Optimal Prefix-Free Codes for Unequal Letter Costs: Dynamic Programming with the Monge Property *

Phil Bradford † **Mordecai J. Golin** ‡
Lawrence L. Larmore § **Wojciech Rytter** ¶

Abstract

In this paper we discuss a variation of the classical *Huffman coding problem*: finding optimal prefix-free codes for unequal letter costs. Our problem consists of finding a minimal cost prefix-free code in which the encoding alphabet consists of unequal cost (length) letters, with lengths α and β. The most efficient algorithm known previously required $O(n^{2+\max(\alpha,\beta)})$ time to construct such a minimal-cost set of n codewords. In this paper we provide an $O(n^{\max(\alpha,\beta)})$ time algorithm. Our improvement comes from the use of a more sophisticated modeling of the problem combined with the observation that the problem possesses a "Monge property" and that the SMAWK algorithm on monotone matrices can therefore be applied.

*This work was partially done by the last two authors while visiting Department of Computer Science, Bonn University, Germany

†Max-Planck-Institut für Informatik, 66123 Saarbruecken, Germany

‡Hong Kong UST, Clear Water Bay, Kowloon, Hong Kong. email:GOLIN@CS.UST.HK This research partially supported by HK RGC CERG grant 652/95E

§Department of Computer Science, University of Nevada, Las Vegas, NV 89154-4019. Email: larmore@cs.unlv.edu. Research supported by NSF grant CCR-9503441.

¶Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02–097 Warszawa, Poland, and Department of Computer Science, University of Liverpool. of Computer Science, Bonn University.

1 Introduction

The problem of finding optimal prefix-free codes for unequal letter costs (and the associated problem of constructing optimal lopsided trees) is an old and hard classical one. The problem consists of finding a minimal cost prefix-free code in which the encoding alphabet consists of unequal cost (length) letters, of lengths α and β, $\alpha \leq \beta$. The code is represented by a *lopsided tree*, in the same way as a Huffman tree represents the solution of the Huffman coding problem. Despite the similarity, the case of unequal letter costs is much harder then the classical Huffman problem; no polynomial time algorithm is known for general letter costs, despite a rich literature on the problem, *e.g.*, [1, 7]. However there are known polynomial time algorithms when α and β are integer constants [7].

The problem of finding the minimum cost tree in this case was first studied by Karp [9] in 1961 who solved the problem by reduction to integer linear programming, yielding an algorithm exponential in both n and β. Since that time there has been much work on various aspects of the problem such as; bounding the cost of the optimal tree, Altenkamp and Mehlhorn [2], Kapoor and Reingold [8] and Savari [15]; the restriction to the special case when all of the weights are equal, Cot [5], Perl Gary and Even [14], and Choi and Golin [4]; and approximating the optimal solution, Gilbert [6]. Despite all of these efforts it is still, surprisingly, not even known whether the basic problem is polynomial-time solvable or in NP-complete.

The only technique other than Karp's for solving the problem is due to Golin and Rote [7] who describe an $O(n^{\beta+2})$-time dynamic programming algorithm that constructs the tree in a top-down fashion. This is the the most efficient known algorithm for the case of small β; in this paper we apply a different approach by constructing the tree in a bottom-up way and describing more sophisticated attacks on the problem. The first attack permits reducing the search space in which optimal trees are searched for. The second shows how, surprisingly, monotone-matrix concepts, e.g., the *Monge property* [13] and the SMAWK algorithm [3] can be utilized.

Combining these two attacks improves the running time of of [7] by a factor of $O(n^2)$ down to $O(n^\beta)$.

Our approach requires a better understanding of the combinatorics of lopsided trees; to achieve this we also introduce the new crucial concept of *characteristic sequences*.

Let $0 \leq \alpha \leq \beta$. A tree T is a *binary lopsided α, β tree* (or just a *lopsided tree*) if every non-leaf node u of the tree has two sons, the length of the edge connecting u to its left son is α, and the length of the edge connecting u to its right son is β. Figure 1 shows a 2-5 lopsided tree. Let T be a lopsided tree and $v \in T$ some node. Then

$$\begin{aligned} \mathit{depth}(T, v) &= \text{sum of the lengths of the edges connecting } \mathit{root}(T) \text{ to } v \\ \mathit{depth}(T) &= \max\{\mathit{depth}(T, v) : v \in T\} \end{aligned}$$

For example, the tree in Figure 1 has depth 20. Now suppose we are given a sequence of nonnegative weights $P = \{p_1, p_2, \ldots, p_n\}$. Let T be a lopsided tree with n leaves labeled $v_1, v_2, \ldots, v_n$. The weighted external path length of the tree is

$$\mathit{cost}(T, P) = \textstyle\sum_i p_i \cdot \mathit{depth}(T, v_i).$$

Given P, the problem that we wish to solve is to construct a labeled tree T that minimizes $\mathit{cost}(T, P)$.

As was pointed out quite early [9] this problem is equivalent to finding a minimal cost prefix-free code in which the encoding alphabet consists of two (or generally, more) unequal cost (length) letters, of lengths α and β. Also note that if $\alpha = \beta = 1$ then the problem reduces directly to the standard Huffman-encoding problem.

Notice that, given any particular tree T, the cost actually depends upon the labeling of the leaves of T, the cost being minimized when $p_1 \leq p_2 \leq \cdots \leq p_n$ and $\mathit{depth}(T, v_1) \geq \mathit{depth}(T, v_2) \geq \cdots \geq \mathit{depth}(T, v_n)$. We therefore will always assume that the leaves of T are labeled in nonincreasing order of their depth. We will also assume that $p_1 \leq p_2 \leq \cdots \leq p_n$.

Note: In this extended abstract we omit many technical proofs.

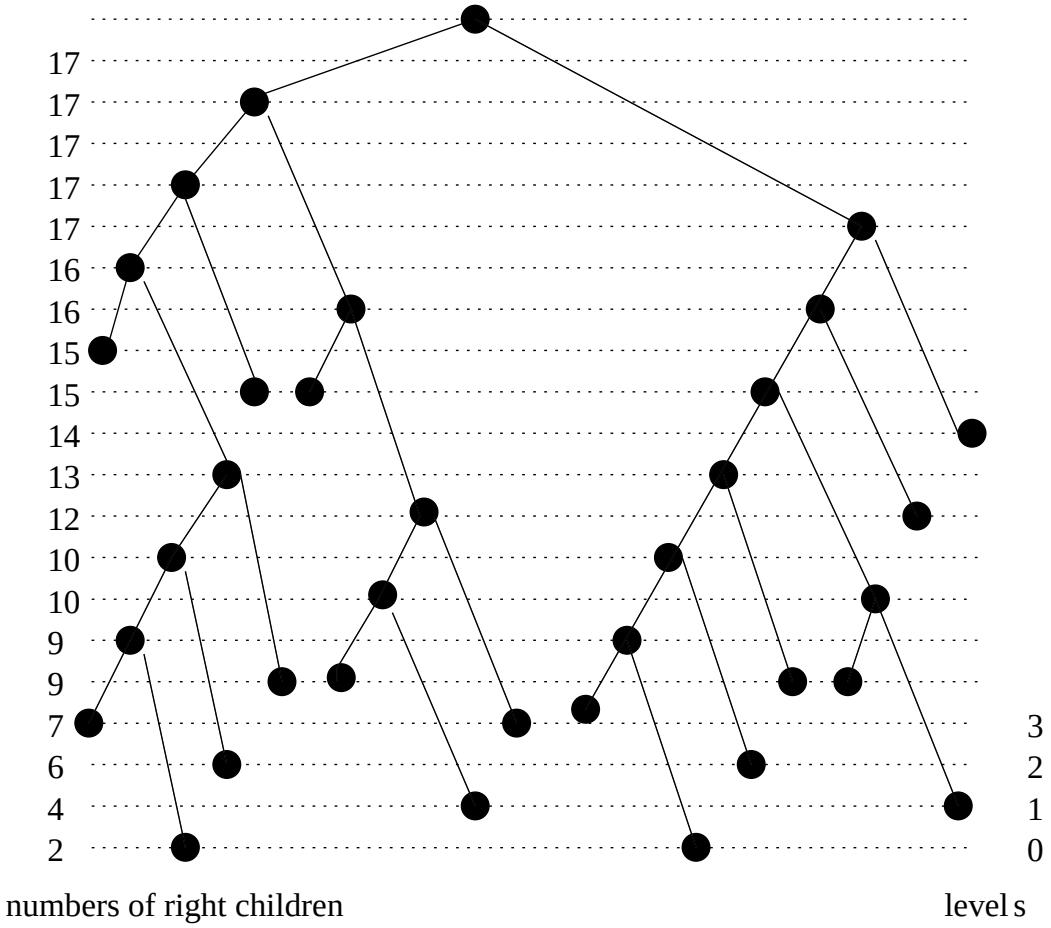

Figure 1: An example 2-5 tree T. The characteristic sequence $B = \mathit{sequence}(T)$ is (2,4,6,7,9,9,10,10,12,13,14,15,16,16,17,17,17,17,17).

2 Combinatorics of lopsided trees and monotonic sequences

The first crucial concept in this paper is the *characteristic sequence* of a tree T. Denoted by $\mathit{sequence}(T)$ this is the vector $B_T = (b_0, b_1, \ldots, b_{d-1})$ in which b_i is the number of right children on or below level i for $0 \leq i < d$, where d is the height of the tree. and the levels are enumerated from bottom to top (See Figure 1).

Let n and P be fixed. Now let $B = b_0, b_1, \ldots, b_{d-1}$ be any sequence, not necessarily one of the form $B = B_T$ defined by some tree T. B is said to be

monotonic if $d \geq \beta$ and

$$0 \leq b_0 \leq b_1 \leq b_2 \leq \cdots \leq b_{d-1}.$$

Note that the number of right children on or below level i of tree T can not decrease with i so for all trees T, B_T is a monotonic sequence.

A monotonic sequence B of length d *terminates* in a β-tuple $(\gamma_\beta, \gamma_{\beta-1}, \ldots, \gamma_1)$ if $\forall j$, $0 \leq j < \beta$, $b_{d-j} = \gamma_j$. Note that if T is a lopsided tree with n leaves then T must have $n-1$ internal nodes and thus $n-1$ right children. Furthermore the top β levels of T can not contain any right children. Thus if $B = \mathit{sequence}(T)$ for some T then B terminates in a β tuple $(n-1, n-1, \ldots, n-1)$.

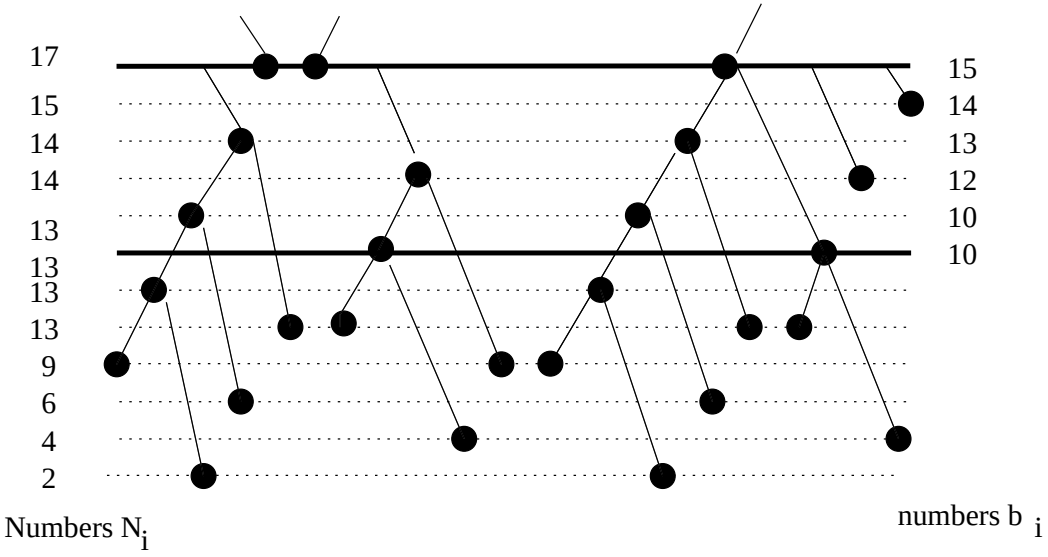

Figure 2: The bottom forest F_{11} of the tree T from Figure 1.

For a monotonic sequence $B = b_0, b_1, \ldots, b_{d-1}$ define

$$N_k(B) = b_k + b_{k-(\beta-\alpha)} - b_{k-\beta}, \qquad S_i = \sum_{j \leq i} p_j, \qquad \mathit{cost}(B, P) = \sum_{0 \leq k < d} S_{N_k(B)} \tag{1}$$

If $i < 0$ or $i > n$ then $S_i = \infty$. For a tree T, denote by $\mathcal{F}_k = \mathit{forest}_k(T)$ the forest resulting by taking all nodes at level k and below (See Figure 2). Denote by $N_k(T)$ the number of leaves in $\mathit{forest}_k(T)$. (Note that we have overloaded the notation $N_k(\ )$ to apply to both trees and sequences.)

The following lemma collects some basic facts:

Lemma 1 *Let T be a lopsided tree and $B = sequence(T)$. Then*
(P1) $cost(T,P) = \sum_{0 \le k < depth(T)} S_{N_k(T)}$,
(P2) $\forall\ 0 \le i < d = depth(T), \quad N_i(T) = b_i + b_{i-(\beta-\alpha)} - b_{i-\beta}$
(where $\forall\ j < 0,$ we set $b_j = 0$).
(P3) $cost(T,P) = cost(B,P)$,

Proof.
We omit the proof of (P1) which is straightforward but tedious. To prove (P2), note that $\mathcal{F}_i$ is a forest, hence

$$\begin{aligned} N_i(T) &= \{u \in F_i \,:\, u \text{ is a leaf in } \mathcal{F}_i\} && (2)\\ &= \textit{Number of internal nodes in } \mathcal{F}_i + \textit{Number of trees in } \mathcal{F}_i && (3)\end{aligned}$$

The first summand in the last line is easily calculated. A node at height k is internal in $\mathcal{F}_i$ if and only if it is the father of some right son at level $k - \beta$. Thus

$$\textit{Number of internal nodes in } \mathcal{F}_i = b_{i-\beta}. \qquad (4)$$

The second summand is only slightly more complicated to calculate. The number of trees in $\mathcal{F}_i$ is exactly the same as the number of *tree-roots* in $\mathcal{F}_i$. Now note that a node in $\mathcal{F}_i$ is a tree-root in $\mathcal{F}_i$ if and only if its father is not in $\mathcal{F}_i$. Thus a right son at height k in $\mathcal{F}_i$ is a tree-root if and only if $i - \beta < k \le i$ and there are exactly $b_i - b_{i-\beta}$ such nodes.

Similarly a left son at height k is a tree-root if and only if $i - \alpha < k \le i$. This may occur if and only if the left son's right brother is at height k, where $i - \beta < k \le i - (\beta - \alpha)$. The number of such nodes is therefore $b_{i-(\beta-\alpha)} - b_{i-\beta}$.

We have therefore just seen that

$$\textit{Number of trees in } \mathcal{F}_i = (b_i - b_{i-\beta}) + (b_{i-(\beta-\alpha)} - b_{i-\beta}). \qquad (5)$$

Combining (4) and (5) completes the proof of (P2). (P3) follows from (P1) and (P2). □

Now define a sequence B to be *legal* if B is monotonic and $B = sequence(T)$ for some lopsided tree T. The lemma implies that minimizing cost over all legal sequences is exactly the same as minimizing cost over all lopsided trees.

However, not all sequences are legal so this knowledge does not at first seem to help us. In the next section we sketch a proof of the following fact. Given any minimum-cost *monotonic* sequence that terminates in the β-tuple $(n-1, n-1, \ldots, n-1)$ it is possible to build a legal sequence with the same cost. Since all legal sequences are monotonic this legal sequence must be a minimal-cost legal sequence and thus correspond to a minimum-cost tree. In other words, to find a minimal-cost tree it will suffice to find a minimum-cost monotonic sequence terminating in $(n-1, n-1, \ldots, n-1)$.

3 Relation between minimum sequences and optimal trees

We start by assuming that $B = \mathit{sequence}(T)$ for some T. In T the weight p_1 is associated with some lowest leaf at level 0. The left sibling of this leaf is associated with some other weight p_k. How can such a k be identified?

Observe that this sibling can be the lowest leaf in the tree which is a left-son, i.e., the lowest left node in T. Such a node appears on level $\beta - \alpha$ (see the left tree in Figure 3). The number of leaves below this level is $b_{\beta-\alpha-1}$, so assuming that we list items consecutively with respect to increasing levels, the lowest left-son leaf has index $k = \mathit{FirstLeft}(B)$, where

$$\mathit{FirstLeft}(B) = b_{\beta-\alpha-1} + 1$$

We state without proof the intuitive fact that if T is an optimal tree in which p_1, p_k label sibling leaves, then the tree T' that results by (i) removing those leaves and (ii) labeling their parent (now a leaf) with $p_1 + p_k$ will also be an optimal tree for the leaf set $P' = P \cup \{p_1 + p_k\} - \{p_1, p_k\}$ (see the right tree in Figure 3), Also, calculation shows that

$$\mathit{cost}(T, P) = \mathit{cost}(T', P') + \beta \cdot p_1 + \alpha \cdot p_k. \tag{6}$$

The rest of this section will be devoted to translating this intuition into facts about trees and sequences.

If $p_1,\ p_k$ are siblings in a tree T then denote by $T' = merge(T, 1, k)$ the tree in which leaves $p_1,\ p_k$ are removed and their parent is replaced by a leaf with weight $p_1 + p_k$ (see Figure 3). We also write $unmerge(T', 1, k) = T$. Thus

$$cost(unmerge(T', 1, k), P) = cost(T', P') + \beta \cdot p_1 + \alpha \cdot p_k. \tag{7}$$

For the sequence $B = (b_0, b_1, \ldots b_d)$ denote

$$dec(B) = B' = (b_0 - 1, b_1 - 1, b_2 - 1, \ldots b_d - 1).$$

Note that (after any leading zeros are deleted) this sequence is the characteristic sequence of $T' = merge(T, 1, k)$.

Assume Γ is a sorted sequence of positive integers, x is a positive integer, and $insert(\Gamma, x)$ is the sequence Γ with x inserted and sorted (as in insertion sort). Now denote by $delete(P, p_1, p_k)$ the sequence P with elements p_1 and p_k deleted, and define

$$P' = package\_merge(P, 1, k) = insert(delete(P, p_1, p_k), p_1 + p_k).$$

For example if $P = \{2, 3, 4, 5, 10\}$ then

$$\begin{aligned} P' = delete(P, 2, 4) &= \{3, 5, 10\} \\ insert(P', 6) &= \{3, 5, 6, 10\} \\ package\_merge(P, 1, 3) &= \{3, 5, 6, 10\} \end{aligned}$$

Lemma 3.1 (insertion-sort lemma)
If $t \leq length(\Gamma)$ and Γ is a sorted sequence then

(1) $Pref_t(insert(\Gamma, x)) \leq Pref_t(\Gamma)$, **(2)** $Pref_t(insert(\Gamma, x)) \leq Pref_{t-1}(\Gamma) + x$.

Lemma 3.2 *If $j \geq k$ then*

(1) $Pref_{j-2}(P') \leq Pref_j(P) - p_1 - p_k$, **(2)** $Pref_{j-1}(P') \leq Pref_j(P)$.

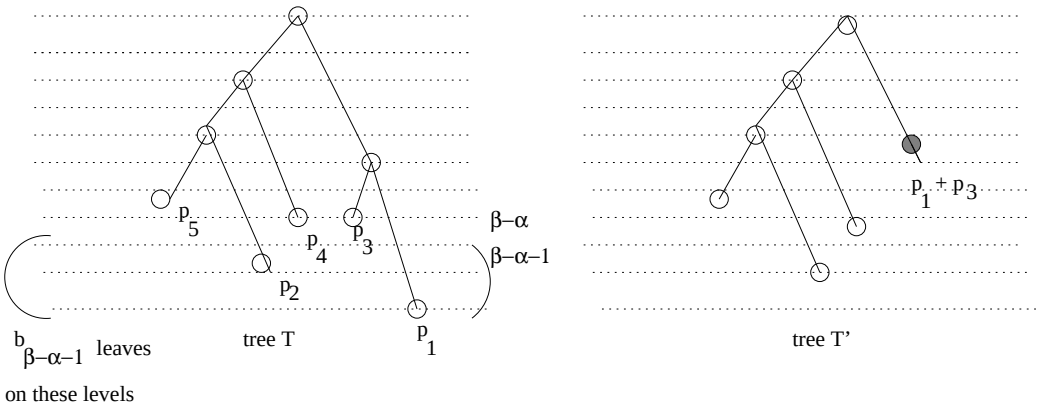

Figure 3: The correspondence between trees T, T' and their sequences: $T' = merge(T, 1, 3)$ and $sequence(T) = B = (1, 2, 2, 3, 3, 4, 4, 4, 4, 4)$ $sequence(T') = dec(B) = B' = (0, 1, 1, 2, 2, 3, 3, 3, 3, 3)$; $FirstLeft(B) = b_{\beta-\alpha-1} + 1 = b_{5-2-1} + 1 = 3$ and $cost(T) = cost(T') + 2p_4 + 5p_1$.

Proof: Let $delete(P, p_1, p_k) = \Gamma$, observe that for $j \geq k$ we have

$$Pref_{j-2}(\Gamma) = Pref_j(P) - p_1 - p_2 \tag{8}$$

(1) Apply point (1) of Lemma 3.1 and Equation 8 to the sequence Γ, where $P' = insert(\Gamma, x)$ with $x = p_1 + p_k$ and $t = j - 2$.

(2) Apply point (2) of Lemma 3.1 with $x = p_1 + p_2$. We have

$$Pref_{j-1}(P') = Pref_{j-1}(insert(\Gamma, x)) \leq Pref_{j-2}(\Gamma) + x = Pref_j(P),$$

due to Equation 8. This completes the proof. □

Lemma 3.3 (key-lemma)
Let $k = FirstLeft(B)$, $P' = merge\_package(P, 1, k)$ and $B' = dec(P)$, then

$$cost(B', P') \leq cost(B, P) - \beta \cdot p_1 - \alpha \cdot p_k.$$

Proof: Observe that

$$N_i(B') = \begin{cases} N_i(B) - 1 & \text{if } i < \beta - \alpha \\ N_i(B) - 2 & \text{if } \beta - \alpha \leq i < \beta \\ N_i(B) - 1 & \text{if } \beta \leq i < d \end{cases}$$

The ith part (denoted $term(i, B)$) of the cost of B is $Pref_{N_i(B)}(P)$, and ith part (denoted $term(i, B')$) of the cost of B' is $Pref_{N_i(B')}(P')$. We now proceed to the case by case analysis

CASE 1: $\underline{i < \beta - \alpha.}$ Then $term(i, B) - term(i, B') = p_1$. Summing over i's yields

$$\sum_{0 \le i < \beta - \alpha} term(i, B') = \sum_{0 \le i < \beta - \alpha} term(i, B) - (\beta - \alpha)p_1. \tag{9}$$

CASE 2: $\underline{\beta - \alpha \le i < \beta}$ Now $term(i, B) = Pref_j(P)$ and $term(i, B') = Pref_{j-2}(P')$ for some $j \ge k$, and, due to Lemma 3.2, the difference between these values is at least $p_1 + p_k$. Hence

$$\sum_{\beta - \alpha \le i < \beta} term(i, B') \le \sum_{\beta - \alpha \le i < \beta} term(i, B) - \alpha(p_1 + p_k). \tag{10}$$

CASE 3: $\underline{\beta \le i}$ In this case $term(i, B) = Pref_j(P)$ and $term(i, B') = Pref_{j-1}(P')$ for some $j \ge k$, and due to Lemma 3.2 $term(i, B') \le term(i, B)$. Hence

$$\sum_{\beta \le i} term(i, B') \le \sum_{\beta \le i} term(i, B). \tag{11}$$

Combining (9), (10) and (11) yields the thesis. □

This lemma permits us to prove that minimum-cost monotonic sequences have the same cost as minimum cost trees and permit the construction of such trees:

Theorem 3.4 ((correctness))
Assume B is a minimum cost monotonic sequence terminating in $(n-1, n-1, \ldots, n-1)$ for the sequence P. Then there is a tree T such that:
(1) *$cost(T, P) = cost(B, P)$.*
Furthermore if $n > 2$ then
(2) *$FirstLeft(B)$ is the index of the left brother of p_1 in T,*
(3) *$B' = dec(B)$ is a minimum cost sequence for $P' = package\_merge(P, 1, k)$.*

Proof:
The proof is by induction with respect to the number n of items in P. If $n = 2$ then all legal sequences have the form

$$b_0 = b_1 = \cdots = b_{d-1} = 1$$

where $d \ge \beta$. The sequence with $d = \beta$ has minimum cost and this sequence is also the minimum-cost monotonic sequence.

So now suppose that $n > 2$. Let $B' = dec(B)$ and T' be a minimum cost tree for P'. P' has $n-1$ items, so by the induction hypothesis $cost(T', P')$ equals the minimum cost of a monotonic sequence for P'. In particular, by Lemma 3.3, we have

$$cost(T', P') \leq cost(B', P') \leq cost(B, P) - \alpha \cdot p_k - \beta \cdot p_1. \tag{12}$$

Take $T = unmerge(T', 1, k)$. Then by Equality (6) and Inequality (12) we have:

$$cost(sequence(T), P) = cost(T, P) = cost(T', P') + \alpha \cdot p_k + \beta \cdot p_1 \leq cost(B, P).$$

B was chosen to be a minimal cost sequence so all of the inequalities must be equalities and, in particular, we find that $cost(T, P) = cost(B, P)$. Hence T is the required tree, and this completes the proof of (**1**).

We also find that

$$cost(T', P') + \alpha \cdot p_k + \beta \cdot p_1 = cost(B, P)$$

so plugging back into (12) we find that $cost(T', P') = cost(B', P')$. Since T' is a minimal cost tree for P' the induction hypothesis implies B' is a minimum cost sequence for P' proving (**3**). The proof of (**2**) follows from the details of the construction. □

Note that this theorem immediately implies that, given a minimum-cost sequence B for P, we can construct a minimum-cost tree for P. If $n = 2$ the tree is simply one root with two children. If $n > 2$ calculate $B' = dec(B)$ and $P' = package\_merge(P, 1, k)$ in $O(n)$ time. Recursively build the optimal tree T' for P' and then replace its leaf labelled by $p_1 + p_k$ by an internal node whose left child is labelled by p_k and whose right child is labelled by p_1. This will be the optimal tree. Unwinding the recursion we find that the algorithm uses $O(n^2)$ time (but this can easily be improved down to $O(n \log n)$ with a careful use of data structures).

4 The Monge property and the algorithm

We now introduce the weighted directed graph G whose vertices are monotonic β-tuples of nonnegative integers in the range $[0 \ldots n-1]$. There is an edge between two vertices if and only they "overlap" in a $(\beta-1)$-tuple, precisely defined below.

Suppose $i_0 \leq i_1 \leq i_2 \leq \ldots i_{\beta-1} \leq i_\beta$. Let $u = (i_0, i_1, i_2, \ldots, i_{\beta-1})$ and $v = (i_1, i_2, \ldots, i_{\beta-1}, i_\beta)$. Then there is an edge from u to v if $u \neq v$, and furthermore, the weight of that edge is

$$\mathit{Weight}(u, v) = \mathit{EdgeCost}(i_0, i_1, \ldots, i_\beta) = S_{i_\beta + i_\alpha - i_0}$$

Observe that if (u, v) is an edge in G then the monotonicity of $(i_0, i_1, i_2, \ldots, i_{\beta-1}, i_\beta)$ guarantees that u is lexicographically smaller as a tuple than v. In other words the lexicographic ordering on the nodes is a topological ordering of the nodes of G; the existence of such a topological ordering implies that G is acyclic. Note that the β-tuple of zeros, $(0, \ldots 0)$, is a source. We refer to this node as the *initial node* of the graph. Note also that the β-tuple $(n-1, \ldots, n-1)$ is a sink; we refer to it as the *final node* of the graph.

For any vertex u in the graph, define $\mathit{cost}(u)$ to be the weight of a shortest (that is, least weight) path from the initial node to u.

Suppose we follow a path from the source to the sink and, after traversing an edge (u, v), output i_β, the final element of v. The sequence thus outputted is obviously a monotonic sequence terminating in the β-tuple $(n-1, n-1, \ldots, n-1)$ and from the definition of $\mathit{Weight}(u, v)$ the cost of the path is exactly the cost of the sequence. Similarly any monotonic sequence terminating in the β-tuple $(n-1, n-1, \ldots, n-1)$ corresponds to a unique path from source to sink in G.

In particular, given a tree T and $B = \mathit{sequence}(T)$ Lemma 1 implies that the cost of the path corresponding to B is exactly the same as $\mathit{cost}(T)$.

Example.
The tree T from Figure 3 has $B = \mathit{sequence}(T) = (1, 2, 2, 3, 3, 4, 4, 44)$ and its corresponding path in the graph G

$$\begin{aligned}
&(0,0,0,0,0) \xrightarrow{S_1} (0,0,0,0,1) \xrightarrow{S_2} (0,0,0,1,2) \\
&\xrightarrow{S_2} (0,0,1,2,2) \xrightarrow{S_4} (0,1,2,2,3) \xrightarrow{S_5} (1,2,2,3,3) \xrightarrow{S_5} (2,2,3,3,4) \\
&\xrightarrow{S_5} (2,3,3,4,4) \xrightarrow{S_5} (3,3,4,4,4) \xrightarrow{S_5} (3,4,4,4,4) \xrightarrow{S_5} (4,4,4,4,4)
\end{aligned}$$

The cost of this path and also of the tree T is

$$S_1 + 2 \cdot S_2 + S_4 + 6 \cdot S_5$$

The above observations can be restated as

Observation 2 *Assume T is a tree and $B = sequence(T)$. Then $cost(T) = cost(B)$ equals the cost of the path in G corresponding to B.*

The correctness theorem and the algorithm following it can can now be reformulated as follows:

Theorem 4.1 *The cost of a shortest path from the initial node to the final node is the same as the cost of a minimum cost tree. Furthermore given a minimum cost path a minimum-cost tree can be reconstructed from it in $O(n^2)$ time.*

Observe that G is acyclic and has $O(n^{\beta+1})$ edges. The standard dynamic-programming shortest path algorithm would therefore find a shortest path from the source to the sink, and hence a min-cost tree, in $O(n^{\beta+1})$ time. We now discuss how to find such a path in $O(n^\beta)$ time. Our algorithm obviously cannot construct the entire graph since it is too large. Instead we use the fact that, looked at in the right way, our problem possesses a Monge property.

A 2-dimensional matrix A is defined to be a *Monge matrix* [13] if for all i, j in range

$$A(i,j) + A(i+1,j+1) \leq A(i,j+1) + A(i+1,j) \tag{13}$$

Now let $\delta = (i_1, i_2, \ldots, i_{\beta-1})$ be any monotonic $(\beta-1)$-tuple of integers. For $0 \leq i \leq i_1$ and $i_{\beta-1} \leq j \leq n-1$, define

$$EdgeCost_\delta(i,j) = EdgeCost(i, i_1, \ldots, i_{\beta-1}, j) = S_{j+i_\alpha-i}$$

$$A_\delta(i,j) = cost(i, i_1, \ldots, i_{\beta-1}) + EdgeCost_\delta(i,j).$$

The important observation is that

Theorem 4.2 ((Monge-property theorem))
For fixed δ, the matrix A_δ is a two-dimensional Monge matrix.

Proof:
Let $\delta = (i_1, i_2, \ldots, i_{\beta-1})$. We prove Equation (13), where $A = A_\delta$. If the right hand side of Equation (13) is infinite, we are done. Otherwise, by the definitions of the S_k, and of A_δ, cancelling terms when possible, we have

$$A_\delta(i,j+1) + A_\delta(i+1,j) - A_\delta(i,j) - A_\delta(i+1,j+1) = p_{j+i_\alpha-i+1} - p_{j+i_\alpha-i} \geq 0$$

which completes the proof. □

A 2×2 matrix A is defined to be *monotone* if either $A_{11} \leq A_{12}$ or $A_{21} \geq A_{22}$. An $n \times m$ matrix A is defined to be *totally monotone* if every 2×2 submatrix of A is monotone. The SMAWK algorithm [3] takes as its input a function which computes the entries of an $n \times m$ totally monotone matrix A and gives as output a non-decreasing function f, where $1 \leq f(i) \leq m$ for $1 \leq i \leq n$, such that $A_{i,f(i)}$ is the minimum value of row i of A. The time complexity of the SMAWK algorithm is $O(n+m)$, provided that each computation of an A_{ij} takes constant time. Note that every Monge matrix is totally monotone so our matrices A_δ are totally monotone. This fact permits us to prove:

Theorem 4.3 ((Shortest-path theorem))
A shortest path from a source node to the sink node in G can be constructed in $O(n^\beta)$ time.

Proof:
The case where $\beta = 1$ requires an exceptional proof, because the proof below fails if the sequence δ is a 0-tuple. However, that case is already proved in [11]. Thus, we assume $\beta \geq 2$.

In this extended abstract we actually only show how to calculate the cost of the shortest path. Transforming this calculation into the *construction* of the actual path uses standard dynamic programming backtracking techniques that we will leave to the reader.

Our approach is actually to calculate the values of $cost(u)$ for *all* monotonic β-tuples $u = (i_0, i_1, \ldots, i_{\beta-1})$. In particular, this will calculate the value of $cost(n-1, n-1, \ldots, n-1)$ which is what is really required.

For fixed $\delta = (i_1, i_2, \ldots, i_{\beta-1})$ note that

$$\forall j \geq i_{\beta-1},\ cost(\delta, j) = \min\{A_\delta(i,j)\ :\ i \leq i_1\}$$

Also note that $A_\delta(i,j)$ can be calculated in constant time provided the values of $cost(i,\delta)$ is known. This means that, given a fixed δ, if the values of $cost(i,\delta)$ are already known for all i then the values of $cost(\delta, j)$ for *all* j can be calculated in

total time $O(n)$ using the SMAWK algorithm. We call this $O(n)$ step *processing* δ.

Our algorithm to calculate $cost(i_0, i_1, \ldots, i_{\beta-1})$ for all β-tuples is simply to process all of the $(\beta - 1)$ tuples in lexicographic order. Processing in this order ensures that at the time we process δ the values of $cost(i, \delta)$ are already known for all i.

Using the SMAWK algorithm it thus takes $O(n)$ time to process each of the $O(n^{\beta-1})$ $(\beta - 1)$-tuples so the entire algorithm uses $O(n^\beta)$ time as stated. □

Algorithm *Optimal_Tree_Construction*
  sequence construction phase:
    compute a shortest path π from source to sink in G;
    let B be the sequence corresponding to π;
  tree reconstruction phase:
    construct optimal tree T from B using
    recursive algorithm described following the
    Correctness Theorem
end of algorithm.

Theorem 4.4 ((main result))
We can construct a minimum cost lopsided tree in $O(n^\beta)$ time.

Proof:
If $\beta = 1$ use the basic Huffman encoding algorithm which runs in $O(n)$ time. Otherwise apply the algorithm *Optimal_Tree_Construction*. Theorem 4.3 tells us that B can be computed in $O(n^\beta)$ time.

The algorithm described following the Correctness Theorem for constructing an optimal tree from B runs in $O(n^2) = O(n^\beta)$ time completing the proof of the theorem.

□

We conclude by pointing out, without proof, that the algorithm

Optimal_Tree_Construction can be straightforwardly extended in two different directions:

Theorem 4.5
We can construct a minimum cost lopsided tree in $O(n \cdot \log^2 n)$ *time with* $O(n^{\beta-1})$ *processors of a PRAM.*

Theorem 4.6 ((height limited trees))
We can construct a minimum cost lopsided tree with height limited by L *in* $O(n^{\beta} \cdot L)$ *time.*

(A tree with height limited by L is one in which no node has depth greater than L.)

References

[1] Julia Abrahams, "Code and Parse Trees for Lossless Source Encoding," *Sequences'97,* (1997).

[2] Doris Altenkamp and Kurt Mehlhorn, "Codes: Unequal Probabilities, Unequal Letter Costs," *J. Assoc. Comput. Mach.* **27** (3) (July 1980), 412–427.

[3] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber, Geometric applications of a matrix-searching algorithm, *Algorithmica* **2** (1987), pp. 195–208.

[4] Siu-Ngan Choi and M. Golin, "Lopsided trees: Algorithms, Analyses and Applications," *Automata, Languages and Programming*, Proceedings of the 23rd International Colloquium on Automata, Languages, and Programming (ICALP 96).

[5] N. Cot, "A linear-time ordering procedure with applications to variable length encoding," *Proc. 8th Annual Princeton Conference on Information Sciences and Systems*, (1974), pp. 460–463.

[6] E. N. Gilbert, "Coding with Digits of Unequal Costs," *IEEE Trans. Inform. Theory,* **41** (1995).

[7] M. Golin and G. Rote, "A Dynamic Programming Algorithm for Constructing Optimal Prefix-Free Codes for Unequal Letter Costs," *Proceedings of the 22nd International Colloquium on Automata Languages and Programming (ICALP '95),* (July 1995) 256-267. Expanded version to appear in *IEEE Trans. Inform. Theory.*

[8] Sanjiv Kapoor and Edward Reingold, "Optimum Lopsided Binary Trees," *Journal of the Association for Computing Machinery* **36** (3) (July 1989), 573–590.

[9] R. M. Karp, "Minimum-Redundancy Coding for the Discrete Noiseless Channel," *IRE Transactions on Information Theory,* **7** (1961) 27-39.

[10] Abraham Lempel, Shimon Even, and Martin Cohen, "An Algorithm for Optimal Prefix Parsing of a Noiseless and Memoryless Channel," *IEEE Transactions on Information Theory*, **IT-19**(2) (March 1973), 208–214.

[11] L. L. Larmore, T. Przytycka, W. Rytter, Parallel computation of optimal alphabetic trees, SPAA93.

[12] K. Mehlhorn, "An Efficient Algorithm for Constructing Optimal Prefix Codes," *IEEE Trans. Inform. Theory* , **IT-26** (1980) 513-517.

[13] G. Monge, *Déblai et remblai*, Mémoires de l' Académie des Sciences, Paris, (1781) pp. 666-704.

[14] 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) (April 1975), 202–214,

[15] Serap A. Savari, "Some Notes on Varn Coding," *IEEE Transactions on Information Theory*, **40** (1) (Jan. 1994), 181–186.

[16] Robert Sedgewick, *Algorithms,* Addison-Wesley, Reading, Mass.. (1984).