useful even if it yields slighltly worse compression. Preset dictionaries may also yield improved compression results when the cost of sending an explicit dictionary would be more than the gain the explicit dictionary would yield over the preset dictionary. This situation may occur when documents are short and a suitably effective preset dictionary can be found. Preset dictionaries arise in for example fax transmission and JPEG encoding [4].

*Harvard University, Division of Engineering and Applied Sciences, 33 Oxford St., Cambridge, MA 02138. Supported in part by an Alfred P. Sloan Research Fellowship, NSF CAREER grant CCR-9983832, and an equipment grant from Compaq Computer Corporation. E-mail: michaelm@eecs.harvard.edu.

Proceedings of the Data Compression Conference (DCC'01)
1068-0314/01 $10.00 © 2001 IEEE

A natural extension to this idea is to allow multiple preset dictionaries. Flag bits at the beginning of a file can be used to denote which (if any) preset dictionary to use. Allowing multiple dictionaries intuitively should improve compression by providing more flexibility. Such an idea is quite natural; indeed, the ZLIB library, designed for LZ77-based compression, allows for multiple preset dictionaries [1]. The tradeoff is that more space is required to store the preset dictionaries, and more computation is required to test which dictionary should be used for compression. Note that this additional computation is required only at the compression end, and is easily parallelized.

In this paper, we relate the problem of finding optimal multiple preset dictionaries to the model of *segmentation problems* introduced in [3]. This connection between a simple compression problem and a natural economics problem may be interesting in its own right. In the spirit of these results, we refer to problems related to finding multiple preset dictionaries as *compression segmentation problems* Using this connection, we show that natural compression segmentation problems for Huffman trees and LZ77-based compression are NP-hard.

2 The catalog segmentation problem

The problem of finding optimal families of preset dictionaries is related to the segmentation problems defined by Kleinberg, Papadimitriou, and Raghavan. The canonical segmentation problem is the *catalog segmentation* problem, which we first describe informally. A seller can send a catalog to all customers in its database. Only r items can be advertised in a catalog. Given previous history, the seller can exactly tell which people will buy which items. The goal is to maximize the number of sales. If the seller could create just one catalog, the optimal solution would be to include the r most popular items. Suppose instead the seller can create k different catalogs and send exactly one to each customer. How should the seller determine the k catalogs that will maximize the number of sales?

Following [3], we formally define the catalog segmentation problem as follows. Consider the customers as sets of items $S_1, S_2, \ldots, S_n$ over a ground set U. Catalogs $X_1, X_2, \ldots, X_k$ are also sets of items. The goal is to choose the X_i such that $|X_i| \leq r$ for all i and

$$\sum_{j=1}^{n} \max_{1 \leq i \leq k} (|X_i \cap S_j|)$$

is maximized.

Theorem 1 [3] *The catalog segmentation problem is NP-hard (even for $k = 2$).*

Even though the catalog segmentation problem is NP-hard, it can be solved in polynomial time for any fixed r and k, since there are only $\binom{|U|}{r}$ possible catalogs.

Although in [3] the authors say that the catalog segmentation problem (and several natural variants) are NP-hard, complete proofs are not given. For completeness we offer our own simple proof of Theorem 1, suggested to us by Steve Lumetta, below. We then reduce the catalog segmentation problem to the problems of finding optimal multiple preset dictionaries for Huffman coding and Lempel-Ziv coding, thereby showing that these problems are NP-hard. For convenience for the remainder of the paper we focus on the case where $k = 2$, although our results are easily generalized to other values of k.

Theorem 2 *The catalog segmentation problem is NP-hard for $k = 2$.*

Proof: We reduce from the well-known NP-hard problem Graph Bisection [2]: given a graph $G = (V, E)$ with an even number of vertices, split V into two disjoint sets V_1 and V_2 with $|V_1| = |V_2| = |V|/2$ such that the number of edges adjacent to both V_1 and V_2 is minimized. We turn an instance of simple graph bisection into a catalog segmentation problem as follows. For each vertex, create a corresponding item. If d is the maximum degree of the graph, create for each item $d + 1$ customers who want to purchase only that item. For each edge, create a customer that wants to purchase only those two items corresponding to the vertices adjacent to that edge. Now suppose we can have $r = |V|/2$ items in each catalog. It it easy to see that the optimal pair of catalogs must contain all $|V|$ items. Otherwise, some item appears in both catalogs, but since the maximum degree of the graph is d replacing one copy of the repeated item by some item that does not appear improves the number of items sold. Because the optimal pair of catalogs contains all $|V|$ items, we may conclude that it also provides a bisection that minimizes the number of edges crossing from V_1 to V_2. This completes the reduction. □

3 Huffman coding

We now define the *Huffman code segmentation problem.* We are given a collection of documents $D_1, D_2, \ldots, D_n$ over an alphabet Σ. Finding an optimal sequence of Huffman code word lengths over Σ to compress these documents is trivial; it simply requires summing the character frequencies over all of the documents and using the standard Huffman tree algorithm. Suppose, however, we were allowed to construct k different Huffman codes, and use the best one to compress each document. The Huffman code segmentation problem is to minimize the total compressed size given the D_i and $k \geq 2$.

To see how the Huffman code segmentation problem might naturally arise, suppose we plan to design multiple preset Huffman codes for a large, arbitrary collection of documents, such as all Web pages. We might then sample n representative pages as a test set in order to develop our Huffman codes, which will be used over the larger class of documents. The Huffman code segmentation problem designs the k best codes for this test set.

Theorem 3 *The Huffman code segmentation problem is NP-hard.*

Proof: We reduce from catalog segmentation for the case $k = 2$. Recall for the catalog segmentation problem we have a ground set U with $|U| = m$ and n subsets $S_1, \ldots, S_n$ of U. We wish to find two subsets X and Y of U with size r such that

$$\sum_{j=1}^{n} \max(|X \cap S_j|, |Y \cap S_j|)$$

is maximized. We will design a related Huffman code segmentation problem so that each element in the ground set corresponds to a character of Σ, and each character has depth d or $d+1$ for some d in the pair of optimal Huffman trees. The sets X and Y will correspond to the characters of depth d derived from elements of U in each Huffman tree.

More specifically, let d be the smallest integer such that $2^{d+1} \geq m + r$. Our alphabet Σ will consist of $2^{d+1} - r$ characters. The first m characters, $u_1, u_2, \ldots, u_m$ represent characters that correspond to elements of U. We also introduce additional characters $v_1, v_2, \ldots, v_h$, where $h = 2^{d+1} - r - m$, so that there are $2^{d+1} - r$ total characters.

For each set S_j we construct a corresponding document D_j. Let z be a sufficiently large constant (such as 8). The document D_j will contain z occurrences of each character u_q such that item q is contained in S_j, and $z - 1$ occurrences of every other character.

With this construction, we may assume without loss of generality that all of the characters $v_1, v_2, \ldots, v_h$ should have depth at least as large as any character u_i in both of the Huffman trees in the solution, because their frequency is at least as large in every document. Similarly, if the depths of

Proceedings of the Data Compression Conference (DCC'01)
1068-0314/01 $10.00 © 2001 IEEE

all characters in both trees are not within one of each other, the total cost can be improved by flattening the offending tree. That is, if some node has depth a and two other nodes have depth (at least) $a+2$, we may improve the tree by replacing it with one where all three nodes have depth $a+1$. This reduces the compression cost by at least $2(z-1)n - zn > 0$.

Hence there must be exactly r characters from U with depth d in each of the two trees of the solution, and all other characters have depth $d+1$. We show that the sets of r characters with depth d in the two trees yield the sets X and Y for the catalog segmentation problem, by replacing characters with the corresponding elements. The cost of compressing D_j using optimal pair of Huffman trees is the sum of the following terms: $(z-1)(d+1)h$ for characters $v_1, v_2, \ldots, v_h$; $zd(\max(|X \cap S_j|, |Y \cap S_j|)$ for characters in S_j of depth d in the better tree; and $z(d+1)(m - \max(|X \cap S_j|, |Y \cap S_j|)$ for other characters u_i. Hence the total compression cost over the n documents is

$$n(z-1)(d+1)h + nz(d+1)m - z\sum_{j=1}^{n} \max(|X \cap S_j|, |Y \cap S_j|).$$

Minimizing the compression is therefore equivalent to maximizing the result of the catalog segmentation problem.

Also, the corresponding decision version, which asks if there is a pair of trees that compresses the documents down to t total bits, is clearly NP-complete. □

We note an obvious approximation result is that using one Huffman tree is at most $\lceil \log_2 k \rceil$ bits per character worse than using k Huffman trees, since we could clearly combine the k separate trees into a single super-tree. In other words, given the optimal Huffman trees for a given k, we could design a compression scheme where the first $\lceil \log_2 k \rceil$ bits would specify which of the k trees to use, and then use the appropriate codeword from that tree; the optimal single Huffman tree performs better than this solution. Proving better approximation results remains open question.

4 Preset dictionaries for Deflate

The ZLIB format was primarily designed for use with the DEFLATE procedure, an LZ77-based algorithm [1]. Since the LZ77 format is standard and described fully in most basic compression texts (e.g., [4]), we rely on an informal description here. As a document is sequentially compressed (or decompressed), there is a window into the previous stream of characters. The

Proceedings of the Data Compression Conference (DCC'01)
1068-0314/01 $10.00 © 2001 IEEE

current sequence of characters can be compressed by providing a pointer into the window of the previous character stream and a length denoting how many characters starting from that pointer are the same as the current stream. The decompressor can use these pointers to efficiently reconstruct the original text. In this setting, a preset dictionary consists of a sequence of characters that the compressor and decompressor use as an implicit prefix to the stream to be compressed. As an example, we might expect most Web pages to include the character string "http://www". Including this string in a preset dictionary may therefore improve compression. We note that finding even a single optimal preset dictionary for a given set of documents is non-trivial, and we do not currently know a solution. There are unusual subtleties, including how the position of the character sequence in the dictionary affects the amount of compression and possible overlaps of words. A natural approach for English text, however, is to find the most frequently used words and use them as the basis for a dictionary.

The *LZ77 segmentation problem* is to determine given $k \geq 2$ and a set of documents $D_1, D_2, \ldots, D_n$ over an alphabet Σ the k best preset dictionaries of size at most s, where the cost of compressing D_i is taken to be the minimum number of bits over the choice of the k dictionaries. When $k \geq 2$, the problem is NP-hard.

Theorem 4 *The LZ77 segmentation problem is NP-hard.*

Proof:

We again reduce from the catalog segmentation problem for $k = 2$. The main problem is to avoid complications introduced by string position and strings sharing characters (overlapping), so the corresponding compression problem matches the segmentation problem.

Given a catalog segmentation problem, we construct an LZ77 segmentation problem whose alphabet Σ has size $(z+1)|U|$ for a value of z to be determined. For each u_i in the ground set $|U|$ we associate $z+1$ distinct characters from Σ so that the characters associated with each u_i are disjoint. Let us consider a specific u_i with associated characters $w_0, w_1, \ldots, w_z$. We associate a string with u_i of length $3z$ of the form $(w_0)^z w_1 w_2 \ldots w_z (w_0)^z$. That is, with u_i we associate a string consisting of z occurrences of a *boundary character* w_0, followed by the *base string* of z other characters associated with u_i, followed again by z occurrences of the boundary character w_0. For each set S_j of the catalog segmentation problem, there is an associated document D_j constructed by concatenating all the strings associated with the elements of S_j. We seek dictionaries with size rz. Note that as each string corresponding to a u_i consists of distinct letters we avoid the problem of overlapping strings discussed above for the case of one dictionary.

It is not too hard to see that the the optimal preset dictionaries consist of concatenated strings of length z, with each such string corresponding to the middle third of a string corresponding to some u_i. Note first that no boundary character should be included in the preset dictionaries, as strings of consecutive boundary characters are easy to compress. (Indeed, the string of z successive boundary characters requires only $O(\log|\Sigma| + \log z)$ bits; the first terms represents the cost of denoting the first appearance of the character, the second represents the cost of describing the length of the subsequent match.) Also, a preset dictionary should not contain substrings of base strings of size strictly less than z. Any such dictionary could be improved by replacing a subblock containing two or more base strings with a single base string, choosing the base string of the most frequent u_i with characters in the subblock for the documents using that preset dictionary.

Also, the value of z can be chosen sufficiently large (but still polynomial in the input size) so that the ordering of the strings in the preset dictionaries and the documents D_j has a lower order effect. Hence we can effectively ignore ordering, and focus instead on how many length z strings each document matches with each dictionary. This is because a failure to match a length z string corresponding to some u_i will cost $O(z \log|\Sigma|)$ bits to write out the uncompressed characters, whereas a successful match will require $O(\log rz)$ bits for the relevant pointers describing the location of the match and the length of the match. The number of matches is therefore the dominant term in the compressed size.

Hence, with these conditions, the compression gain for each document is proportional (up to lower order terms) to the number of strings in the document that are matched in the dictionary. The optimal solution to the LZ77 segmentation problem therefore naturally yields a corresponding optimal solution to the catalog segmentation problem. Each dictionary maps to a catalog by mapping length z strings of the same character in the dictionaries to items in the catalogs. $\square$

5 Conclusions

Preset dictionaries have proven useful for various compression schemes, including JPEG and fax transmission. Using multiple preset dictionaries offers the potential for improved compression, and hence one might hope that optimal multiple preset dictionaries could easily be found. We have instead shown that the problem is NP-hard by showing a reduction to a simple and useful NP-hard problem, catalog segmentation.

In practice, approximations would clearly be suitable. Heuristic techniques for the catalog segmentation problem as discussed in [3] could easily be applied. Provable approximations remain an interesting open problem.

References

[1] P. Deutsch, J-L. Gailly. ZLIB Compressed Data Format Specification Version 3.3. Network Working Group RFC 1950, 1996.

[2] M. Gary, D. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237-267, 1976.

[3] J. Kleinberg, C. Papadimitriou, P. Raghavan. Segmentation problems: A micro-economic view of data mining. In *Proc. 30th ACM Symposium on Theory of Computing*, pages 473-482, 1998.

[4] I. Witten, A. Moffat, T. Bell. **Managing Gigabytes: 2nd Edition**. Morgan Kaufmann, San Francisco, 1999.

1068-0314/01 $10.00 © 2001 **IEEE**