A Generic Top-Down Dynamic-Programming Approach to Prefix-Free Coding*

Mordecai GOLIN† Xiaoming XU ‡ Jiajin YU§

Abstract

Given a probability distribution over a set of n words to be transmitted, the *Huffman Coding* problem is to find a minimal-cost prefix free code for transmitting those words. The basic Huffman coding problem can be solved in $O(n \log n)$ time but variations are more difficult. One of the standard techniques for solving these variations utilizes a top-down dynamic programming approach.

In this paper we show that this approach is amenable to dynamic programming speedup techniques, permitting a speedup of an order of magnitude for many algorithms in the literature for such variations as mixed radix, reserved length and one-ended coding. These speedups are immediate implications of a general structural property that permits batching together the calculation of many DP entries.

1 Introduction

Optimal prefix-free coding, or *Huffman coding*, is a standard compression technique. Consider an *encoding alphabet* $\Sigma = \{\sigma_1, \ldots, \sigma_r\}$. A *code* $W = \{w_1, w_2, \ldots, w_n\}$ is a set of *code words* $w_i \in \Sigma^*$. Code W is *prefix-free* if $\forall w, w' \in W$ w is not a prefix of w'. As an example, $\{01, 00, 100\}$ is a prefix-free code but $\{01, 00, 001\}$ is not, because 00 is a prefix of 001.

For $w \in \Sigma^*$, let $|w|$ devote the *length* of w, i.e., the number of characters in w. For example $|0101| = 4$.

The input to the problem is a discrete probability distribution $P = \{p_1, p_2, \ldots, p_n\}$, $\sum_i p_i = 1$, $\forall i$, $p_i \geq 0$.

*Work of all of the authors was partially supported by Hong Kong CERG grant 613507. Work of the 2nd and 3rd authors also partially supported by a grant from the National Natural Science Foundation of China (No. 60573025) and by the Shanghai Leading Academic Discipline Project, project number B114. The order of authors follows the international standard of alphabetic order of the last name. In China, where first-authorship is a particularly important aspect of a publication, the order of authors should be Xiaoming XU, Jiajin YU and Mordecai GOLIN.

†Dept of Computer Science & Engineering, Hong Kong UST, Hong Kong, China. *golin@cs.ust.hk*.

‡Dept of Computer Science & Engineering, IIPL, Shanghai Key Lab, Fudan University, Shanghai, China. *xuxiaoming@fudan.edu.cn*.

§Dept of Computer Science & Engineering, IIPL, Shanghai Key Lab, Fudan University, Shanghai, China. *jiajinyu@fudan.edu.cn*.

The output is a a prefix-free code $W = \{w_1, w_2, \ldots, w_n\}$ whose expected encoding length $\sum_{i=1}^n p_i|w_i|$ is minimized over all n word prefix-free codes. Formally set $Cost(W, P) = \sum_{i=1}^n w_i p_i$. Then

$$Cost(P) = \min_{\substack{W' \subseteq \Sigma^*, |W'|=n \\ W \text{ is prefix-free}}} Cost(W', P) \tag{1.1}$$

In [15], Huffman gave a classical $O(n \log n)$ time greedy algorithm for solving the binary case ($r = 2$) of this problem. Huffman also extended the algorithm to solve the general r-ary case with the same time bound. If the p_i's are given in sorted order, Huffman's algorithm can be improved to $O(rn)$ time [19].

The correctness of the Huffman algorithm, although easy to prove, is very strongly dependent upon properties of optimal prefix-free codes. Almost any extra constraint or generalization added to the problem description will invalidate the algorithm's correctness. Many such constraints/generalizations appear in the literature ([1] is a nice survey) and all require special purpose algorithms to address them.

Some examples of such prefix-free coding problems are *Length-limited* coding e.g, [16, 17, 3, 18], *Unequal-cost* coding e.g., [6, 13, 7, 12] *Mixed-radix* coding [10], *Reserved-length* coding [4], and *One-ended* coding [5, 8, 9],

The major observation is that all of the best algorithms known for these problems use some form of dynamic programming (DP) to build an optimal (min-cost) coding tree that corresponds to an optimal code.

These DPs primarily differ in whether they build the tree from the bottom-up or the top-down. The best algorithms for Length-limited and Unequal-cost coding use what essentially reduces to a bottom-up DP model combined with some DP-speedup techniques, e.g., Monge speedups using the SMAWK algorithm of [2] (see [7] for an example of this technique and [18] for a more sophisticated but more specialized speedup method); the best algorithms for Reserved-length and One-ended coding use a top-down DP approach. (Mixed-radix coding [10] uses a totally different DP approach described later)

Length-Limited and Unequal-Cost coding *could* be solved using a top-down approach but the bottom-up solutions are better for two reasons. The first is that

the bottom-up solutions use a more compact solution space than corresponding top-down ones would. This is due to the exploitation of some very problem-specific combinatorial structures of their corresponding optimal code trees. The second is that their bottom-up DPs turn out to have special properties, e.g., the Monge property, which enable speeding up the calculation of table entries. The top-down DPs used in the last two problems don't have such a compact representations and they also, before this paper, didn't seem to possess any special property that would lead to speedups.

The main result of this paper is a revisiting of the generic top-down DP approach for solving prefix-free coding problems. We will show that, in this setup, many natural coding problems will have an obvious and simple batching speedup. That is, we will be able to partition the DP table entries into smaller batches (groups) and exploit relationships between entries within a batch to fill in all of the entries in each batch in $O(1)$ amortized time per entry. This will enable speeding up known solutions to the last three problems by at least one factor of n. The interesting observation is that the *same* speedup technique works for all of these problems. Table 1 lists the speedups.

1.1 The problems We start by quickly recalling the standard correspondence between prefix-free codes and trees. Let $r = |\Sigma|$ be the size of the alphabet and consider an r-ary tree T in which the i^{th}edge leaving a node is labelled with character σ_i. Associate with node $v \in T$ the unique word read off walking down the path from the root to v. The set of words W associated with the leaves of T is prefix-free. Conversly, given a prefix-free code W one can build a tree whose leaves are exactly the nodes associated with the words of W. See Figure 1

Given tree T associated with code W, denote its leaves by $v_1, v_2, \ldots, v_n$ where v_i is the leaf associated with word w_i. Let $d(v_i)$ be the *depth* of v_i in T. By the correspondence, $d(v_i) = |w_i|$ so

$$Cost(T, P) = \sum_i d(v_i)p_i = \sum_i |w_i|p_i = Cost(W, P)$$

where $Cost(T, P)$ can be understood as the weighted external-path-length of T. So the prefix-free coding problem is equivalent to finding a tree with minimal external path length. For this reason, most algorithms for finding prefix-free codes are stated as tree algorithms.

We now quickly discuss the problems mentioned in the previous sections and their tree equivalents and then state our new results for these problems.

Mixed-Radix Coding:
In Mixed-Radix Coding the size of the encoding alphabet used depends upon the position of the character within the codeword. This corresponds to constructing a tree in which the *arity* (number of children) of an internal node depends upon the level of the node. That is, as part of the problem definition, we are given a sequence $t_0, t_1, \ldots$ of integers, $t_i \geq 2$ such that the maximum arity of a node on level i is t_i.

The coding version of the problem was motivated [10] by coding with side-channel information and the tree version by problems in multi-level data storage. Chu and Gill [10] solved this problem by introducing an alphabetic version of it and then solving a special case of the alphabetic version. Their algorithm runs in $O(n^4 \log n)$ time; we will improve this to $O(n^3)$.

Reserved-Length Coding:
Recall that $|w_i|$ is the length of the i^{th}codeword. In reserved-length coding there are specific restrictions as to the permitted values of $|w_i|$. There are two versions of this problem. In the first version, the *given-lengths case*, $\Lambda = \{\gamma_1, \gamma_2, \ldots, \gamma_g\}$ is given as part of the input and we must find a minimum-cost code such that $\forall i, |w_i| \in \Lambda$. This corresponds to building a min-cost tree in which all leaves are on levels in Λ.

In the second version, the *g-lengths case*, Λ is not given in advance. The restriction now is to find a minimum-cost code under the restriction that $|\Lambda| \leq g$, where Λ is the set of codeword lengths used. This corresponds to building a tree in which at most g levels may contain leaves.

Baer [4] introduces these problems in the context of fast decoding and used a top-down DP approach to solve the first one in $O(|\Lambda|n^3)$ time and the second one in $O(n^3 g^3 \log^g n)$ time . We will reduce these two cases, respectively, to $O(|\Lambda|n^2)$ and $O(n^2 g \log n)$ time.

One-Ended Coding:
In One-Ended coding the aim is to find a min-cost binary prefix-free code in which every word must end with a **1**. This corresponds to finding a min-cost tree in which only right leaves (leaves that are the right children of their parents) are labelled with the p_i and counted in the calculation of the cost.

One-Ended Coding was introduced by Berger and Yeung [5] in the context of self-synchronizing codes. Their algorithm ran in exponential time. This was later improved by De Santis, Capocelli and Persiano [8] to another exponential-time algorithm with a smaller exponential base. Chan and Golin [9] showed how to use a top-down DP to derive an $O(n^3)$ time algorithm. We will reduce this down to an $O(n^2)$ time one.

In Section 2 we introduce a new coding problem called *Generalized Mixed-Radix Coding*, develop a top-down DP approach for solving it and then speed it

Problem	Previous Best Result	This paper
Mixed Radix Coding	$O(n^4 \log n)$ [10]	$O(n^3)$
Reserved Length Coding: g specific lengths given	$O(gn^3)$ [4]	$O(gn^2)$
Reserved Length Coding: at most g lengths allowed	$O(g^3n^3 \log^g n)$ [4]	$O(gn^2 \log n)$
One-ended Coding	$O(n^3)$ [9]	$O(n^2)$

Table 1: A comparison of our new results with previous ones

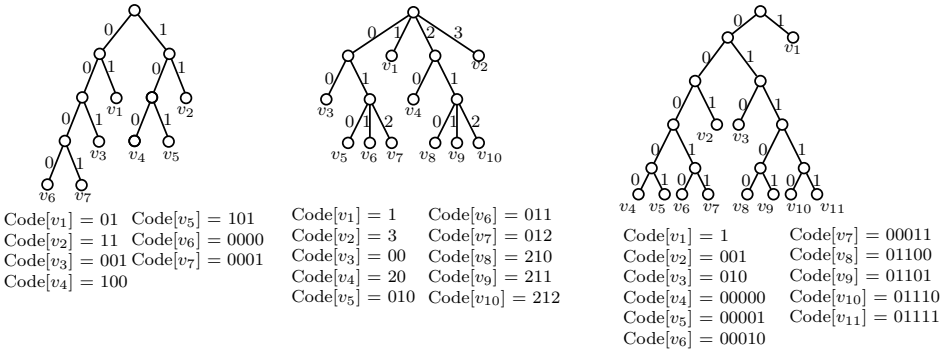

Figure 1: Examples of the code-tree correspondence. Codes are written below their corresponding tree. The leftmost figure is a standard binary tree. The middle is a Mixed-radix tree with level arities $(t_0, t_1, t_2) = (4, 2, 3)$. The rightmost is a Reserved-length tree with codewords only on levels $\Lambda = \{1, 3, 5\}$. Note that leaves in all the trees are labelled from top-to bottom

up by batching. In Section 3 we reduce both Mixed-Radix Coding and Reduced Length Coding to (multiple) applications of GMR and thus take adavantage of the DP speedup. In Section 4 we reduce the running time of One-Ended coding using an almost identical technique. Since the analysis of One-Ended coding is very similar to that of the GMR problem, we do not provide the details in this extended abstract.

2 The Top Down DP for Generalized Mixed-Radix Coding

We start by introducing the Generalized Mixed-Radix (GMR) problem, develop a top-down DP for solving it and then see how to speed it up.

In a *generalized mixed radix* tree, both the *arity* of an internal node v and the *length* of an edge leaving v depends on the *level* of v. Figure 2 illustrates these and other definitions in this section. More formally,

DEFINITION 1. *Given a sequence of arities* $R = (r_1, r_2, \ldots)$ *and a sequence of edge lengths* $C = (c_1, c_2 \ldots)$*, a generalized mixed radix (GMR) tree* T satisfying R*,* C *is a tree in which internal node* v *at level* $i-1$ *has at most* r_i *children and and the length of an edge from* v *to any of its children is* c_i*.*

We now distinguish between the level $\ell(v)$ of a node v, which is the number of edges from the root to v, and its depth, which is the weighted path length from the root to v. More formally,

DEFINITION 2. *The level of node* v *in tree* T *is the number of edges on the unique path from the root to* v *and will be denoted by* $\ell(v)$*. The level of tree* T *is*

$$\begin{aligned} \ell(T) &= \max\{\ell(v) \mid v \text{ is a node in } T\} \\ &= \max\{\ell(v) \mid v \text{ is a leaf in } T\}. \end{aligned}$$

The depth *of a node on level* i *of the tree will be the sum of the lengths of the edges on the path from the root to level* i*, i.e.,* $depth(v) = L(i) = \sum_{j=1}^{i} c_i$*. The depth of tree* T *will be* $depth(T) = L(\ell(T))$*.*

There is an obvious definition of cost in such trees.

DEFINITION 3. *Given* R, C *as above*

Let T *be any generalized mixed radix tree* T *for* R, C *with* n *leaves labeled* $v_1, v_2, \cdots, v_n$*. Then*

$$Cost(T) = \sum_i depth(v_i) \cdot p_i \tag{2.2}$$

The problem to be solved is, given P and R, C, find a min-cost tree T^* with n leaves, i.e., $Cost(T^*) = min\{Cost(T) \mid T \text{ has } n \text{ leaves}\}$

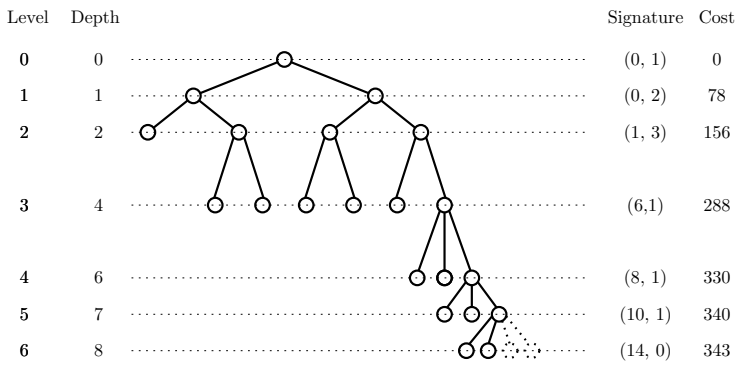

Figure 2: A generalized mixed radix tree with $R = (2, 2, 2, 3, 3, 4)$ and $C = (1, 1, 2, 2, 1, 1)$. Note that $n = 12$ but to make the tree full we needed to add two extra leavs on the bottom level. Also, the signatures and costs on level i are of the truncated tree containing the nodes on the first i levels. Costs are for $P = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$.

2.1 The Basic Top-Down Dynamic Program In this section we quickly describe the standard top-down DP formulation. Since variations of this formulation have been extensively used before for various coding problems, e.g. [13, 11, 9, 4], we only sketch the method but do not rigorously prove its correctness.

In what follows $P = \{p_1, p_2, \cdots, p_n\}$ with $p_1 \geq p_2 \geq \cdots \geq p_n \geq 0$ is given and fixed. The p_i can be arbitrary weights and are not required to sum to 1. The p_i sequence is implicitly padded so that, for $i > n$, $p_i = 0$. Finally, for $m \geq 0$, set $W_m = \sum_{i>m} p_i$.

We start with some standard simplifying assumptions about min-cost (optimal) trees T^*. We will show that there always exists at least one optimal tree satisfying these assumptions. Since our goal is to find any optimal tree, our search can be restricted to trees satisfying the assumptions.

In what follows, an internal node v, $\ell(v) = i$, in tree T will be *full* if and only if it has r_{i+1} children in T. Tree T will be *full* if all of its internal nodes are full.

Assumption 1: If $i < j$ then, in T^*, $depth(v_i) \leq depth(v_j)$.
If this was not true we could label v_i with p_j and v_j with p_i. The resulting tree has cost no greater than the original one so it remains optimal. We may therefore always implictly assume that leaf weights in trees are non-increasing in the level of the tree.

Assumption 2: There is a full tree T^* with the same cost as the optimal tree for n leaves. T^* has n' leaves where

$$\max(n, r_{\ell(T^*)}) \leq n' \leq n + r_{\ell(T^*)} - 1$$

This will be a consequence of the padding of P.

Let T be an optimal tree with *exactly* n leaves. Suppose $\ell(T) = \ell(v_n) = \ell$.

First note that all internal nodes v with $\ell(v) < \ell - 1$ are full. Otherwise we could add a new leaf child of v at level $\ell(v) + 1 < \ell(v_n)$ and label it with p_n, creating a tree with smaller cost than optimal tree T. Thus, the only non-full internal nodes in T are on level $\ell - 1$.

Next note that we may assume that *at most* one internal node at level $(\ell - 1)$ is not full; otherwise leaves on level i can be shifted to the left, so that all internal nodes on level-$(i - 1)$, except possibly the rightmost one, are full. Make this rightmost node full by adding an appropriate number leaves to it and call this new full tree T^*. Note that $\ell(T^*) = \ell(T) = \ell$ and $cost(T) = cost(T^*)$. This follows because P was padded by setting $p_i = 0$ for $i > n$. Let n' be the number of leaves in T^*.

By definition $n \leq n'$. Since T contains at least one internal node on level $\ell - 1$, T^* contains at least r_ℓ leaves on level $\ell - 1$. Thus $\max(n, r_\ell) \leq n'$

Furthermore, $n' \leq n + r_\ell - 1$ because T^* was created by adding at most $r_\ell - 1$ leaves to T.

Assumption 3: $\ell(T^*) \leq n$.
This will follow from the fact that we may assume that $\forall i, r_i \geq 2$.

Assumption 4: The cost of a tree is fully determined by its *leaf sequence*, i.e., the number of leaves on each level. No other structural properties need to be maintained.

This follows directly from the previous assumptions, i.e., the fullness of optimal trees. Although we will talk about constructing *trees* we will really be constructing the corresponding *leaf sequences,* e.g., sequences denoting how many leaves are on on each level. Since the cost of a tree is fully determined by its leaf sequence this does not cause any problems.

Suppose that T' is a tree with $n' > n$ leaves. Create T'' by pruning the deepest leaves from T'

one-by-one, until exactly n leaves remain. Then, by construction, T'' is a tree with exactly n leaves such that $cost(T'') \leq cost(T')$. This observation, Assumption 2 and Assumption 3, tell us that we can find the optimal tree for n leaves by first finding – for every $\ell \leq n$, and every n' satisfying $\max(n, r_\ell) \leq n' \leq n + r_\ell - 1$– the cost of the min-cost full tree of level at most ℓ with n' leaves. Then we take the minimum cost tree among all such trees and prune it until it has exactly n leaves. The resulting tree wil be the optimal tree for n leaves.

There are only $O(n^2)$ (ℓ, n') pairs that need to be examined; given their costs, finding the optimal pair requires only $O(n^2)$ time. (The subsequent pruning operation can easily be done in $O(n)$ time.) The hard part is, for each given (ℓ, n') pair, to find the costs of the appropriate min-cost full tree and then, if necessary, build it.

The intuition behind the solution is to build optimal trees top-down, starting with an initial tree – the root – building successively bigger trees level-by-level by making some nodes on the bottom level internal. Part of the specification of these intermediate *truncated trees* will be an explicit statement of the number of nodes on their bottom levels that will become internal when they are further expanded. The process ends when a tree whose bottom level contains no internal nodes is constructed.

Since we are only interested in constructing full trees and the truncation of a full tree up to any level is full, *this process may implicitly assume that every intermediate tree built is full.*

To transform this intuition into a dynamic program we will need to somehow encode the space of intermediate trees compactly and introduce an appropriate definition of *cost* for intermediate trees.

DEFINITION 4.
Tree T is an i-level tree *if all nodes $v \in T$ satisfy $\ell(v) \leq i$.*

If T is an i-level tree its i-level signature *is the ordered pair*

$$sig_i(T) = (m, b)$$

in which

$$\begin{aligned} m &= |\{v \in T \mid v \text{ is a leaf, } \ell(v) \leq i\}| \\ b &= |\{v \in T \mid v \text{ is an internal node } \ell(v) = i\}| \end{aligned}$$

In the above definition, "internal" means that the leaf at the bottom level is tagged as being made internal if the tree grows to its next level.

Let T be an i-level tree with $sig_i(T) = (m, b)$. The i-level partial cost *of T is*

$$Cost_i(T) = \sum_{t=1}^{m} depth(v_t) \cdot p_t + L(i) \cdot \sum_{t=m+1}^{n} p_t \quad (2.3)$$

Figure 2 illustrates this definition.

Consider the possible signatures that could occur. Suppose that T is an i-level truncation of some optimal tree T^* with $\ell(T^*) = \ell$ and $sig_i(T) = (m, b)$.

If T is a proper truncation of T^*, i.e, $b > 0$, then $\ell(T) < \ell$. Thus every labelled leaf in T is one of $v_1, \ldots, v_n$ in T^* and every one of the b nodes on the bottom level of T that is labelled as "internal" is the ancestor of one of $v_1, \ldots, v_n$ in T^*. Thus $m + b \leq n$.

If T is not a proper truncation of T^*, i.e., $T = T^*$, then $b = 0$ and, from Assumption 2, $\max(n, r_\ell) \leq m \leq n + r_\ell - 1$. This motivates

DEFINITION 5. *(m, b) is a* valid i-level signature *if*

- *If $b > 0$ then $m + b \leq n$.*
- *If $b = 0$ then $\max(n, r_i) \leq m \leq n + r_i - 1$.*

Obviously, the number of valid i-level signatures is $O(n^2)$.

We can now introduce the DP table.

DEFINITION 6. *Let (m, b) be a valid i-level signature. Set $OPT^i[m, b]$ to be the minimum i-level partial cost over all i-level trees T with signature (m, b). More precisely*

$$OPT^i[m, b] = \min \{Cost_i(T) \mid \exists T, T \text{ is a } i\text{-level full tree with } sig_i(T) = (m, b)\}$$

If no such tree exists, set $OPT^i[m, b] = \infty$.

If $t > n$ then $p_t = 0$. Thus, if T^* is an i-level full tree with $n' \geq n$ leaves then, by definition, $Cost_i(T^*) = \sum_{t=1}^{n} depth(v_t) \cdot p_t = Cost(T^*)$. So, $OPT^i[n', 0]$ will be the optimal actual cost of an i-level full tree with $n' \geq n$ leaves, which is what we want.

DEFINITION 7. *Let T' be an $(i-1)$-level tree with $sig_{i-1}(T') = (m', b')$.*
Expand T' to a full i-level tree by adding all $r_i b'$ nodes on level i.

For b satisfying $0 \leq b \leq b' r_i$, the b^{th} expansion *of T' is the i-level tree created by denoting b of these $r_i b'$ nodes as internal and making the remaining $b' r_i - b$ nodes into leaves. We denote this by*

$$T = Expand_i(T', b).$$

Note that $sig_i(T) = (m, b)$ where $m = m' + b' r_i - b$.

We now extend the definition of expansions to signatures

DEFINITION 8. *If (m', b'), (m, b) are valid signatures such that $0 \leq b \leq b' r_i$ and $m = m' + b' r_i - b$ we write*

$$(m', b') \xrightarrow{i} (m, b).$$

It is easy to prove the following by construction:

LEMMA 2.1. *Let T' be an $(i-1)$-level tree with $sig_i(T') = (m', b')$ and (m, b) such that $(m', b') \xrightarrow{i} (m, b)$. Then $T = Expand_i(T', b)$ is a well-defined i-level tree with $sig_i(T) = (m, b)$.*

This implies the following corollary, which is the basis of the correctness of the dynamic program.

COROLLARY 2.1. *Let $(m_0, b_0) = (0, 1)$ be the unique level-0 tree with internal root. Then the lemma implies that every sequence*

(2.4)
$$(m_0, b_0) \xrightarrow{1} (m_1, b_2) \xrightarrow{2} \cdots \xrightarrow{i-1} (m_{i-1}, b_{i-1}) \xrightarrow{i} (m_i, b_i)$$

corresponds to an i-level tree T with $sig_i(T) = (m_i, b_i)$ that can be constructed top-down from the root by following the appropriate expansions given by the sequence. In particular, if $(m_i, b_i) = (n', 0)$ then the constructed tree has exactly n' leaves.

LEMMA 2.2. *Let T' and T be, respectively, $i-1$ and i-level trees with $sig_{i-1}(T') = (m', b')$ and $sig_i(T) = (m, b)$. If $(m', b') \xrightarrow{i} (m, b)$ then*

$$Cost_i(T) = Cost_{i-1}(T') + c_i W_{m'}.$$

Proof: From Lemma 2.1, $T = Expand(T', b)$ so level i contains the leaves $v_{m'+1}, \ldots, v_m$ and

$$\begin{aligned} Cost_i(T) &= \sum_{t=1}^{m} depth(v_t) \cdot p_t + L(i) \cdot \sum_{m' < t \le n} p_t \\ &= \sum_{t=1}^{m'} depth(v_t) \cdot p_t \\ &\quad + (L(i-1) + c_i) \left(\sum_{m' < t \le m} p_t + \sum_{m < t \le n} p_t \right) \\ &= Cost_{i-1}(T') \\ &\quad + c_i \sum_{m' < t \le n} p_t = Cost_{i-1}(T') + c_i W_{m'} \end{aligned}$$

This tells us that the cost of the n'-leaf tree associated with sequence (2.4) can be calculated level by level to be $\sum_{t=1}^{i} c_t W_{m_t}$ where $m_i = n'$. Combining all of the above, we can now write a simple DP that models building optimal trees from the top-down.

LEMMA 2.3. *The optimal cost of an i-level tree with signature (m, b) satisfies*

(2.5)
$$OPT^i[m, b] = \min_{\{(m', b') \mid (m', b') \xrightarrow{i} (m, b)\}} \left\{ OPT^{i-1}[m', b'] + c_i W_{m'} \right\}.$$

Initial conditions are that $OPT^0[0, 1] = 0$ with all other entries being set to ∞.

The entries $OPT^i[m, b]$ only depend upon the entries $OPT^{i-1}[m, b]$ so the table can be filled in using the order $i = 1, 2, \ldots, n$.

For any given level i there are only $O(n^2)$ valid i-level signatures. From Definition 8, for every (m, b), there are only $O(n)$ signatures such that $(m', b') \xrightarrow{i} (m, b)$ So, for fixed i, filling in all of the $OPT^i[m, b]$ requires $O(n^3)$ time, with $O(n^4)$ total time needed to fill in all of the enties $OPT^i[m, b]$, $i \le n$.

We will now see how, for fixed i, to calculate the values $OPT^i[m, b]$ in only $O(n^2)$ time. This will, as promised, reduce the total running time for filling in all n levels of the table to to $O(n^3)$.

2.2 Batching for Speedup We now see how to fill in the DP entries in a faster way. We first need two more definitions.

DEFINITION 9. *For $1 < d$, define*

$$\begin{aligned} \mathcal{I}(d) &= \{(m, b) \mid m + b = d\}, \\ \mathcal{I}'_i(d) &= \{(m', b') \mid m' + b' r_i = d\}. \end{aligned}$$

For any fixed i and $(m, b) \in \mathcal{I}_i(d)$, the definition of $\xrightarrow{i}$ implies that

"$(m', b') \xrightarrow{i} (m, b)$"
if and only if
"$(m', b') \in \mathcal{I}'_i(d)$ with $b \le b' r_i$".

This immediately permits rewriting (2.5) as

LEMMA 2.4. *If $(m, b) \in \mathcal{I}(d)$ for some $d \le n$ and*

(2.6)
$$OPT^i[m, b] = \min_{\substack{(m', b') \in \mathcal{I}'_i(d) \\ b \le b' r_i}} \left\{ OPT^{i-1}[m', b'] + c_i W_{m'} \right\}.$$

We now claim that, for fixed $d \le n$, the calculation of the values $OPT^i[m, b]$ for *all* $(m, b) \in \mathcal{I}(d)$ can be batched together in in $O(d) = O(n)$ time, i.e., in amortized $O(1)$ time per entry.

For fixed $d \le n$ suppose $(m', b') \in \mathcal{I}'_i(d)$, i.e., $m' + b' r_i = d$. This implies $b' \le \lfloor d / r_i \rfloor$. $\forall 0 \le b' \le \lfloor d / r_i \rfloor$, set

$$\begin{aligned} \gamma(b') &= OPT^{i-1}[m',\, b'] + c_i W_{m'} \\ &= OPT^{i-1}[d - b' r_i,\, b'] + c_i W_{d - r_i b'}. \end{aligned}$$

Note that all of the $\gamma(b')$ can be calculated in $O(d)$ total time (we assume that the W_m had been previously precalculated in $O(n)$ time and stored for later use). Equation (2.6) just says that for $(m, b) \in \mathcal{I}(d)$,

(2.7)
$$OPT^i[m, b] = \min \left\{ \gamma(b') \;\middle|\; (b / r_i) \le b' \le \lfloor d / r_i \rfloor \right\}.$$

$(m, b) \in \mathcal{I}(d)$ implies $m = d - b$. Since $b \le r_i \lfloor d/r_i \rfloor$, $m \ge t = d - r_i \lfloor d/r_i \rfloor = d \bmod r_i$. So

$$\mathcal{I}(d) = \{ X_m \mid m = t, t+1, \ldots, d-1 \}$$

where

$$X_m = \big(m,\, d-m\big).$$

Then (2.7) can be rewritten as

(2.8)

$$OPT(X_m) = \min\left\{\gamma(b') \;\middle|\; (d-m)/r_i \le b' \le \lfloor d/r_i \rfloor\right\}$$

This immediately yields

$$OPT[X_t] = \gamma\left(\lfloor d/r_i \rfloor\right).$$

Furthermore, $\forall m > t$,

if $r_i \not| (d-m)$

$$OPT[X_m] = OPT[X_{m-1}]$$

else if $r_i | (d-m)$

$$OPT[X_m] = \min\left(OPT[X_{m-1}], \gamma\left((d-m)/r_i\right)\right)$$

Thus, we can calculate all of the $OPT[X_m]$ for $X_m \in \mathcal{I}(d)$ in $O(d)$ time by working in the order $m = t, t+1, \ldots, d$.

For fixed i, to fill in all valid signatures we start by implicitly setting all entries to ∞ and then iterate for $d = 1, 2, 3, \ldots$, for each value of d using $O(d)$ time to calculate all of the entries $OPT^i[m, b]$ with $m + b = d$. The question is where to stop the iteration.

From Definition 5 we know that if $b > 0$ then $d = m+b \le n$ while if $b = 0$ then $\max(n, r_i) \le m \le n + r_i - 1$. There are now two cases.

$r_i \le n$: Then $m+b < 2n$ so stop the iteration at $d = 2n-1$. All of the valid entries will have been filled in using $\sum_{d<2n} O(d) = O(n^2)$ time.

$r_i > n$: In this case if $b = 0$ then $r_i \le m \le r_i + n - 1$ and $(m', b') \xrightarrow{i} (m, 0)$ implies that either $(m', b') = (m, b)$ or $(m', b') = (m - r_i, 1)$, i.e., $(m, 0)$ has only two possible predecessors. We can therefore fill in the full table in two phases. In the first, fill in all valid entries $OPT^i[m, b]$ with $m + b = d$ for $d = 2, 3, \ldots, n$ in $O(n^2)$ time. In the second, fill in the all valid entries of the form $OPT^i[m, 0]$ with $r_i \le m \le r_i + n - 1$ in $O(n)$ time, by checking the two predecessors of each possible entry.

3 Mixed-Radix Coding and Reserved-Length Coding

We now see how to solve both Mixed-Radix Coding and Reserved-Length Coding via the GMR approach.

3.1 Mixed-Radix Coding Chu and Gill's Mixed-Radix Coding problem [10] is exactly the GMR problem restricted to all of the edge costs being identically 1, i.e., $\forall i, c_i = 1$. The algorithm in the previous section solves this in $O(n^3)$ time, improving upon the $O(n^4 \log n)$ time of [10].

3.2 Reserved-Length Coding In the *reserved-length coding problem*, there are restrictions as to permissible codeword lengths. In the tree version of the problem, these become restrictions on the allowable levels on which leaves can appear. More formally. let

$$Level(T) = \{\ell(v) \mid v \text{ a leaf in } T\}.$$

There are two versions of the problem.

In the first version of the problem, the *given-lengths* case, a set of integers $\Lambda = \{\gamma_1, \ldots, \gamma_g\}$ is given (w.l.o.g, $0 < \gamma_1 < \gamma_1 < \gamma_2 < \cdots < \gamma_g$; we may also add $\gamma_0 = 0$ since, if $n > 1$, the root will never be internal) and we are asked to find a minimum-cost r-ary tree among all trees with $Level(T) \subseteq \Lambda$.

In the second version of the problem, the *g-lengths* case, an integer g is given and we are asked to find a minimum-cost r-ary tree among all trees with $|Level(T)| \le g$.

3.2.1 The Given-Lengths Case Let T be an optimal r-ary tree for given P and $\Lambda = \{\gamma_1, \ldots, \gamma_g\}$.

All leaves in T are at a level γ_i for some $\gamma_i \in \Lambda$. Consider any internal node v at level γ_{i-1}. It has no leaf descendants at any level ℓ with $\gamma_{i-1} < \ell < \gamma_i$. We may therefore assume that all of its $r^{\gamma_i - \gamma_{i-1}}$ descendants at level γ_i are in the tree, i.e., that v is the root of a *complete* subtree of height $\gamma_i - \gamma_{i-1}$.

We may therefore create a new tree T' as follows. The root of T' corresponds to the root of T. Nodes in T' at level i are in 1-1 correspondence with nodes at level γ_i in T and there is an edge from node u on level $i-1$ to node v on level i in T' if u is the level γ_{i-1} ancestor of v in T. See Figure 3 for an illustration.

By construction, T' is a GMR tree with $r_i = r^{\gamma_i - \gamma_{i-1}}$ and $c_i = \gamma_i - \gamma_{i-1}$. Furtheremore, the construction can be reversed, with any generalized mixed arity tree with these parameters being transformable into a restricted length tree with the same cost for the given Λ.

Since there are at most g levels, our generic GMR algorithm solves this problem in $O(gn^2)$ time, improving upon the $O(gn^3)$ algorithm of [4] .

3.2.2 The g-lengths case If the levels on which leaves appeared were known to be $\Lambda = \{\gamma_1, \ldots, \gamma_g\}$ then this is exactly the given-lengths case, which as seen, is equivalent to building an optimal GMR tree with

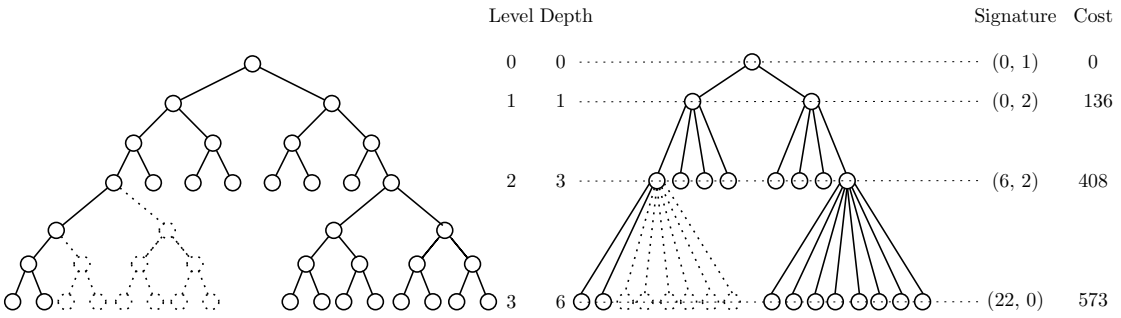

Figure 3: A Reserved-Length tree (left) with $\Lambda = \{1, 3, 6\}$, $n = 16$ leaves and the corresponding GMR tree (right). Note that even though they are allowed, there are no leaves on level 1. All internal nodes on level 3 should have 8 descendents on level 6 so 6 leaves were added to make the tree full. Signatures and costs for the GMR tree at level i are for the truncated tree containing the first i levels. Costs are for $P = \{1, 2, \ldots, 16\}$.

$(r_i, c_i) = (r^{\gamma_i - \gamma_{i-1}}, \gamma_i - \gamma_{i-1})$. The added complication here is to *guess* Λ.

This is equivalent to the problem of building a slightly generalized version of a GMR tree in which, instead of guessing Λ, we instead, at each level i, guess the pair $(r_i, c_i) = (r^t, t)$ for $t \geq 1$. Any such pair is allowable but once t is chosen, it applies to all nodes on level i. Furthermore, since the tree only needs n leaves we may assume $r^t \leq rn$ and thus may restrict $t \leq 1 + \log_r n$.

This motivates slightly modifying the GMR model to allow *choices* of (r_i, c_i).

Recall that the original definition of GMR specifies arities $R = (r_1, r_2, \ldots)$ and edge lengths $C = (c_1, c_2, \ldots)$. We now replace these with $\overline{R} = (\overline{r}_1, \overline{r}_2, \ldots)$ and edge lengths $\overline{C} = (\overline{c}_1, \overline{c}_2, \ldots)$ where

$$\overline{r}_i = \{r_{i,1}, r_{i,2}, \ldots, r_{i,\Delta_i}\}, \quad \overline{c}_i = \{c_{i,1}, c_{i,2}, \ldots, c_{i,\Delta_i}\},$$

are *sets* of Δ_i possibilities for level i. A *permissible tree* is a GMR tree for some sequence $R = (r_{1,j_1}, r_{2j_2}, \ldots)$ and edge lengths $C = (c_{1,j_1}, c_{2,j_2}, \ldots)$ where $\forall i,\ 1 \leq j_i \leq \Delta_i$.

Given P, an optimal tree would now be a min-cost permissible tree for the given $\overline{R}$, $\overline{C}$.

The discussion above tells us that to solve the g-lengths problem, it is only necessary is solve this new generalized version of the GMR problem to construct a minimum-cost g-level tree where, for every i, $\overline{r}_i$ and $\overline{c}_i$ are the sets defined by

$$\Delta_i = 2 + \lfloor \log_r n \rfloor,$$

and

$$\forall 1 \leq j \leq \Delta_i, \quad r_{i,j} = r^{j-1},\ c_{i,j} = j - 1.$$

The modifications to the definitions and algorithms are straightforward. Signatures are defined the same way as before. Definition 5 of valid signatures needs to be modified to allow $r_{i,j}$.

DEFINITION 10. *(m, b) is a* valid i-level signature *if*

- *If $b > 0$ then $m + b \leq n$.*
- *If $b = 0$ then $\exists j$, such that $\max(n, r_{i,j}) \leq m \leq n + r_{i,j} - 1$*

Note that the number of valid i-level signatures is $O(n^2 + n\Delta_i)$.
Definition 8 also needs to be slightly generalized:

DEFINITION 11. *If (m', b'), (m, b) are, respectively, valid $(i-1)$ and i-level signatures such that $0 \leq b \leq b' r_{i,j}$ and $m = m' + b' r_{i,j} - b$, we write*

$$(m', b) \xrightarrow{i,j} (m, b).$$

We now, similarly as before, define

$$OPT^i[m, b] = min\,\{Cost_i(T) \mid \exists T, \text{T is a i-level tree with } sig_i(T) = (m, b)\}$$

The only major difference is in the analogue of Lemma 2.3, which gives the DP for calculating $OPT^i[m, b]$. This now needs to be split into two phases; the first calculates, for every j, the optimimum value of $OPT^i[m, b]$ assuming that $(r_i, c_i) = (r_{i,j}, c_{i,j})$. The second takes the minimum of this value over all j. More specifically:

LEMMA 3.1. *For $1 \leq j \leq \Delta_{i-1}$, set*

(3.9)
$$OPT^{i,j}[m, b] = \min_{\{(m',b') \mid (m',b') \xrightarrow{i,j} (m,b)\}} \left\{ OPT^{i-1}[m', b'] + c_{i,j} W_{m'} \right\}.$$

The optimal cost of an i-level tree with signature (m, b) then satisfies

$$OPT^i[m, b] = \min_{1 \leq j \leq \Delta_i} OPT^{i,j}[m, b] \tag{3.10}$$

Initial conditions are that $OPT^0(0, 1) = 0$ with all other entries being set to ∞.

Given the values $OPT^{i-1}[m,b]$, the batching speedup of Subsection 2.2 now permits, for any fixed j, calculating all of the values $OPT^{i,j}[m,b]$ in $O(n^2)$ time. Thus, all of the values $OPT^i[m,b]$ can be calculated in $O(\Delta_i n^2)$ time.

The total amount of work required for calculating $OPT^g[m,b]$ from scratch is then $O\left(n^2 \sum_{i=0}^{g} \Delta_i\right)$.

In the g-lengths problem, $\Delta_i = O(\log_r n)$ so the total runnnig time for solving the g-lengths problem is $O(gn^2 \log n)$, improving the $O(n^2 g^2 \log^{g-1} n)$ running time of the algorithm in [4].

4 One-Ended Coding

We now consider the problem of constructing minimum-cost binary prefix-free codes having the property that each codeword ends with a "**1**". The original algorithms [5, 8] for this problem were exponential. [9] presented a top-down DP running in $O(n^3)$ time. Using the batched speedup technique developed in Subsection 2.2 we can develop a modified top-down DP that reduces the running time to $O(n^2)$. The DP and its solution are very similar to those developed for the GMR algorithm so we only sketch them here. Full details can be found in the full version of this paper posted in the e-server archive [14].

As in Section 2, the algorithm will find a min-cost coding tree. We must first modify the code-tree correspondence to reflect the new 1-ended requirement. Assume that a left edge is labelled with a '0' and a right edge with a '1'. A node is a 0-node (1-node) if the edge connecting it to its parent is labelled by a 0 (1). See Figure 4. We will extend this naturally to 0-leaves and 1-leaves, and 0-internal nodes and 1-internal nodes.

To reflect the 1-ended restriction on the codes, only 1 leaves will be labelled with probabilities from $P = \{p_1, \ldots, p_n\}$. Let v_i be the 1-leaf in tree T labelled with p_i. Then we may still write $Cost(T) = \sum_i d(v_i)p_i$. As before, we pad P so that if $i > n$ then $p_i = 0$. Also, as before, we assume that the tree is full, i.e., that every internal node has two children. A final assumption is that *all* 0-leaves will be on the bottom level of the tree. Although this might increase the number of 1-leaves in the tree, these trees will be labelled with p_i, $i > n$, so they will carry zero cost. A node in T^* will be *good* if it's a 1-leaf that gets labelled with one of $p_1, \ldots, p_n$. It will be *bad* if it is a 0-node or a 1-node that doesn't get lablelled with one of $p_1, \ldots, p_n$.

We will build trees T^* from the top-down, level-by-level, At step i our current tree T will be the first i levels of T^*. We will identify in T the the number of nodes that are good leaves and the number of nodes, on T's bottom level, that are bad. Note that if $i < \ell(T^*)$ then, in the next expansion of level, *all* bad nodes on the current bottom level will become internal, each one contributing one new 1-node and one new 0-node on the new bottom level. If $i = \ell(T^*)$ then $T = T^*$ and the bad nodes are extra (those that get labelled with p_j, $j > n$) 1-leaves and 0-leaves on level i.

This motivates modifying the definition of signatures and costs:

Definition 12.
If T is an i-level tree its i-level signature *is the ordered pair*

$$sig_i(T) = (m, b)$$

in which

$$\begin{aligned} m &= |\{v \in T \mid v \text{ is a good 1-leaf, } \ell(v) \le i\}| \\ b &= |\{v \in T \mid v \text{ is a bad node, } \ell(v) = i\}| \end{aligned}$$

In the above definition, b is counting leaves in T that, if the tree is expanded one level further, will become internal nodes. Let T be the starting (0-level) tree containing only the root. Since the root will always be expanded, it is bad, so $sig_0(T) = (0, 1)$. This will later be the starting point of our dynamic program.

Let T be an i-level tree with $sig_i(T) = (m, b)$. The i-level partial cost *of T is*

$$Cost_i(T) = \sum_{t=1}^{m} depth(v_t) \cdot p_t + i \cdot \sum_{t=m+1}^{n} p_t \tag{4.11}$$

It is not hard to work out that the signatures of all interesting optimal trees and their truncations must satisfy $m \le n$ and $b \le 2n-1$; we will call such signatures *valid*. The analogue to Definition 8 is then

Definition 13. *If (m', b'), (m, b) are signatures that satisfy*

$$m' \le m \le n, \quad m = m' + 2b' - b, \quad 1 \le b' \le b \le 2b'.$$

we write

$$(m', b) \to (m, b).$$

As in Lemma 2.1, $(m', b) \to (m, b)$ corresponds to an tree with signature (m', b') *expanding* one level to a tree with signature (m, b).

$OPT[m, b]$ will be the minimum cost of a tree with signature (m, b). Our goal will be to find the minimum value of of $OPT[n, b]$ (an corresponding tree) where $1 \le b \le 2n - 1$.

The preceding definitions immediately lead to a DP for solving this problem. The naive algorithm for filling in the DP's size $O(n^2)$ table would require $O(n^3)$ time but a batching argument exremely similar to that in Subsection 2.2 permits filling in the table in $O(n^2)$ time. As mentioned, full details are provided in the full version of this paper posted in the e-server archive [14].

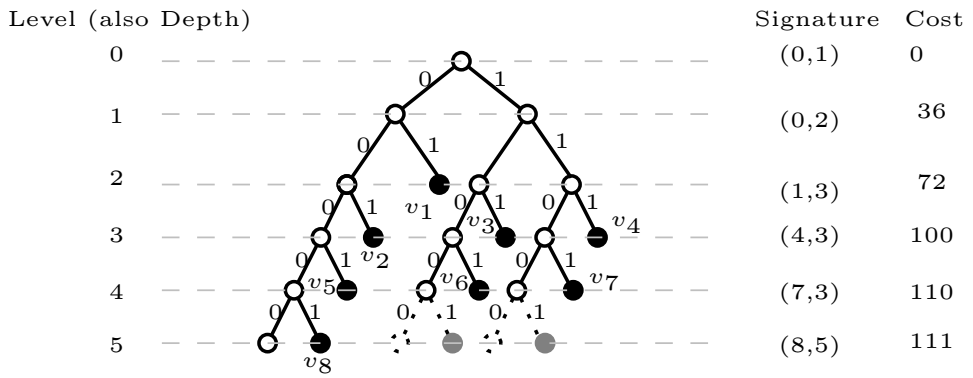

Figure 4: A One-Ended Coding tree for $n = 8$ leaves. Note that only 1-nodes are labelled leaves. On the bottom level, the two dotted 0-leaves and their two grey 1-leaf siblings are 'bad' nodes added to make the tree full. Signatures and Costs at level i are those of the (truncated) subtree containing the first i levels of the tree. Costs are calculated for $P = \{1, 2, 3, 4, 5, 6, 7, 8\}$.

References

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

[2] Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987.

[3] Alok Aggarwal, Baruch Schieber, and Takeshi Tokuyama. Finding a minimum-weight k-link path in graphs with the concave Monge property and applications. *Discrete and Computational Geometry*, 12:263–280, 1994.

[4] M.B. Baer. Reserved-length prefix coding. In *Proceedings of the 2008 IEEE International Symposium on Information Theory*, pages 2469–2473, July 2008.

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

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

[7] P. Bradford, M. Golin, L. L. Larmore, and W. Rytter. Optimal prefix-free codes for unequal letter costs and dynamic programming with the monge property. *Journal of Algorithms*, 42:277–303, 2002.

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

[9] Sze-Lok Chan and Mordecai J. Golin. A dynamic programming algorithm for constructing optimal "1"-ended binary prefix-free codes. *IEEE Transactions on Information Theory*, 46(4):1637–1644, 2000.

[10] Ke-Chiang Chu and John Gill. Mixed-radix Huffman codes. In Ricardo Baeza-Yates and Udi Manber, editors, *Computer science: research and applications*, pages 209–218, New York, NY, USA, 1992. Plenum Press.

[11] Shlomi Dolev, Ephraim Korach, and Dmitry Yukelson. The sound of silence: Guessing games for saving energy in mobile environment. In *Proceedings of INFOCOM'99*, pages 768–775, 1999.

[12] Sorina Dumitrescu. Faster algorithm for designing optimal prefix-free codes with unequal letter costs. *Fundamenta Informaticae*, 73(1-2):107–117, 2006.

[13] 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.

[14] Mordecai Golin, Xiaoming Xu, and Jiajin Yu. A generic top-down dynamic-programming approach to prefix-free coding (full version). *arXiv e-print server*, http://arxiv.org/abs/0809.4577, 2008.

[15] David A. Huffman. A method for the construction of minimum-redundancy codes. In *Proceedings of the Institution of Radio Engineers*, volume 40, pages 1098–1101, 1952.

[16] Richard M. Karp. Minimum-redundancy coding for the discrete noiseless channel. *IEEE Transactions on Information Theory*, 7(1):27–38, 1961.

[17] Lawrence L. Larmore and Daniel S. Hirschberg. A fast algorithm for optimal length-limited Huffman codes. *Journal of the ACM*, 37(3):464–473, 1990.

[18] Baruch Schieber. Computing a minimum weight k-link path in graphs with the concave Monge property. *Journal of Algorithms*, 29(2):204–222, 1998.

[19] Jan van Leeuwen. On the construction of huffman trees. In *Proceedings of the 3rd International Colloquium on Automata, Languages and Programming*, pages 382–410, 1976.