HKUST Theoretical Computer Science Center Research Report HKUST-TCSC-2002-02

Huffman Coding with Unequal Letter Costs

[Extended Abstract]

Mordecai J. Golin
Hong Kong UST Clear Water Bay Kowloon, Hong Kong
golin@cs.ust.hk

Claire Kenyon
Laboratoire de Recherche en Informatique (LRI) Université Paris-Sud France
kenyon@lri.fr

Neal E. Young
Akamai Technologies Cambridge, MA USA
neal@young.name

ABSTRACT

In the standard Huffman coding problem, one is given a set of words and for each word a positive frequency. The goal is to encode each word w as a codeword $c(w)$ over a given alphabet. The encoding must be prefix free (no codeword is a prefix of any other) and should minimize the weighted average codeword size $\sum_w \text{freq}(w)\,|c(w)|$. The problem has a well-known polynomial-time algorithm due to Huffman [15].

Here we consider the generalization in which the letters of the encoding alphabet may have non-uniform lengths. The goal is to minimize the weighted average codeword length $\sum_w \text{freq}(w)\,\text{cost}(c(w))$, where $\text{cost}(s)$ is the sum of the (possibly non-uniform) lengths of the letters in s. Despite much previous work, the problem is not known to be NP-hard, nor was it previously known to have a polynomial-time approximation algorithm. Here we describe a polynomial-time approximation scheme (PTAS) for the problem.

Categories and Subject Descriptors

E.4 [**Data**]: Coding And Information Theory—*Data compaction and compression*; F.2 [**Theory Of Computation**]: Analysis Of Algorithms And Problem Complexity

General Terms

Algorithms

1. INTRODUCTION

Given a set of W of n words with associated probabilities or frequencies $p_1 \geq p_2 \geq \cdots \geq p_n > 0$ and an *encoding alphabet* Σ, the *prefix coding* problem, sometimes known as the *Huffman encoding* problem is to find a prefix-free code over Σ of minimum cost. This problem is very well studied and has a well-known $O(n)$-time greedy algorithm due to Huffman [15] ($O(n \log n)$ if the p_i are not sorted in advance).

Here we consider the generalization of the problem in which the letters used for encoding can have different costs. That is, letting $r = |\Sigma|$, the r letters have associated costs $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_r$ and the cost of a codeword is defined to be the sum of the costs of its letters (rather than the length of the codeword).

For an example of the problem we are addressing refer to Figure 1. Both codes have minimum cost for the frequencies $(p_1, p_2, p_3, p_4) = (2, 2, 1, 1)$ but under different letter costs. The code $\{00, 01, 10, 11\}$ has minimum cost for the standard Huffman case of $\Sigma = \{0, 1\}$ in which the length of each letter is 1, i.e., the cost of a word is the number of bits it has. The code $\{aaa, aab, ab, b\}$ has minimum cost for the alphabet $\Sigma = \{a, b\}$ in which the length of a "a" is 1 and the length of a "b" is 3.

This generalization is motivated by coding problems in which different characters have different transmission times or storage costs [5; 22; 19; 28; 29]. One example is the telegraph channel [10; 11] in which $\Sigma = \{\cdot, -\}$ and $\ell_2 = 2\ell_1$, i.e., in which dashes are twice as long as dots. Another is the (a, b) run-length-limited codes used in magnetic and optical storage [16; 12], in which the codewords are binary and constrained so that each **1** must be preceded by at least a, and at most b, **0**'s. (This example can be modeled by the problem studied here by using an encoding alphabet of $r = b - a + 1$ characters $\{0^k 1 : k = a, a+1, \ldots, b\}$ with associated costs $\{\ell_i = a + i - 1\}$.)

The literature contains many algorithms for the generalized problem. The special case when all the *probabilities* are equal (but not the letter lengths), known as the Varn coding problem, is solvable in polynomial-time [29; 1; 7; 25; 13; 6]. For the generalized problem, Blachman [5], Marcus [22], and (much later) Gilbert [11] give heuristic constructions. Karp gave the first algorithm yielding an exact solution (assuming the letter costs are integers); Karp's algorithm transforms the problem into an integer program and does not run in polynomial time [19]. Karp's result was followed by many others [21; 9; 8; 23; 3] presenting solutions of cost at most $\text{OPT} + f(\ell_1, \ell_2, \ldots, \ell_r)$ where OPT is the cost of the optimal code and $f(\ell_1, \ell_2, \ldots, \ell_r)$ is some fixed function of the edge costs, with the different algorithms having different $f(\cdot)$ (note that for these results the p_i are considered to be probabilities that sum to 1). These algorithms essentially demonstrate that a generalized version of entropy is a lower bound on the code cost and then design algorithms that

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for pro£t or commercial advantage and that copies bear this notice and the full citation on the £rst page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior speci£c permission and/or a fee.
STOC'02, May 19-21, 2002, Montreal, Quebec, Canada.
Copyright 2002 ACM 1-58113-495-9/02/0005 ..$5.00

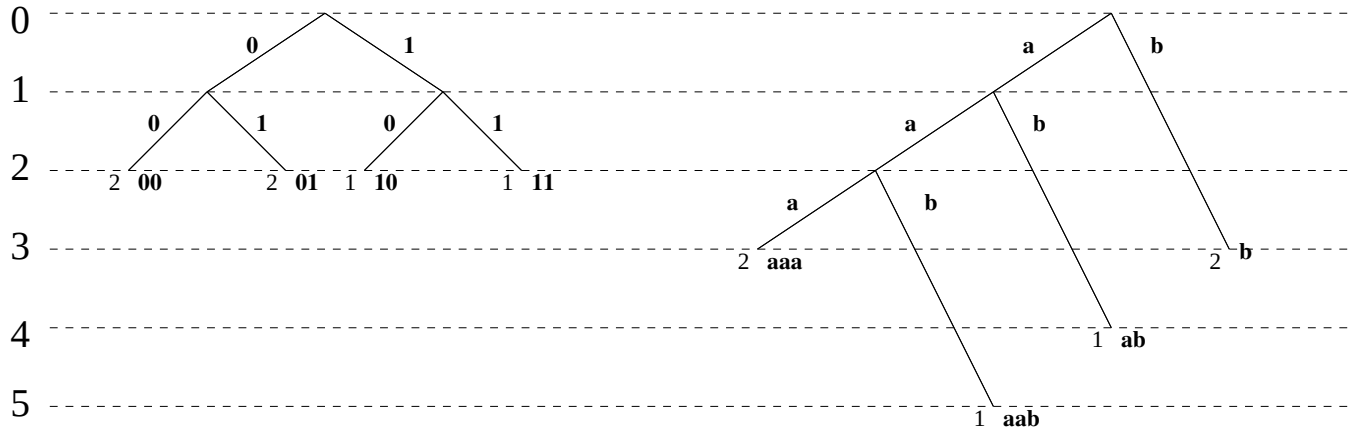

Figure 1: Two minimum-cost codes for the frequencies $(p_1, p_2, p_3, p_4) = (2, 2, 1, 1)$ but under different alphabets. Codes are represented using the standard tree format with leaves signifying codewords. The left tree is for $\Sigma = \{0, 1\}$ with $\ell_1 = cost(0) = 1$ and $\ell_2 = cost(1) = 1$. The right tree is for $\Sigma = \{a, b\}$ with $\ell_1 = cost(a) = 1$ and $\ell_2 = cost(b) = 3$. The four codewords encoded by the left tree are $\{00, 01, 10, 11\}$; the cost of the code is $2 \cdot 2 + 2 \cdot 2 + 1 \cdot 2 + 1 \cdot 2 = 12$. The four codewords encoded by the right tree are $\{aaa, aab, ab, b\}$; the cost of the code is $2 \cdot 3 + 2 \cdot 3 + 1 \cdot 4 + 1 \cdot 5 = 21$.

construct codes within an additive function of that entropy. They do not imply imply a PTAS for the problem, though, even with fixed letter costs. Golin and Rote [12] gave a dynamic programming algorithm that produces exact solutions in $O(n^{\ell_r+2})$ time for the special case when the ℓ_i are restricted to be integers. Bradford et. al. [24] improved this to $O(n^{\ell_r})$ when $r = 2$.

For further references on Huffman coding with unequal letter costs, see Abrahams' recent survey on source coding [2, Section 2.7], which contains a section on the problem.

Despite the extensive literature, there is no known polynomial-time algorithm for the generalized problem, nor is the problem known to be NP-hard. Before this work, the problem was not known to have any polynomial-time approximation algorithm. Our main result here is a polynomial-time approximation scheme (PTAS) for the problem:

THEOREM 1. *Given an instance $((p_i), (\ell_j))$ of the Huffman coding problem with unequal letter costs, and given a positive ϵ, there exists an algorithm that constructs a prefix code of cost at most $(1 + \varepsilon)$OPT; this algorithm runs in time $nd \log(n) \exp(O(\ln(1/\epsilon)^2/\epsilon^2))$, where d is the number of distinct letter costs.*

The algorithm is based on a new relaxation of Huffman coding with unequal letter costs. The relaxation, called the *k-prefix code problem*, allows codewords of cost more than k to be prefixes of other codewords. The algorithm uses grouping and enumeration techniques to find a near-minimum-cost k-prefix code (where k is a constant depending only on ϵ), and then converts this k-prefix code into a true prefix code increasing the cost by at most a $1 + O(\epsilon)$ factor.

The techniques introduced in this paper can also be used to construct PTAS's for the *regular-language prefix-coding* problem, a different generalization of Huffman coding that asks for a minimum-cost prefix code under the additional restriction that all codewords belong to a given regular language $\mathcal{L}$. As one example, the binary codes, constrained so all codewords must end in a **1**, are used for group testing and the construction of self-synchronizing codes [4; 26]. As another example, binary codes whose codewords contain at most a specified number of **1**'s are used for energy minimization of transmissions in mobile environments [27]. Algorithms (other than exhaustive search) for the regular-language prefix-coding problem generalize [12] and run in time $n^{\Theta(S(\mathcal{L}))}$ where $S(\mathcal{L})$ is the number of states in the smallest deterministic finite automaton that accepts $\mathcal{L}$. In this extended abstract we do not discuss how the techniques here extend to that problem.

Alphabetic coding is like the generalized problem considered here, but with an additional constraint on the code: the codewords must be chosen in increasing alphabetic order (with respect to the words to be encoded). This problem arises in designing testing procedures in which the time required by a test depends upon the outcome of the test [20, 6.2.2, ex. 33] and has also been studied under the names *dichotomous search* [14] or the *leaky shower* problem [18]. Alphabetic coding has a polynomial-time algorithm [17].

2. NOTATIONS AND DEFINITIONS

A problem instance is specified by a set W of n words with associated frequencies $p_1 \geq p_2 \geq \cdots \geq p_n > 0$, an alphabet Σ of $r \geq 2$ letters with associated costs $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_r$, and an $\epsilon > 0$. The *Huffman coding problem with unequal letter costs* is the problem of finding a prefix code of minimum cost.

DEFINITIONS 1. *A* **code** *(usually denoted c) is an injective map from W to Σ^*. Its image $c(W)$ is called the set of* **codewords of** *c. A set $S \subset \Sigma^*$ is* **prefix-free** *if no element of S is a prefix of any other element of S; a* **prefix code** *is one whose set of codewords is prefix-free. The* **cost** *of a*

code c is $\sum_{i=1}^{n} p_i \, cost(c(w_i))$, where cost$(x)$ is the sum of the costs of the letters of x.

Unless otherwise stated, all the codes considered here are **ordered**, i.e. the map c assigns codewords of smaller costs to words of larger probability: $\text{cost}(c(w_1)) \leq \text{cost}(c(w_2)) \leq \cdots \leq \text{cost}(c(w_n))$. Clearly, any optimal code must be ordered.

DEFINITION 2. *A* **k-prefix code** *is a code in which no codeword of cost less than k is a prefix of any other codeword.*

One of the main reasons that prefix-codes are useful is that they are uniquely decipherable. A k-prefix code is in general not uniquely decipherable and therefore not particularly useful by itself. The problem of designing a k-prefix code is introduced and used here solely as an intermediate tool for solving the original problem.

We generally use w to denote a word to be encoded, x to denote a potential codeword (a string in Σ^*), cost(x) to denote the sum of the costs of the letters in x's, and $|x|$ (the **size** of x) to denote the number of letters. To distinguish the given words W from the potential codewords Σ^*, we call the former **words** and the latter **strings**. We use d to denote the number of distinct letter costs.

The following assumption about the input is convenient:

ASSUMPTION 1. *Each ℓ_i for $i \geq 2$ is an integer multiple of ϵ, ℓ_2 equals 1, and ϵ is either an integer multiple of ℓ_1 or evenly divides ℓ_1, Thus, all codeword costs are integer multiples of* $\min\{\ell_1, \epsilon\}$.

It is also without loss of generality:

LEMMA 1. *The following reduction reduces any problem instance to an instance satisfying Assumption 1:*
1. Scale the ℓ_i's uniformly to make $\ell_2 = 1$.
2. Decrease ϵ to its next smallest multiple, or divisor, of ℓ_1.
3. Increase each ℓ_i for $i \geq 2$ to its next largest multiple of ϵ.
4. Scale the ℓ_i's, and ϵ, uniformly to make $\ell_2 = 1$.

PROOF. Step 2 decreases ϵ by at most a factor of 2. Step 3 increases the cost of any solution by at most a $1+\epsilon$ factor. Step 4 decreases ϵ by at most a factor of $1+\epsilon$. □

In the main sections (particularly Lemmas 7 and 8) we also assume that $\ell_1 > \epsilon/n$. The special case $\ell_1 \leq \epsilon/n$ is easy and is dealt with in Section 7.

Before we explain the main algorithm (Algorithm 3), we first explain two subroutines used in that algorithm: Algorithm 1 for constructing a so-called *leveled* k-prefix code meeting certain constraints, and Algorithm 2 for converting such a code into a true prefix code.

Algorithm 1 – builds a leveled k-prefix-free set of codewords given the level 0 codeword and the number of codewords per level.

INPUT: Letter costs, directed graph D, constraints $(f(0), f(1), \ldots, f((k-1)/\epsilon))$.

OUTPUT: leveled k-prefix code with $f(i)$ codewords in each level $i \geq 1$ and a codeword in level zero of size $f(0)$ (if $f(0) > 0$). (Or "inconsistent" if no such code exists.)

1: $S \leftarrow \emptyset$
2: For any node ℓ of D, define $v_S(\ell)$ to be the number of strings of cost ℓ having no prefix in S. (The algorithm will build a table v with $v[\ell] = v_S(\ell)$ as it proceeds.)
3: If $f(0) > 0$, then $S \leftarrow \{a^{f(0)}\}$ (a is the smallest letter).
4: For each node ℓ on level 0, initialize $v[\ell]$: $v[\ell] = 1$ if $\ell < f(0)$ or $f(0) = 0$; $v[\ell] = 0$ otherwise.
5: **for** $i = 1, 2, \ldots, (k-1)/\epsilon$ **do**
6: For each node ℓ of D in level i, by order of increasing costs, initialize $v[\ell]$ using the recurrence $v_S(\ell) = \sum_{i=1}^{r} v_S(\ell - \ell_i)$.
7: Let $\ell = 1 + i\epsilon - \min\{\ell_1, \epsilon\}$. If $v[\ell] < f(i)$, then return "inconsistent". Otherwise, choose $f(i)$ codewords of cost ℓ and add them to S. Decrement $v[\ell]$ by $f(i)$.
8: Complete S by adding $n - |S|$ strings of minimum cost among the strings of cost $\geq k$ that don't have a prefix of cost $< k$ in S. If $|S| < n$ and there aren't any such strings, return "inconsistent". (Find the $n - |S|$ strings by extending D as needed beyond cost k.)
9: Return the set of codewords S.

3. FINDING AN OPTIMAL K-PREFIX CODE

Algorithm 1 finds an optimal k-prefix code meeting some given constraints. The first constraint is that the code should be *leveled*:

DEFINITIONS 3. *For $i \in \{1, \ldots, (k-1)/\epsilon\}$, define the ith* **level of** Σ^* *to be the set of strings x such that $1 + (i-1)\epsilon \leq$ cost$(x) < 1 + i\epsilon$. Define level 0 to be the set of strings that cost less than 1.*

A code is **maximal within level** *i if every codeword in level i is of cost $1 + i\epsilon - \min\{\ell_1, \epsilon\}$.*

A code is **leveled** *if it is maximal within all levels $i > 0$.*

Note that level 0 can only contain words of a^*, where a is the letter of cost ℓ_1. Note also that by assumption either (i) ℓ_1 is an integer multiple of ϵ, in which case each level $i > 0$ consists of the strings of cost $1 + (i-1)\epsilon$ and every code is trivially maximal, or (ii) ℓ_1 evenly divides ϵ, in which case every string in level i has cost $1 + (i-1)\epsilon + j\ell_1$ for some integer $j \leq n$.

The second constraint that the constructed codeword must meet is specified by a tuple $f = (f(0), \cdots, f((k-1)/\epsilon))$ of integers, with the following meaning.

1. If $f(0) = 0$, then there must be no codeword in level 0; if $f(0) > 0$, then the set of codewords must contain the codeword $a^{f(0)}$ from level 0.

Algorithm 2 — converts a k-prefix code into a prefix code

INPUT: letter costs; k-prefix code c
OUTPUT: prefix code c'

1: Let a and b be the letters of cost ℓ_1 and ℓ_2, respectively.
2: For positive integer i, define $\text{enc}(i) = b'_1 b'_1 b'_2 b'_2 b'_j b'_j ab$ where $b'_1 b'_2 \dots b'_j$ is obtained from the binary representation $b_1 b_2 \dots b_j$ of i by replacing each "0" with "a" and each "1" with b. Define $\text{enc}(0) = ab$.
3: **for** each codeword of cost $\geq k$ **do**
4: Let α be the smallest prefix of cost $\geq k$ and β the remaining suffix. Replace the codeword by $\alpha\,\text{enc}(i)\beta\,b$, where i is the number of b's in β.
5: Return the modified code.

2. For every level $i > 1$, the set of codewords must contain exactly $f(i)$ codewords in level i.

Algorithm 1 uses a directed graph D that is computed by Algorithm 3 and given to Algorithm 1 as input. The nodes of D are all possible codeword costs in $[0, k]$, with an arc from ℓ to ℓ' if and only if $\ell' - \ell \in \{\ell_1, 1, \cdots, \ell_r\}$.

Here is the performance guarantee of Algorithm 1:

LEMMA 2. *Given a constraint $f = (f(0), \dots, f((k-1)/\epsilon))$, if there exists a leveled k-prefix code consistent with the constraint, then Algorithm 1 constructs one of minimum cost.*

PROOF. It is straightforward to verify that Algorithm 1 correctly computes each $v_S(\ell)$ and produces a leveled k-prefix code consistent with the constraint, if one exists.

Among the codes meeting the constraints of being leveled and consistent with f, the chosen code has minimum cost because the cost of codewords on levels $0, 1, \dots, (k-1)/\epsilon$ is determined by those constraints, and, given the codewords in those levels, the chosen code takes a set of codewords of cost $\geq k$ of minimum possible cost. □

4. CONVERTING A K-PREFIX CODE INTO A PREFIX CODE

Algorithm 2 converts a k-prefix code into a prefix code. The next lemma captures what we need to know about Algorithm 2 in order to use it in the main algorithm (Algorithm 3):

LEMMA 3. *Given any k-prefix code c of cost α, Algorithm 2 constructs a prefix code c' of cost at most $\alpha[1 + \ell_2(5 + 2\log_2 k)/k]$.*

PROOF. First we analyze the cost. Let $c(w) = \alpha\beta$ and $c'(w) = \alpha\,\text{enc}(i)\beta\,b$, respectively, be an original and modified codeword in Algorithm 2. From $i \leq \text{cost}(\beta)$ it follows that $\text{cost}(c'(w)) \leq \text{cost}(c(w)) + \ell_2[5 + 2\log_2 \text{cost}(c(w))]$. Since $\text{cost}(c(w)) \geq k$ if the codeword is modified, each modified codeword costs at most $1 + \ell_2(5 + 2\log_2 k)/k$ times as much as the original.

Next we show that c' is prefix free. Suppose $c'(v)$ is a prefix of $c'(w)$ for some $v, w \in W$. Since the original code was k-prefix free, it must be that

$$\begin{aligned} c'(v) &= \alpha\,\text{enc}(i)\beta b \\ \text{and} \quad c'(w) &= \gamma\,\text{enc}(j)\delta b \end{aligned}$$

where α and γ each have cost $\geq k$ but have no proper prefix of cost $\geq k$, and where i and j are the number of b's in β and δ, respectively (as in Algorithm 2). Since $c'(v)$ is a prefix of $c'(w)$, α is a prefix of $c'(w)$, which means $\alpha = \gamma$. Thus, $\text{enc}(i)$ is a prefix of $\text{enc}(j)\delta b$. Since every letter in enc() is doubled except the last two, it must be that $i = j$. Thus, βb is a prefix of δb. But (since $i = j$) β has the same number of b's as δ, so it must be that $\beta = \delta$. Finally, we can conclude that $\alpha\beta = \gamma\delta$. Since these were the original codewords assigned to v and w, it must be that $v = w$.

□

5. THE MAIN ALGORITHM

The main algorithm is Algorithm 3. It consists of:

1. Some preprocessing (Steps 1, 2, 3, and 4).
2. Calling Algorithm 1 for $O_\epsilon(1)$ different constraint tuples $(f(0), \dots, f((k-1)/\epsilon))$ (Steps 5, 6, 7).
3. Choosing the best k-prefix code among those output by Algorithm 2, transforming it into a prefix code (Step 9), and returning the resulting code.

Next we analyze the cost of the code produced.

LEMMA 4. *Step 4 partitions W into $O(k/\epsilon^2)$ groups.*

PROOF. Take any two consecutive groups other than G_1. The cumulative probability of the words in the two groups is at least $(1-p_1)\epsilon^2/(2k)$. Thus there can be at most $1 + 4k/\epsilon^2$ such groups. □

LEMMA 5. $OPT \geq 1 - p_1$.

PROOF. At most one codeword can belong to a^*. All the other codewords contain at least one letter which costs at least 1, and their cumulative frequency is at least $1 - p_{\max} = 1 - p_1$. □

We now focus on Steps 5 through 8. The analysis of Algorithm 1 implies that these find a code that is optimal among leveled k-prefix codes meeting the grouping constraints (mapping elements within each group to the same level). The next lemma implies that this code has cost at most $1 + O(\epsilon)$ times the minimum cost of any k-prefix code.

LEMMA 6. *For any k-prefix code c, there exists a leveled k-prefix code c' that maps group elements within each group to the same level, and such that cost$(c') \leq$ cost$(c)(1 + O(\varepsilon))$.*

Algorithm 3 — finds a $(1+O(\epsilon))$-optimal prefix code

INPUT: Set W of words with frequencies $p_1 \geq p_2 \geq \cdots \geq p_n > 0$; letter costs $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_r$; $\varepsilon > 0$.

OUTPUT: prefix code of cost at most $(1+O(\epsilon))$OPT

1: Adjust ϵ and the ℓ_i's so that $\ell_2 = 1$, ϵ is an integer divisor or multiple of ℓ_1, and each ℓ_i for $i \geq 2$ is a multiple of ϵ (as in Lemma 1).
2: Choose $k = \Theta(\log(1/\epsilon)/\epsilon)$ so $k-1$ is a multiple of ϵ.
3: Construct a directed graph D whose nodes are all codeword costs in $[0,k]$, with an arc from ℓ to ℓ' iff $\ell' - \ell \in \{\ell_1, \ell_2, \cdots, \ell_r\}$. Do this by enumerating the vertices and edges of D using breadth-first search from the root (node 0).
4: Greedily partition W into groups.

- The first group is $G_1 = \{w_1\}$.
- Take i maximum such that $p_i > (1-p_1)\epsilon^2/(2k)$. Then $G_2 = \{w_2\}$, $G_3 = \{w_3\}$, ..., and $G_i = \{w_i\}$.
- While W is not empty, greedily take for the next group $\{w_j, w_{j+1}, \ldots, w_\ell\}$, where $p_j + \cdots + p_\ell \leq (1-p_1)\epsilon^2/k < p_j + \cdots + p_{\ell+1}$.

5: By exhaustive search, guess whether there is a codeword of cost < 1 and what its size is (call it $f(0)$), and, for each $i = 1, 2, \ldots, (k-1)/\epsilon$, guess which groups will be assigned to level i (i.e., with costs in $[1+\varepsilon(i-1), 1+\varepsilon i)$) and let $f(i)$ denote the total number of words in those groups.
6: **for** each such guess $(f(0), \ldots, f((k-1)/\epsilon))$ **do**
7: Use Algorithm 1 to construct an optimal k-prefix code consistent with the guess (if one exists).
8: Let c be the minimum-cost code constructed.
9: Using Algorithm 2, convert c into a prefix code c'.
10: Return c'.

PROOF. Let c be an optimal k-prefix code. We modify c level by level so that for each i, c is maximal within level i and so that level i contains all or none of the elements of each group, as follows. Since level 0 contains at most one codeword, and the first group contains exactly one element, this is already true for level 0. Assume that we have already modified c to guarantee those properties for levels up to $i-1$, and consider level i.

Leveling phase for level i. First modify c so that it is maximal within level i by taking every codeword z in level i and padding it with enough a's (i.e., replace z by za^j) so that its cost is $1 + i\epsilon - \min\{\ell_1, \epsilon\}$, i.e. so that adding one more letter a would move the codeword out of level i. (Here we use the assumption that every letter cost is a multiple of ϵ, with the possible exception of ℓ_1 which then evenly divides ϵ.)

Grouping phase for level i. Next modify the code so that level i contains all or none of the elements of each group (i.e., no group is "split"). By induction, no group is split in levels $0, \ldots, i-1$. Since the code is ordered, at most one group can be split from level i. Take each codeword in that group and on level i and pad it with an a to move it out of level i. Then, if necessary, reorder the code (on levels greater than i).

When this construction has been done for every i, the original k-prefix code c has been changed into a leveled k-prefix code c' that maps elements within each group to the same level. Next we show that the cost of c' is $1+O(\epsilon)$ times the cost of c.

First consider a codeword that is padded in the leveling phase for some level i but not in the grouping phase for that level. Since each codeword has this happen at most once, and the operation increases the cost of the codeword by at most ϵ times the original cost of the codeword, the total increase in cost due to such operations is at most ϵ times the original cost of the code.

Next consider codewords that are padded in the grouping phases. In the leveling and grouping phases for a given level i, the cost of a codeword is increased by at most $\epsilon + \ell_1 \leq 2$. Since only one group is padded in the grouping phase, and (by the grouping criterion) the group has total probability at most $(1-p_1)\epsilon^2/k$, the total increase in cost of all codewords in the group during the leveling and grouping phase for level i is at most $2(1-p_1)\epsilon^2/k$. Since there are $O(k/\epsilon)$ levels, the total increase in cost due to these operations for all levels is $O((1-p_1)\epsilon)$. By Lemma 5, this is $O(\epsilon)$ times the original cost of the code.

□

Together with Lemma 2, Lemma 6 implies that at the end of Step 8, Algorithm 3 finds a k-prefix code that has cost at most $(1+O(\epsilon))OPT$. Lemma 3 implies that, for k chosen as in Step 2, the main algorithm converts the k-prefix code into a prefix code that also has cost at most $(1+O(\epsilon))OPT$. This concludes the proof of the performance guarantee in Theorem 1 for the case $\ell_1 > \epsilon/n$.

6. ANALYSIS OF RUNNING TIME

We analyze the directed graph D which was built in Step 3 of Algorithm 3 and used by Algorithm 1.

LEMMA 7. *Graph D has at most nk/ϵ nodes and dnk/ϵ edges, where d is the number of distinct letter costs among $\{\ell_1, \ldots, \ell_r\}$.*

PROOF. From Step 1 of Algorithm 3 all letter costs other than ℓ_1 are multiples of ϵ, and ℓ_1 is either a multiple of ϵ or evenly divides ϵ. Moreover, we've assumed $\ell_1 > \epsilon/n$. Thus each of the k/ϵ levels has at most n distinct costs. Thus D has at most nk/ϵ nodes; each node has outdegree at most d, the number of distinct letter costs. □

In particular, in Step 3 of Algorithm 3 graph D can be constructed in time $O(ndk/\epsilon)$. Using an array with a bucket for each possible codeword cost, we can access the nodes of D in constant time.

LEMMA 8. *Algorithm 1 can be implemented to run in time $O(ndk/\epsilon)$.*

PROOF. Since $\ell_1 > \epsilon/n$, there are $O(n/\epsilon)$ nodes on level 0, so Step 3 of Algorithm 1 takes time $O(n/\epsilon)$.

The loop of Step 5 is iterated k/ϵ times. The total time for Step 6, is at most $O(1)$ per edge, therefore at most dnk/ϵ total.

Total time for Step 7 is $O(n)$ ($O(1)$ per codeword). (The exact cost of each iteration depends on details of implementation of the codeword set S. We use an implicit representation of the coding tree corresponding to S by highlighting an edge e of D and giving it a "codeword weight" $w(e)$ if it is used to build $w(e)$ codewords of S. This data structure can be used to obtain the desired time bounds. The actual code S can be constructed from this implicit representation in time proportional to the size of S.)

Finally, Step 8 is implemented by doing a breadth-first search to extend D beyond cost k, starting from all the nodes of D which have $v_S(\ell) \geq 1$. Since the outdegree of any node is bounded by d, and we stop as soon as we have identified at most n shortest eligible strings of cost $\geq k$, this has complexity $O(nd)$.

Overall, the algorithm thus has running time $O(ndk/\epsilon)$. □

Remark: Constructing an explicit representation of S would require, in general, complexity $\Omega(n^2)$, since the optimal codewords can have total length $\Omega(n^2)$ (for example, if $S = \{b, ab, a^2b, \ldots, a^nb\}$, as might be the case when $\ell_1 \ll 1$).

LEMMA 9. *Algorithm 2 can be implemented in time $O(n)$.*

PROOF. (sketch) First represent S with a tree as in the usual Huffman encoding problem, compacting the representation as you go so that every internal node has at least two children, and so the total number of nodes is $O(n)$. Then calculate the number of b's recursively in time $O(n)$. Finally, modify the tree to insert the extra letters. This can be implemented in time $O(n)$. □

We now go back to Algorithm 3. It is easy to see that everything outside Step 6 takes time $O(nd)$. From Lemma 4, there are only $O(k/\epsilon^2)$ groups, and $O(k/\epsilon)$ levels. Since the assignment of groups to levels respects the ordering of both, the map is determined by knowing, for each level, the maximum index of any group assigned to it. Thus there are $\exp(O(\ln(k/\epsilon)k/\epsilon))$ maps from groups to levels, hence $\exp(O(\ln(k/\epsilon)k/\epsilon))$ choices for $(f(1), \ldots, f(k/\epsilon))$.

How about $f(0)$: how many possibilities must we try? As written, the algorithm tries every possibility in level 0, which could mean as many as $\Theta(n)$, however it is easy to modify the algorithm so that we only try $O(\log(n)/\epsilon)$ possibilities for $f(0)$: indeed, rounding costs up to the nearest power of $(1+\epsilon)$, it is enough to try $f(0) = 0, 1, \ldots, 1/\epsilon, (1/\epsilon)(1+\epsilon), (1/\epsilon)(1+\epsilon)^2, \ldots, 1$.

Since each iteration takes time $O(ndk/\epsilon)$, and there are $\exp[O(\ln(k/\epsilon)k/\epsilon)]\log(n)/\epsilon$ iterations, we obtain a complexity of $(nd\log(n)/\epsilon^2)\exp(O(\ln(k/\epsilon)k/\epsilon))$. Expanding k and simplifying, this is $nd\log(n)\exp(O(\ln(1/\epsilon)^2/\epsilon^2))$. This completes the proof of Theorem 1 for the case $\ell_1 > \epsilon/n$.

7. DEALING WITH THE CASE $\ell_1 \leq \epsilon/N$

Recall that a is the shortest letter and b the second shortest letter, of cost $\ell_2 = 1$.

LEMMA 10. *If $\ell_1 \leq \epsilon/n$, then there exists a prefix code c with the following properties. The cost of c is at most $1+\epsilon$ times the minimum cost of any prefix code. There is a unique codeword of the form a^{i_0} where i_0 is of the form $\lfloor(1+\epsilon)^j\rfloor$ with $j = O(\log(n)/\epsilon)$. Every other codeword has one of the following forms:*

1. *ba^jb for some $j \leq n$,*
2. *a^jxa^n where $x \in \Sigma - a$ and $j < i_0$.*

PROOF. Given a minimum-cost prefix code, let i_0 be such that a^{i_0} is the unique codeword in a^*. Modify the code as follows.

1. Replace codeword a^{i_0} by a^{i_1}, where i_1 is the next largest integer of the form $\lfloor(1+\epsilon)^j\rfloor$. Since $i_0 < n$, $j = O(\log(n)/\epsilon)$

2. For each i ($1 \leq i \leq n$), if the ith codeword has two or more occurrences of letters other than a, replace the codeword by ba^ib.

3. Any codeword not modified in Step 1 or 2 must be of the form a^jxa^* where $x \in \Sigma - a$ and $0 \leq j < i_0$. (For any j and x there can be at most one such starting with a^jx.) Replace it by a^jxa^n.

This gives a prefix code of the desired form. It is easy to verify that the cost of each codeword is increased by at most a $1+\epsilon$ factor. □

The algorithm will first find the minimum-cost code among those of the form described in Lemma 10 as follows: for each of the $O(\log(n)/\epsilon)$ choices for i_0, consider the ordered code whose codewords are the n least costly strings in the set

$$\{a^{i_0}\} \cup \{ba^jb : j < n\} \cup \{a^jxa^n : j < i_0\}.$$

Among the codes considered, take the one of minimum cost. By the lemma, the cost of this code will by at most $1+\epsilon$ times the minimum cost of any prefix code.

Under Assumption 1, each iteration can be implemented in $O(n+d)$ time, so for this special case the algorithm takes $O((n+d)\log(n)/\epsilon)$ time.

8. ACKNOWLEDGMENTS

The first author's work was partially supported by HK RGC Competitive Research Grants HKUST 6137/98E, 6162/00E and 6082/01E.

9. REFERENCES

[1] Shimon Even Abraham Lempel and Martin Cohen. An algorithm for optimal prefix parsing of a noiseless

and memoryless channel. *IEEE Transactions on Information Theory*, 19(2):208–214, March 1973.

[2] Julia Abrahams. Code and parse trees for lossless source encoding. *Communications in Information and Systems*, 1(2):113–146, April 2001.

[3] Doris Altenkamp and Kurt Melhorn. Codes: Unequal probabilies, unequal letter costs. *Journal of the Association for Computing Machinery*, 27(3):412–427, July 1980.

[4] Toby Berger and Raymond W. Yeung. Optimum 1-ended binary prefix codes. *IEEE Transactions on Information Theory*, 36(6):1434–1441, November 1990.

[5] N. M. Blachman. Minimum cost coding of information. *IRE Transactions on Information Theory*, PGIT-3:139–149, 1954.

[6] Siu-Ngan Choi and M. Golin. Lopsided trees: Algorithms, analyses and applications. In *Proc. of the 23rd International Colloquium on Automata Languages and Programming (ICALP '96)*, pages 538–549, 1996.

[7] N. Cot. Complexity of the variable-length encoding problem. In *Proc. 6th Southeast Conference on Combinatorics, Graph Theory and Computing*, pages 211–244, 1975.

[8] Norbert Cott. *Characterization and Design of Optimal Prefix Codes*. PhD Thesis, Stanford University, Department of Computer Science, June 1957.

[9] I. Csisz'ar. Simple proofs of some theorems on noiseless channels. *Inform. Contr.*, 514:285–298, 1969.

[10] E. N. Gilbert. How good is morse code. *Inform Control*, 14:585–565, 1969.

[11] E. N. Gilbert. Coding with digits of unequal costs. *IEEE Trans. Inform. Theory*, 41:596–600, 1995.

[12] M. Golin and G. Rote. A dynamic programming algorithm for constructing optimal prefix-free codes for unequal letter costs. *IEEE Transactions on Information Theory*, 44(5):1770–1781, 1998.

[13] M. Golin and N. Young. Prefix codes: Equiprobable words, unequal letter costs. *SIAM Journal on Computing*, 25(6):1281–1292, December 1996.

[14] K. Hinderer. On dichotomous search with direction-dependent costs for a uniformly hidden objec. *Optimization*, 21(2):215–229, 1990.

[15] D. A. Huffman. A method for the construction of minimum redundancy codes. In *Proc. IRE 40*, volume 10, pages 1098–1101, September 1952.

[16] K. A. S. Immink. *Codes for Mass Data Storage Systems*. Shannon Foundations Publishers, 1999.

[17] I. Itai. Optimal alphabetic trees. *SIAM J. Computing*, 5:9–18, 1976.

[18] Sanjiv Kapoor and Edward M. Reingold. Optimum lopsided binary trees. *Journal of the Association for Computing Machinery*, 36(3):573–590, July 1989.

[19] Richard Karp. Minimum-redundancy coding for the discrete noiseless channel. *IRE Transactions on Information Theory*, IT-7:27–39, January 1961.

[20] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.

[21] R. M. Krause. Channels which transmit letters of unequal duration. *Inform. Contr.*, 5:13–24, 1962.

[22] R.S. Marcus. *Discrete Noiseless Coding*. M.S. Thesis, MIT E.E. Dept, 1957.

[23] K. Mehlhorn. An efficient algorithm for constructing nearly optimal prefix codes. *IEEE Trans. Inform. Theory*, 26:513–517, September 1980.

[24] L. L. Larmore P. Bradford, M. Golin and W. Rytter. Optimal prefix-free codes for unequal letter costs and dynamic programming with the monge property. In *Proc. Sixth European Symposium on Algorithms (ESA'98) (LNCS 1461)*, pages 43–54, 1998.

[25] Y. Perl, M. R. Garey, and S. Even. Efficient generation of optimal prefix code: Equiprobable words using unequal cost letters. *Journal of the Association for Computing Machinery*, 22(2):202–214, April 1975.

[26] A. De Santis R. M. Capocelli and G. Persiano. Binary prefix codes ending in a 1. *IEEE Transactions on Information Theory*, 40(4):1296–1302, July 1994.

[27] E. Korach S. Dolev and D. Yukelson. The sound of silence: Guessing games for saving energy in mobile environment. In *Proc. of the Eighteenth Annual Joint Conference of IEEE Computer and Communications Societies (IEEE INFOCOM'99)*, pages 768–775, 1999.

[28] L. E. Stanfel. Tree structures for optimal searching. *Journal of the Association for Computing Machinery*, 17(3):508–517, July 1970.

[29] B. Varn. Optimal variable length codes (arbitrary symbol cost and equal code word probability). *Information Control*, 19:289–301, 1971.