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

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

Abstract

The *Monge property* emerges, sometimes quite mysteriously, in many optimization problems and usually leads to running time improvements of at least a linear factor. In this paper a novel application of *memory reduced dynamic programming* combined with a Monge property and the SMAWK algorithm is given for a variation of the *Huffman coding problem:* optimal prefix-free codes for unequal letter costs. 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. 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. It is still unknown if the problem is NP-complete and no polynomial time algorithm is known for general letter costs, despite a rich literature on the problem, e.g., [1, 7]. In practice the letter costs are small (e.g., Morse codes) and there are known polynomial time algorithms when α and β are integer constants. Using a novel approach we improve the complexity of the best known algorithm for this case by a factor of $O(n^2)$. One linear reduction is obtained by limiting the space of subproblems (reducing the dynamic programming tables); an additional linear factor is due to an employment of the algorithm of [3], which is whimsically known as the SMAWK algorithm, for finding all row minima of a totally monotone 2-dimensional matrix in linear time. For example, for 1,2 trees, the best previously known algorithm works in $O(n^4)$ time, and our improvement gives an $O(n^2)$ time algorithm.

The reduction of the space of subproblems is achieved by omitting some essential information related to partial subtrees. This is the most difficult and unorthodox part of the algorithm and it strongly affects the correctness proof which is unexpectedly complicated. The computed objects correspond to certain paths in a graph. However, due to the omission of essential information, some paths correspond to anomalous and undesirable objects. The main point is that *shortest* paths correspond to good objects. Several new deep and interesting combinatorial properties of lopsided trees were needed to provide the proof of correctness.

A parallel algorithm working in $O(n \log^2 n)$ time with $O(n^{\max(\beta-1,\alpha-1)})$ processors is also given, using a parallel version of the SMAWK algorithm. We also consider the case of *height limited* optimal lopsided trees.

*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.

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

1 Introduction

The problem of optimal prefix-free codes for unequal letter costs (and associated problem of optimal lopsided trees) is an old and hard classical problem whose algorithmics is not well understood despite a lot of ongoing reserach. 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. The problem of finding the minimum cost tree was first studied by Karp [9] in 1961 who solved the problem, (for the case when α, β are integers) by reduction to integer linear programming. 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 [14]; the restriction to the special case when all of the weights are equal, Cot [5], Perl Gary and Even [13], and Choi and Golin [4]; and approximating the optimal solution, Gilbert, [6]. Despite all of these attacks is is still, surprisingly, not even known whether the basic problem is polynomial time solvable. The only other algorithm for solving it is a dynamic programming one due to Golin and Rote [7] who give an $O(n^{\beta+2})$-time algorithm. That algorithm involved a relatively straightforward top-down construction of the optimal tree.

In this paper we describe 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, suprisingly, monotone-matrix techniques (the **Monge property**) can be taken advantage of. Combining these two attacks improves the running time by a factor of $O(n^2)$ down to $O(n^\beta)$. We contribute to combinatorics of lopsided trees and introduce a new crucial concept of **characteristic sequences**.

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

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

Now suppose we are given a sequence of 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 $Cost(T, P) = \sum_i p_i \cdot \text{depth}(T, v_i)$.

Given P, the problem that we wish to solve is to construct a labeled tree T that minimizes $Cost(T, P)$. 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. Notice that given any particular tree T the cost actually depends upon the labeling of the leafs of T, the cost being minimized when $p_1 \le p_2 \le \cdots \le p_n$ and $\text{depth}(T, v_1) \ge \text{depth}(T, v_2) \ge \cdots \ge \text{depth}(T, v_n)$. We therefore will always assume that the leaves of T are labeled in nonincreasing order of their depth. We will also assume that $p_1 \le p_2 \le \cdots \le p_n$.

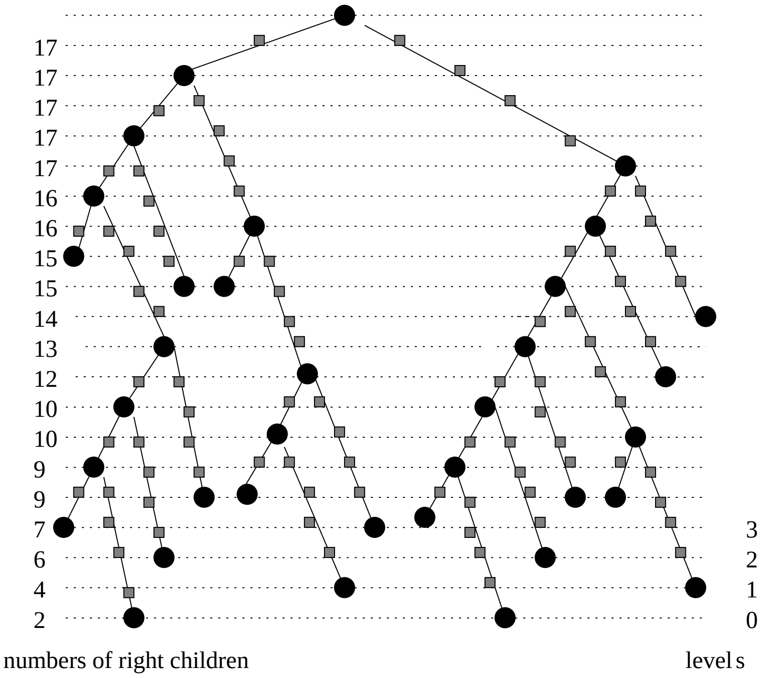

Figure 1: An example 2-5 tree T, special small rectangular nodes are inserted to illustrate the length of edges. We have $sequence(T) = B = (2, 4, 6, 7, 9, 9, 10, 10, 12, 13, 14, 15, 16, 16, 17, 17, 17, 17, 17)$.

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 $sequence(T)$ this is a vector $B = (b_0, b_1, \ldots, b_{d-1})$ in which b_i is number of right children below and including level i for $0 \le i < d$, where d is the height of the tree. (See Figure 1 for example.)

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_T defined by some tree T. B is said to be *monotonic* if $d \ge \beta$ and

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

A monotonic sequence B *terminates* in a β-tuple $(\gamma_\beta,\, \gamma_{\beta-1},\, \ldots,\, \gamma_1)$ if $\forall j,\, 0 \le j < \beta,\, b_{d-j} = \gamma_j$. For a monotonic sequence $B = b_0,\, b_1,\, \ldots,\, b_{d-1}$ define

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

If $i < 0$ or $i > n$ then $S_i = \infty$. For a tree T, denote by $\mathcal{F}_k = forest_k(T)$ 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 $forest_k(T)$. Note that $N_k(T)$, T a tree and $N_k(B)$, B a sequence are different things.

Lemma 1 *Let T be a lopsided tree and $B = sequence(T)$. Then*
(1) $Cost(T, P) = \sum_{0 \le k < \mathrm{depth}(T)} S_{N_k(T)}$, (2) $Cost(T) = Cost(B)$,

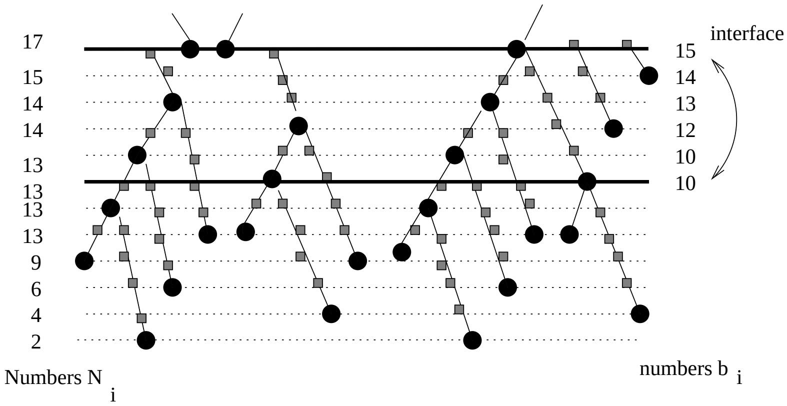

Figure 2: The bottom forest F_{11} of the tree T from Figure 1, its interface (information sufficient to compute optimally the upper part of the tree) is given by $(b_6, b_7, \dots, b_{11}) = (10, 10, 12, 13, 14, 15)$. The interface consists here of a 6-tuple, our main *trick* is to deal later only with 5 -tuples (in this case). The cost of the forest is $\sum_{i=0}^{11} S_{N_i}$, where N_i are shown on the left side. The cost of the whole tree equals the cost of the 19th forest (the tree without the root). Observe that $N_{11} = b_{11} + b_8 - b_6$.

(3) $\forall\ 0 \le i < d = depth(T), \quad N_i = b_i + b_{i-(\beta-\alpha)} - b_{i-\beta}$
where, by convention, $\forall\ j < 0,$ *we set* $b_j = 0$.

Proof. Item 1 can be proven by direct calculation. To prove 3, $\mathcal{F}_i$ is a forest, hence

$$N_i = \{u \in F_i \,:\, u \text{ is a leaf in } \mathcal{F}_i\} \tag{2}$$
$$= \text{Number of internal nodes in } \mathcal{F}_i + \text{ Number of trees in } \mathcal{F}_i \tag{3}$$

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 parent of some right child at level $k - \beta$. Thus

$$\text{Number of internal nodes in } \mathcal{F}_i = b_{i-\beta}. \tag{4}$$

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 parent is not in $\mathcal{F}_i$. Thus a right child 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 child at height k is a tree-root if and only if $i - \alpha < k \le i$. This may occurs if and only if the left child's right sibling 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

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

Combining (4) and (5) completes the proof of part 3. Part 2 follows from parts 1 and 3. $\square$

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.

Since not all sequences are legal this doesn't really add much. We will now see, though, that the minimal cost legal sequence is actually also a minimal cost *monotonic* sequence. While it might not be true that *every* min-cost monotonic sequence is legal we will see how, given a min-cost mosnotonic sequence, to transform it into a min-cost legal sequence (and thus an optimal tree). These facts, to be seen in the next sections, will provide a more effecient algorithm

3 Relation between minimum sequences and optimal trees

We consider first how to identify a left brother p_k of the smallest item p_1 if the sequence B is legal, *i.e.*, corresponds to a tree. Observe that the first leaf which is a left son starts at level $\beta - \alpha$ (see the left tree in Figure 3) and 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 first left-son leaf has index $FirstLeft(B)$, where $FirstLeft(B) = b_{\beta-\alpha-1} + 1$.

Intuitively, 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) labelling their parent (now a leaf) with $p_1 + p_k$, will also have to be an optimal tree for its new leaf set. Also, calculation shows that $cost(T) = cost(T') + \beta \cdot p_1 + \alpha \cdot p_k$. The rest of this section will be devoted to translating this intuition into facts about trees and sequences.

If $p_1,\ p_k$ are brothers 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 father is replaced by a leaf with weight $p_1 + p_k$ (see Figure 3). Denote also $unmerge(T', 1, k) = T$. (Although this operator is formally awkward it's meaning is intuitively clear). For the sequence $B = (b_0, b_1, \ldots b_d$ denote by $dec(B) = B' = (b_0 - 1, b_1 - 1, b_2 - 1, \ldots b_d - 1)$, where the leading zeros are deleted, but for convenience we index B' as the sequence of d numbers.

Denote by $delete(P, p_1, p_k)$ the sequence P with elements p_1 and p_k deleted and define

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

Straight calculation shows that

$$cost(unmerge(T', 1, k)) = cost(T') + \beta \cdot p_1 + \alpha \cdot p_k \tag{6}$$

Example. In Figure 3 we have: $T' = merge(T, 1, 3)$ 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 $cost(T) = cost(T') + 2p_4 + 5p_1$.

Lemma 3.1
(1) *Let B be a legal sequence and $k = FirstLeft(B)$. There is a tree T, such that $sequence(T) = B$ and p_k is the left brother of p_1 in T.*

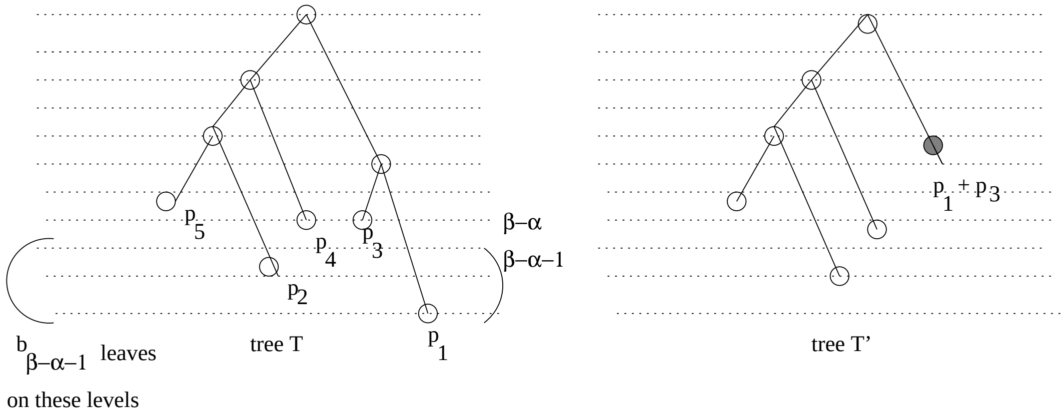

Figure 3: The correspondence between trees T, T' and their sequences

(2) *If T' is a tree corresponding to a sequence B' and to a sequence $P' = package\_merge(P, 1, k)$, then $B' = dec(B)$ for some legal sequence B and $cost(B') = cost(B) - \beta \cdot p_1 - \alpha \cdot p_k$.*

Now we need some combinatorial properties related to monotonic sequences $(b_0, b_1, \ldots, b_d)$ in which we delete/insert elements. 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 (we perform like in insertion sort). Denote by $Pref_t(\Gamma)$ the sum of the first t elements of a sequence Γ.

Lemma 3.2 (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 3.3 *If $j \geq k$ then*

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

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

$$Pref_{j-2}(\Gamma) = Pref_j(P) - p_1 - p_2 \tag{7}$$

(1) Apply point (1) of Lemma 3.2 and Equation 7 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 3.2 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 7. This completes the proof. □

Lemma 3.4 (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

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

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

$$\sum_{0 \le i < \beta - \alpha} term(i, B') = \sum_{0 \le i < \beta - \alpha} term(i, B) - (\beta - \alpha)p_1. \tag{8}$$

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 3.3, the difference between these values is at least $p_1 + p_k$. Hence

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

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 3.3 $term(i, B') \le term(i, B)$. Hence

$$\sum_{\beta \le i} term(i, B') \le \sum_{\beta \le i} term(i, B). \tag{10}$$

Combining (8), (9) and (10) yields the thesis. □

Theorem 3.5 (correctness)
Assume B is a minimum cost monotonic sequence for the sequence P of items. Then there is a tree T such that :
(1) *$cost(T) = cost(B)$,* **(2)** *$LeftLeaf(B)$ is the index of the left brother of p_1 in T,* **(3)** *$B' = dec(B)$ is a minimum cost sequence for $P' = merge\_package(P, 1, k)$.*

Proof: The proof is by induction with respect to the number n of items. Let $B' = dec(B)$. Take a minimum cost tree T' for P'. P' corresponds to $n - 1$ items, so by inductive argument $cost(T')$ equals the minimum cost of a sequence for P', in particular, due to key-lemma (3.4) we have

$$cost(T') = cost(B') \le cost(B) - \alpha \cdot p_k - \beta \cdot p_1 \tag{11}$$

Take $T = unmerge(T', 1, k)$. Then due to Equality (6) and Inequality (11) we have:

$$cost(T) = cost(T') + \alpha \cdot p_k + \beta \cdot p_1 \le cost(B).$$

Since $sequence(T)$ is also a sequence for P (of the same cost as T) and B is of minimum cost we should have equality here. Hence T is the required tree and this completes the proof. □

4 The Monge property and the algorithm

Define a weighted directed graph G whose vertices are monotonic β-tuples of nonnegative integers in the range [0 ... n-1]. There is an edge between two vertices if and only they "overlap" in a $(\beta-1)$-tuple, precisely defined below.
Suppose $i_0 \le i_1 \le i_2 \le \dots i_{\beta-1} \le i_\beta$. Let $u = (i_0, i_1, i_2, \dots, i_{\beta-1})$ and $v = (i_1, i_2, \dots, i_{\beta-1}, i_\beta)$. Then there is an edge from u to v if $u \ne v$, and furthermore, the weight of that edge is

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

Observe that the graph G is acyclic. Note that the β-tuple of zeros, $(0, \dots 0)$, is a source. We refer to this node as the *initial node* of the graph. Note also that the β-tuple $(n-1, \dots n-1)$, is a sink, which we refer to as the *final node* of the graph. For any vertex u in the graph, define $cost(u)$ to be the weight of a shortest (that is, least weight) path from the initial node to u.

Example.
For the tree T from Figure 3 the corresponding path in the graph G is:

$$\begin{array}{l}(0,0,0,0,0) \rightarrow^{S_1} (0,0,0,0,1) \rightarrow^{S_2} (0,0,0,1,2) \\ \rightarrow^{S_2} (0,0,1,2,2) \rightarrow^{S_4} (0,1,2,2,3) \rightarrow^{S_5} (1,2,2,3,3) \rightarrow^{S_5} (2,2,3,3,4) \\ \rightarrow^{S_5} (2,3,3,4,4) \rightarrow^{S_5} (3,3,4,4,4) \rightarrow^{S_5} (3,4,4,4,4) \rightarrow^{S_5} (4,4,4,4,4)\end{array}$$

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 correctness theorem can be reformulated as follows:

Theorem 4.1 *The cost of a shortest path from the initial node to the final node corresponds to the minimum cost of any tree.*

Fact 4.2 *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.*

Observe that G has $O(n^{\beta+1})$ edges, while we construct the algorithm finding a shortest path in only $O(n^\beta)$ time. Hence we do not explicitly construct G, and the graph should be thought of as a conceptual structure.

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

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

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

The matrix A_δ is a **Monge matrix** iff for all i, j in range

$$A_\delta(i,j) + A_\delta(i+1, j+1) \le A_\delta(i, j+1) + A_\delta(i+1, j) \tag{12}$$

Theorem 4.3 (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 12. If the right hand side of Equation 12 is infinite, we are done. Otherwise, by the definition of the S_k, and of A_δ, cancelling terms when possible, it is easy to see that

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

and this completes the proof. □

Theorem 4.4 (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 $\beta \geq 2$.

For fixed $\delta = (i_1, i_2, \ldots, i_{\beta-1})$, we show that the values of $cost(\delta, j) = \min\{A_\delta(i, j) \;:\; i \leq i_1\}$ for all j can be computed in $O(n)$ time, provided the values of $cost(i, \delta)$ are known for all i.

If $u = (i, i_1, i_2, \ldots, i_{\beta-1})$ and $v = (i_1, i_2, \ldots, i_{\beta-1}, j)$ then there is an edge in the graph from u to v. Furthermore $Weight(u, v) = EdgeCost_\delta(i, j)$.

We know that $cost(i_1, i_2, \ldots, i_{\beta-1}, j) = \min\{A_\delta(i, j) \;:\; i \leq i_1\}$.

Using the algorithm of [3] (whimsically known as the SMAWK algorithm), all row minima of A_δ can be computed in linear time, since every Monge matrix is totally monotone. □

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:
for $r = 1$ **to** $n - 1$ **do**
begin
combine the actually smallest item with $LeftLeaf(B)$;
$B := dec(B)$;
end
end of algorithm.

Theorem 4.5 (main result)
We can construct a minimum cost lopsided tree in $O(n^\beta)$ time.

Proof: We apply the algorithm *Optimal_Tree_Construction*. A shortest path B can be computed within required complexity due to Theorem 4.3. Then we can combine p_1 with p_k applying the function $LeftLeaf$ to B. The B' is a minimum sequence for $merge\_package(P, 1, k)$ and we inductively construct tree T' using B', as a part of tree T in which $p_1 + p_k$ is treated as a leaf. Observe that we use a similar approach as in standard Huffman coding, each time we know precisely which items are to be combined if the sequences B, $dec(B)$, $dec(dec(B))$ ... are known. □

Theorem 4.6
We can construct a minimum cost lopsided tree in $O(n \cdot \log^2 n)$ time with $O(n^{\beta-1})$ processors of the PRAM.

In a similar way we can proceed with height-limited lopsided trees, adding to each node of the graph G an extra information l about the actual level. Then we can prove (no new ideas are needed) the following result:

Theorem 4.7 (height limited trees)
We can construct a minimum cost lopsided tree with height limited by L in $O(n^\beta \cdot L)$ time.

References

[1] Julia Abrahams, "Code and Parse Trees for Lossless Source Encoding," *Manuscript,* (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.

[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. 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] 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,

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

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