Twenty (or so) Questions: D-ary Length-Bounded Prefix Coding

Michael B. Baer, *Member, IEEE*

***Abstract*— The Huffman algorithm efficiently finds an optimal prefix code given a probability mass function. However, some applications call for restrictions on feasible codes. Length-limited prefix coding restricts the set of codes to those for which none of the n codewords is longer than a given length, $l_{\max}$. This paper generalizes two algorithms used for length-limited prefix coding, without increasing complexity, in order to introduce a minimum codeword length constraint $l_{\min}$ and to be applicable to both binary and nonbinary codes. Previously, nonbinary cases needed a slower dynamic programming approach, and — although useful in limiting memory usage of multiple codes — the minimum length constraint was not optimized for. These extensions have various applications including fast decompression, context-based coding, and a modified version of the game "Twenty Questions." This paper also uses them to solve the problem of finding an optimal code with limited fringe, that is, finding the best code among codes with a maximum difference between the longest and shortest codewords. The previously proposed method for solving this problem was nonpolynomial time.**

I. Introduction

The parlor game best known as "Twenty Questions" has a long history and a broad appeal. It was used to advance the plot of Charles Dickens' *A Christmas Carol* [1], in which it is called "Yes and No," and it was used to explain information theory in Alfréd Rényi's *A Diary on Information Theory* [2], in which it is called "Bar-kochba." The two-person game begins with an answerer thinking up an object and then being asked a series of questions about the object by a questioner. These questions must be answered either "yes" or "no." Usually the questioner can ask at most twenty questions, and the winner is determined by whether or not the questioner can sufficiently surmise the object from these questions.

Many variants of the game exist — both in name and in rules. A recent popular variant replaces the questioner with an electronic device [3]. The answerer can answer the device's questions with one of four answers — "yes," "no," "sometimes," and "unknown." The game also differs from the traditional game in that the device often needs to ask more than twenty questions. If the device needs to ask more than the customary twenty questions, the answerer can view this as a partial victory, since the device has not answered correctly given the initial twenty. However, the device eventually gives up after 25 questions if it cannot guess the questioner's object.

This material was presented in part at the IEEE International Symposium on Information Theory, Nice, France, June 2007

The author is with Ocarina Networks, Inc., 42 Airport Parkway, San Jose, CA 95110-1009 USA (e-mail: calbear@ieee.org).

Consider a short example of such a series of questions, with only "yes," "no," and "sometimes" as possible answers. The object to guess is one of the seven Newtonian colors [4], which we choose to enumerate as follows:

1) Green (G)
2) Yellow (Y)
3) Red (R)
4) Orange (O)
5) Indigo (I)
6) Violet (V)
7) Blue (B).

A first question we ask might be, "Is the color seen as a warm color?" If the answer is "sometimes," the color is green. If it is "yes," it is one of colors 2 through 4. If so, we then ask, "Is the color considered primary?" "Sometimes" implies yellow, "yes" implies red, and "no" implies orange. If the color is not warm, it is one of colors 5 through 7, and we ask whether the color is considered purple, a different question than the one for colors 2 through 4. "Sometimes" implies indigo, "yes" implies violet, and "no" implies blue. Thus we can distinguish the seven colors with an average of $2 - p_1$ questions if p_1 is the probability that color in question is green.

This series of questions is expressible using code tree notation, e.g., [5], in which a tree is formed with each child split from its parent according to the corresponding output symbol, i.e., the answer of the corresponding question. A code tree corresponding to the above series of questions is shown in Fig. 1, where a left branch means "sometimes," a middle branch means "yes," and a right branch means "no." The number of answers possible is referred to by the constant D and the tree is a D-ary tree. In this case, $D = 3$ and the code tree is ternary. The number of outputs, $n = 7$, is the number of colors.

The analogous problem in prefix coding is as follows: A source (the answerer) emits input symbols (objects) drawn from the alphabet $\mathcal{X} = \{1, 2, \ldots, n\}$, where n is an integer. Symbol i has probability p_i, thus defining probability vector $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)$. Only possible symbols are considered for coding and these are sorted in decreasing order of probability; thus $p_i > 0$ and $p_i \leq p_j$ for every $i > j$ such that $i, j \in \mathcal{X}$. (Since sorting is only $O(n \log n)$ time and $O(n)$ space, this can be assumed without loss of generality.) Each input symbol is encoded into a codeword composed of output symbols of the D-ary alphabet $\{0, 1, \ldots, D-1\}$. (In the example of colors, 0 represents "sometimes," 1 "yes," and 2 "no.") The codeword c_i corresponding to input symbol i has length l_i, thus defining length vector $\boldsymbol{l} = (l_1, l_2, \ldots, l_n)$. In Fig. 1, for example, c_7 is 22_3 — the codeword corresponding to blue — so length

$l_7 = 2$. The overall code should be a prefix code, that is, no codeword c_i should begin with the entirety of another codeword c_j. In the game, equivalently, we should know when to end the questioning, this being the point at which we know the answer.

For the variant introduced here, all codewords must have lengths lying in a given interval $[l_{\min},l_{\max}]$. In the example of the device mentioned above, $l_{\min} = 20$ and $l_{\max} = 25$. A more practical variant is the problem of designing a data codec that is efficient in terms of not only compression ratio, but also memory use and coding speed. Moffat and Turpin proposed a variety of efficient implementations of prefix encoding and decoding in [6], each involving table lookups rather than code trees. They noted that the length of the longest codeword should be limited for computational efficiency's sake. Computational efficiency is also improved by restricting the overall range of codeword lengths, reducing the size of the coding tables and the expected time of searches required for decoding. Reducing table size is also important for applications with limited memory in which many different Huffman codes are required, due to the use of multiple contexts. Thus, one might wish to have a minimum codeword size of, say, $l_{\min} = 16$ bits and a maximum codeword size of $l_{\max} = 32$ bits ($D = 2$). If expected codeword length for an optimal code found under these restrictions is too long, $l_{\min}$ can be reduced and the algorithm rerun until the proper trade-off is found between compression ratio and complexity (in terms of speed and memory).

A similar problem is one of determining opcodes of a microprocessor designed to use variable-length opcodes, each a certain number of bytes ($D = 256$) with a lower limit and an upper limit to size, e.g., a restriction to opcodes being 16, 24, or 32 bits long ($l_{\min} = 2$, $l_{\max} = 4$). This problem clearly falls within the context considered here, as does the problem of assigning video recorder scheduling codes; these human-readable decimal codes ($D = 10$) have lower and upper bounds on their size, such as $l_{\min} = 3$ and $l_{\max} = 8$, respectively.

Other problems of interest have $l_{\min} = 0$ and are thus length limited but have no practical lower bound on length [7, p. 396]. Yet other problems have not fixed bounds but a constraint on the difference between minimum and maximum codeword length, a quantity referred to as fringe [8, p. 121]. As previously noted, large fringe has a detrimental effect on the speed and memory usage of a decoder. In Section IX of this paper we discuss how to find such codes.

Note that a problem of size n is trivial for certain values of $l_{\min}$ and $l_{\max}$. If $l_{\min} \geq \log_D n$, then all codewords can have $l_{\min}$ output symbols, which, by any reasonable objective, forms an optimal code. If $l_{\max} < \log_D n$, then we cannot code all input symbols and the problem, as presented here, has no solution. Since only other values are interesting, we can assume that $n \in (D^{l_{\min}}, D^{l_{\max}}]$. For example, for the modified form of Twenty Questions, $D = 4$, $l_{\min} = 20$, and $l_{\max} = 25$, so we are only interested in problems where $n \in (2^{40}, 2^{50}]$. Since most instances of Twenty Questions have fewer possible outcomes, this is usually not an interesting problem after all, as instructive as it is. In fact, the fallibility of the answerer and ambiguity of the questioner mean that a decision tree model is not, strictly speaking, correct. For example, the aforementioned answers to questions about the seven colors are debatable. The other applications of *length-bounded* prefix coding mentioned previously, however, do fall within this model.

If we either do not require a minimum or do not require a maximum, it is easy to find values of $l_{\min}$ or $l_{\max}$ which do not limit the problem. As mentioned, setting $l_{\min} = 0$ results in a trivial minimum, as does $l_{\min} = 1$. Similarly, setting $l_{\max} = n$ or using the hard upper bound $l_{\max} = \lceil (n-1)/(D-1) \rceil$ results in a trivial maximum value. In the case of trivial maximum values, one can actually minimize expected codeword length in linear time given sorted inputs. This is possible because, at each stage in the standard Huffman coding algorithm, the set of Huffman trees is an optimal *forest* (set of trees) [9]. We describe the linear-time algorithm in Section VII.

If both minimum and maximum values are trivial, Huffman coding [10] yields a prefix code minimizing expected codeword length

$$\sum_{i=1}^{n} p_i l_i. \tag{1}$$

The conditions necessary and sufficient for the existence of a prefix code with length vector $\boldsymbol{l}$ are the integer constraint, $l_i \in \mathbb{Z}_+$, and the Kraft (McMillan) inequality [11],

$$\kappa(\boldsymbol{l}) \triangleq \sum_{i=1}^{n} D^{-l_i} \leq 1. \tag{2}$$

Finding values for $\boldsymbol{l}$ is sufficient to find a corresponding code, as a code tree with the optimal length vector can be built from sorted codeword lengths in $O(n)$ time and space.

It is not always obvious that we should minimize the expected number of questions $\sum_i p_i l_i$ (or, equivalently, the expected number of questions in excess of the first $l_{\min}$,

$$\sum_{i=1}^{n} p_i (l_i - l_{\min})^+ \tag{3}$$

where x^+ is x if x is positive, 0 otherwise). Consider the example of video recorder scheduling codes. In such an application, one might instead want to minimize mean square distance from $l_{\min}$,

$$\sum_{i=1}^{n} p_i (l_i - l_{\min})^2.$$

We generalize and investigate how to minimize the value

$$\sum_{i=1}^{n} p_i \varphi(l_i - l_{\min}) \tag{4}$$

under the above constraints for any *penalty function* $\varphi(\cdot)$ convex and increasing on $\mathbb{R}_+$. Such an additive measurement of cost is called a *quasiarithmetic penalty*, in this case a convex quasiarithmetic penalty.

One such function family is $\varphi(\delta) = (\delta + l_{\min})^2 + b\delta$, a quadratic objective useful in optimizing a communications delay problem [12]. Another function family, $\varphi(\delta) = D^{t(\delta + l_{\min})}$

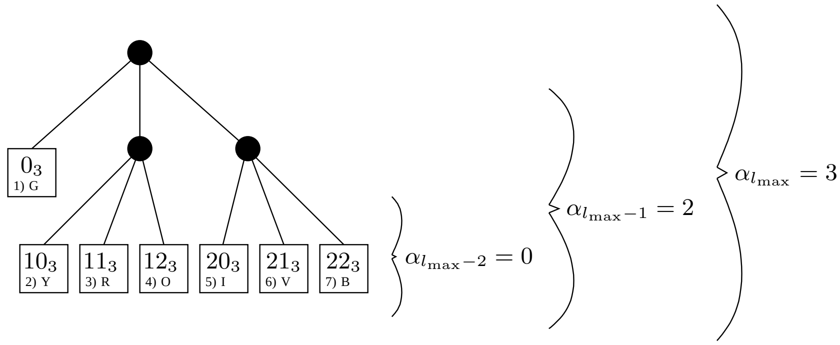

Fig. 1. A monotonic code tree for $n = 7$ and $D = 3$ with $\boldsymbol{l} = (1, 2, 2, 2, 2, 2, 2)$: Each leaf contains the trinary output code, the corresponding object number, and the initial for the corresponding color as in Section I. The α_i's are as defined in Section VIII.

for $t > 0$, can be used to minimize the probability of buffer overflow in a queueing system [13].

Other cost functions that might be of interest concern data expansion [14]. Data expansion occurs where statistics are perfectly known and uncompressed input data are replaced by the compressed data. Uncompressed data take up $\lceil \log_D n \rceil$ D-ary symbols per input symbol. This is replaced by data taking up $l(i)$ D-ary symbols for input item i. Thus, if all occurrences where $l(i)$ exceeds $\lceil \log_D n \rceil$ are prior to the remaining data, the file temporarily expands by up to $\sum_{i=1}^{n} p(i)(l(i) - \lceil \log_D n \rceil)^+$ output symbols per input symbol, or $\varphi(\delta) = (\delta + l_{\min} - \lceil \log_D n \rceil)^+$. This is minimized by a simple fixed-length code. It is not a quantity one would usually want to minimize alone, but an application might trade off this measure with the more traditional measure of expected length, using

$$\varphi(\delta) = \delta + (\lambda)(\delta - \gamma)^+$$

where $\gamma \triangleq \lceil \log_D n \rceil - l_{\min}$ and λ is a positive constant. Such a formulation linearly trades off the two quantities; Huffman coding corresponds to $\lambda = 0$ and fixed-length coding to $\lambda \to \infty$. Solving for this family of objectives, we can minimize one quantity with respect to a constraint on the other or minimize a variety of nonlinear hybrid coding objectives using convex hull techniques like those employed in [12].

Mathematically stating the length-bounded problem,

$$\begin{array}{ll} \text{Given} & \boldsymbol{p} = (p_1, \ldots, p_n),\ p_i > 0; \\ & D \in \{2, 3, \ldots\}; \\ & \text{convex, monotonically increasing} \\ & \varphi : \mathbb{R}_+ \to \mathbb{R}_+ \\ \text{Minimize}_{\{\boldsymbol{l}\}} & \sum_i p_i \varphi(l_i - l_{\min}) \\ \text{subject to} & \sum_i D^{-l_i} \leq 1; \\ & l_i \in \{l_{\min}, l_{\min}+1, \ldots, l_{\max}\}. \end{array}$$

Note that we need not assume that probabilities p_i sum to 1; they could instead be arbitrary positive weights.

Thus, in this paper, given a finite n-symbol input alphabet with an associated probability vector $\boldsymbol{p}$, a D-symbol output alphabet with codewords of lengths $[l_{\min}, l_{\max}]$ allowed, and a constant-time-calculable convex penalty function φ, we describe an $O(n(l_{\max} - l_{\min}))$-time $O(n)$-space algorithm for constructing a φ-optimal code, and sketch an even less complex reduction for the most convex penalty function, $\varphi(\delta) = \delta$, minimization of expected codeword length. In the next section, we present a brief review of the relevant literature. In Section III, we extend to D-ary codes an alternative to code tree notation first presented in [12]. This notation aids in solving the problem in question by reformulating it as an instance of the D-ary Coin Collector's problem, presented in Section IV as an extension of the (binary) Coin Collector's problem [15]. An extension of the Package-Merge algorithm solves this problem; we introduce the reduction and resulting algorithm in Section V. We make it $O(n)$ space in Section VI and refine it in Section VII. The alternative approach for the expected length problem of minimizing (1) — i.e., $\varphi(\delta) = \delta$ — is often faster; this approach is sketched in Section VIII. Algorithmic modifications, applications, possible extensions of this work are discussed in Section IX.

II. Prior Work

Reviewing how the problem in question differs from binary Huffman coding:

1) It can be nonbinary, a case considered by Huffman in his original paper [10];
2) There is a maximum codeword length, a restriction previously considered, e.g., [16] in $O(n^3 l_{\max} \log D)$ time [17] and $O(n^2 \log D)$ space, but solved efficiently only for binary coding, e.g., [15] in $O(n l_{\max})$ time $O(n)$ space and most efficiently in [18];
3) There is a minimum codeword length, a novel restriction;
4) The penalty can be nonlinear, a modification previously considered, but only for binary coding, e.g., [19].

There are several methods for finding optimal codes for various constraints and various types of optimality; we review the three most common families of methods here. Note that other methods fall outside of these families, such as a linear-time method for finding minimum expected length codewords for a uniform distribution with a given fringe [20]. (This differs from the limited-fringe problem of Section IX, in which the distribution need not be uniform and fringe is upper-bounded, not fixed.)

The first and computationally simplest of these are Huffman-like methods, originating with Huffman in 1952 [10] and discussed in, e.g., [21]. Such algorithms are generally linear time given sorted weights and thus $O(n \log n)$ time in general. These are useful for a variety of problems involving penalties in linear, exponential, or minimax form, but not for other nonlinearities nor for length-limited coding. More complex variants of this algorithm are used to find optimal alphabetic codes, that is, codes with codewords constrained to be in a given lexicographical order. These variants are in the Hu-Tucker family of algorithms [9], [22], [23], which, at $O(n \log n)$ time and $O(n)$ space [24], are the most efficient algorithms known for solving such problems (although some instances can be solved in linear time [25], [26]).

The second type of method, dynamic programming, is also conceptually simple but much more computationally complex. Gilbert and Moore proposed a dynamic programming approach in 1959 for finding optimal alphabetic codes, and, unlike the Hu-Tucker algorithm, this approach is readily extensible to search trees [27]. Such an approach can also solve the nonalphabetic problem as a special case, e.g., [16], [28], [29], since any probability vector satisfying $p_i \leq p_j$ for every $i > j$ has an optimal alphabetic code that optimizes the nonalphabetic case. A different dynamic programming approach can be used to find optimal "1"-ended codes [30] and optimal codes with unequal letters costs [31]. Itai [16] used dynamic programming to optimize a large variety of coding and search tree problems, including nonbinary length-limited coding, which is done with $O(n^2 l_{\max} \log D)$ time and $O(n^2 \log D)$ space by a reduction to the alphabetic case. We reduce complexity significantly in this paper.

The third family is that of Package-Merge-based algorithms, and this is the type of approach we use for the generalized algorithm considered here. Introduced in 1990 by Larmore and Hirschberg [15], this approach is most often used for binary length-limited linear-penalty Huffman coding, although it has been extended for application to binary alphabetic codes [32] and to binary convex quasiarithmetic penalty functions [19]. The algorithms in this approach generally have $O(nl_{\max})$-time $O(n)$-space complexity, although space complexity can vary by application and implementation, and the alphabetic variant and some nonquasiarithmetic (and thus nonlinear) variants have slightly higher time complexity ($O(nl_{\max} \log n)$).

To use this approach for nonbinary coding with a lower bound on codeword length, we need to alter the approach, generalizing to the problem of interest. The minimum size constraint on codeword length requires a relatively simple change of solution range. The nonbinary coding generalization is a bit more involved; it requires first modifying the Package-Merge algorithm to allow for an arbitrary numerical base (binary, ternary, etc.), then modifying the coding problem to allow for a provable reduction to the modified Package-Merge algorithm. At times "dummy" inputs are added in order to assist in finding an optimal nonbinary code. In order to make the algorithm precise, the $O(n(l_{\max} - l_{\min}))$-time $O(n)$-space algorithm, unlike some other implementations [15], minimizes *height* (that is, maximum codeword length) among optimal codes (if multiple optimal codes exist).

III. Nodeset Notation

Before presenting an algorithm for optimizing the above problem, we introduce a notation for codes that generalizes one first presented in [12] and modified in [19]. Nodeset notation, an alternative to code tree notation, has previously been used for binary alphabets, but not for general D-ary alphabet coding, thus the need for generalization.

The key idea: Each node (i, l) represents both the share of the penalty (4) (*weight*) and the (scaled) share of the Kraft sum (2) (*width*) assumed for the lth bit of the ith codeword. By showing that total weight is an increasing function of the penalty and that there is a one-to-one correspondence between an optimal code and a corresponding optimal nodeset, we reduce the problem to an efficiently solvable problem, the Coin Collector's problem.

In order to do this, we first need to make a modification to the problem analogous to one Huffman made in his original nonbinary solution. We must in some cases add a "dummy" input or "dummy" inputs of infinitesimal probability $p_i = \epsilon > 0$ to the probability vector to assure that the optimal code has the Kraft inequality satisfied with equality, an assumption underlying both the Huffman algorithm and ours. The positive probabilities of these dummy inputs mean that codes obtained could be slightly suboptimal, but we later specify an algorithm where $\epsilon = 0$, obviating this concern.

As with traditional Huffman coding [10], the number of dummy values needed is $(D - n) \bmod (D - 1)$, where

$$x \bmod y \triangleq x - y\lfloor x/y \rfloor$$

for all integers x (not just nonnegative integers). Such dummy inputs allow us to assume that the optimal tree (for real plus dummy items) is an optimal full tree (i.e., that $\kappa(\boldsymbol{l}) = 1$, where κ is as defined in (2)). For sufficiently small ϵ, the code will be identical to that for $\epsilon = 0$, and, as in traditional Huffman coding, nondummy codewords are identical to the codewords of an optimal code for the original input distribution. We can thus assume for our algorithm that $\kappa(\boldsymbol{l}) = 1$ and $n \bmod (D - 1) \equiv 1$.

With this we now present *nodeset* notation:

Definition 1: A *node* is an ordered pair of integers (i, l) such that $i \in \{1, \ldots, n\}$ and $l \in \{l_{\min} + 1, \ldots, l_{\max}\}$. Call the set of all possible nodes I. This set can be arranged in an $n \times (l_{\max} - l_{\min})$ grid, e.g., Fig. 2. The set of nodes, or *nodeset*, corresponding to input symbol i (assigned codeword c_i with length l_i) is the set of the first $l_i - l_{\min}$ nodes of column i, that is, $\eta_{\boldsymbol{l}}(i) \triangleq \{(j, l) \mid j = i,\ l \in \{l_{\min} + 1, \ldots, l_i\}\}$. The nodeset corresponding to length vector $\boldsymbol{l}$ is $\eta(\boldsymbol{l}) \triangleq \bigcup_i \eta_{\boldsymbol{l}}(i)$; this corresponds to a set of n codewords, a code. Thus, in Fig. 2, the dashed line surrounds a nodeset corresponding to $\boldsymbol{l} = (1, 2, 2, 2, 2, 2, 2)$. We say a node (i, l) has *width* $\rho(i, l) \triangleq D^{-l}$ and *weight* $\mu(i, l) \triangleq p_i \varphi(l - l_{\min}) - p_i \varphi(l - l_{\min} - 1)$, as shown in the example in Fig. 2. Note that if $\varphi(l) = l$, $\mu(i, l) = p_i$.

We must emphasize that the above "nodes" are unlike nodes in a graph; similar structures are sometimes instead called *tiles* [33], but we retain the original, more prevalent term "nodes." Given valid nodeset $N \subseteq I$, it is straightforward to find the corresponding length vector and, if it satisfies the Kraft inequality, a code.

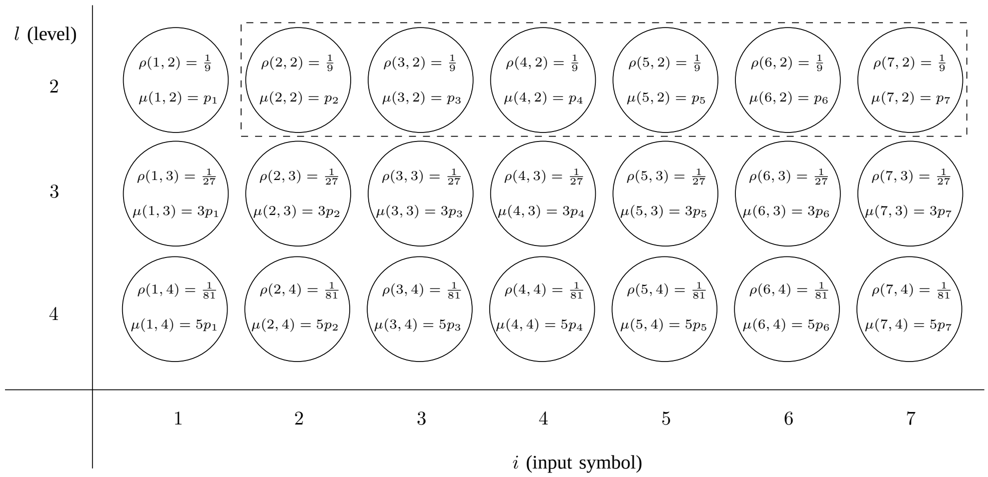

Fig. 2. The set of nodes I with widths $\{\rho(i,l)\}$ and weights $\{\mu(i,l)\}$ for $\varphi(\delta)=\delta^2$, $n=7$, $D=3$, $l_{\min}=1$, $l_{\max}=4$

IV. The D-ary Coin Collector's Problem and the Package-Merge Algorithm

We find optimal codes by first solving a related problem, the Coin Collector's problem. Let $D^{\mathbb{Z}}$ denote the set of all integer powers of a fixed integer $D>1$. The Coin Collector's problem of size m considers "coins" indexed by $i \in \{1,2,\ldots,m\}$. Each coin has a width, $\rho_i \in D^{\mathbb{Z}}$; one can think of width as coin face value, e.g., $\rho_i = 0.25 = 2^{-2}$ for a quarter dollar (25 cents). Each coin also has a weight, $\mu_i \in \mathbb{R}$. The final problem parameter is total width, denoted $\rho_{\rm tot}$. The problem is then:

$$\begin{array}{lll} \text{Minimize}_{\{B \subseteq \{1,\ldots,m\}\}} & \sum_{i\in B} \mu_i & \\ \text{subject to} & \sum_{i \in B} \rho_i = \rho_{\rm tot} & \\ \text{where} & m \in \mathbb{Z}_+ & \\ & \mu_i \in \mathbb{R} & \\ & \rho_i \in D^{\mathbb{Z}} & \\ & \rho_{\rm tot} \in \mathbb{R}_+. & \end{array} \tag{5}$$

We thus wish to choose coins with total width $\rho_{\rm tot}$ such that their total weight is as small as possible. This problem is an input-restricted variant of the knapsack problem. However, given sorted inputs, a linear-time solution to (5) for $D=2$ was proposed in [15]. The algorithm in question is called the *Package-Merge algorithm* and we extend it here to arbitrary D.

In our notation, we use $i \in \{1,\ldots,m\}$ to denote both the index of a coin and the coin itself, and $\mathcal{I}$ to represent the m items along with their weights and widths. The optimal solution, a function of total width $\rho_{\rm tot}$ and items $\mathcal{I}$, is denoted $\mathrm{CC}(\mathcal{I},\rho_{\rm tot})$ (the optimal coin collection for $\mathcal{I}$ and $\rho_{\rm tot}$). Note that, due to ties, this need not be unique, but we assume that one of the optimal solutions is chosen; at the end of Section VI, we discuss how to break ties.

Because we only consider cases in which a solution exists, $\rho_{\rm tot} = \omega \rho_{\rm pow}$ for some $\rho_{\rm pow} \in D^{\mathbb{Z}}$ and $\omega \in \mathbb{Z}_+$. Here, assuming $\rho_{\rm tot} > 0$, $\rho_{\rm pow}$ and ω are the unique pair of a power of D and an integer that is not a multiple of D, respectively, which, multiplied, form $\rho_{\rm tot}$. If $\rho_{\rm tot}=0$, ω and $\rho_{\rm pow}$ are not used. Note that $\rho_{\rm pow}$ need not be an integer.

Algorithm variables

At any point in the algorithm, given nontrivial $\mathcal{I}$ and $\rho_{\rm tot}$, we use the following definitions:

Remainder
$$\rho_{\rm pow} \triangleq \text{the unique } x \in D^{\mathbb{Z}} \text{ such that } \frac{\rho_{\rm tot}}{x} \in \mathbb{Z} \backslash D\mathbb{Z}$$

Minimum width
$$\rho^* \triangleq \min_{i \in \mathcal{I}} \rho_i \quad (\text{note } \rho^* \in D^{\mathbb{Z}})$$

Small width set
$$\mathcal{I}^* \triangleq \{i \mid \rho_i = \rho^*\} \quad (\text{note } \mathcal{I}^* \neq \emptyset)$$

"First" item
$$i^* \triangleq \arg\min_{i \in \mathcal{I}^*} \mu_i$$

"First" package
$$\mathcal{P}^* \triangleq \begin{cases} \mathcal{P} \text{ such that} & \\ \quad |\mathcal{P}| = D, & \\ \quad \mathcal{P} \subseteq \mathcal{I}^*, & \\ \quad \mathcal{P} \preceq \mathcal{I}^* \backslash \mathcal{P}, & |\mathcal{I}^*| \geq D \\ \emptyset, & |\mathcal{I}^*| < D \end{cases}$$

where $D\mathbb{Z}$ denotes integer multiples of D and $\mathcal{P} \preceq \mathcal{I}^* \backslash \mathcal{P}$ denotes that, for all $i \in \mathcal{P}$ and $j \in \mathcal{I}^* \backslash \mathcal{P}$, $\mu_i \leq \mu_j$. Then the following is a recursive description of the algorithm:

Recursive D-ary Package-Merge Procedure

Basis. $\rho_{\rm tot} = 0$: $\mathrm{CC}(\mathcal{I},\rho_{\rm tot}) = \emptyset$.

Case 1. $\rho^* = \rho_{\rm pow}$ and $\mathcal{I} \neq \emptyset$: $\mathrm{CC}(\mathcal{I},\rho_{\rm tot}) = \mathrm{CC}(\mathcal{I}\backslash\{i^*\}, \rho_{\rm tot} - \rho^*) \cup \{i^*\}$.

Case 2a. $\rho^* < \rho_{\rm pow}$, $\mathcal{I} \neq \emptyset$, and $|\mathcal{I}^*| < D$: $\mathrm{CC}(\mathcal{I},\rho_{\rm tot}) = \mathrm{CC}(\mathcal{I}\backslash\mathcal{I}^*, \rho_{\rm tot})$.

Case 2b. $\rho^* < \rho_{\rm pow}$, $\mathcal{I} \neq \emptyset$, and $|\mathcal{I}^*| \geq D$: Create i', a new item with weight $\mu_{i'} = \sum_{i \in \mathcal{P}^*} \mu_i$ and width $\rho_{i'} = D\rho^*$. This new item is thus a combined item, or *package*, formed by combining the D least weighted items of width ρ^*. Let $\mathcal{S} = \mathrm{CC}(\mathcal{I}\backslash\mathcal{P}^* \cup \{i'\}, \rho_{\rm tot})$ (the optimization of the packaged version). If $i' \in \mathcal{S}$, then $\mathrm{CC}(\mathcal{I},\rho_{\rm tot}) = \mathcal{S}\backslash\{i'\} \cup \mathcal{P}^*$; otherwise, $\mathrm{CC}(\mathcal{I},\rho_{\rm tot}) = \mathcal{S}$.

Theorem 1: If an optimal solution to the Coin Collector's problem exists, the above recursive (Package-Merge) algorithm will terminate with an optimal solution.

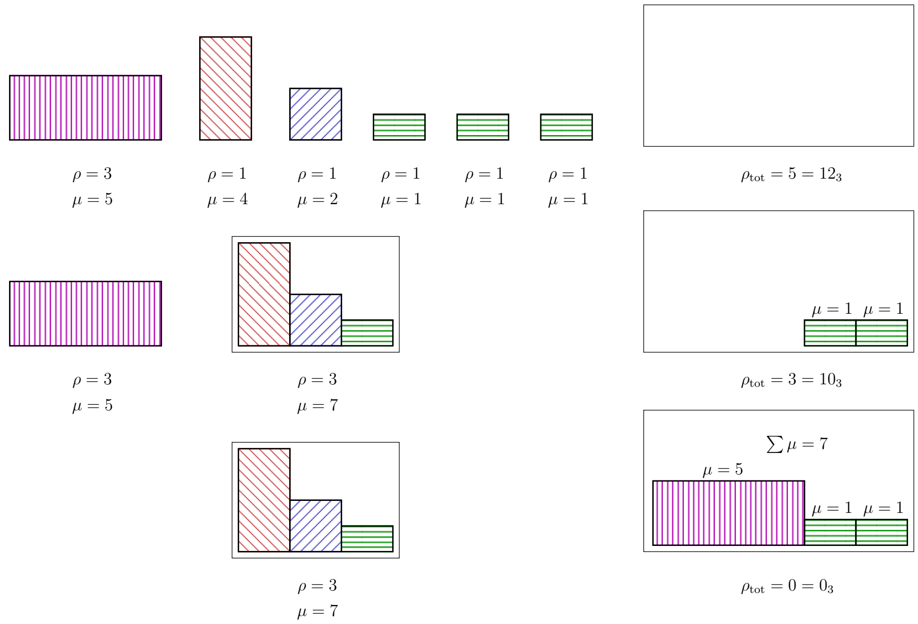

Fig. 3. A simple example of the Package-Merge algorithm

Proof: We show that the Package-Merge algorithm produces an optimal solution via induction on the number of input items. The basis is trivially correct, and each inductive case reduces the number of items by at least one. The inductive hypothesis on $\rho_{\rm tot} \geq 0$ and $\mathcal{I} \neq \emptyset$ is that the algorithm is correct for any problem instance with fewer input items than instance $(\mathcal{I}, \rho_{\rm tot})$.

If $\rho^* > \rho_{\rm pow} > 0$, or if $\mathcal{I} = \emptyset$ and $\rho_{\rm tot} \neq 0$, then there is no solution to the problem, contrary to our assumption. Thus all feasible cases are covered by those given in the procedure. Case 1 indicates that the solution must contain at least one element (item or package) of width ρ^*. These must include the minimum weight item in $\mathcal{I}^*$, since otherwise we could substitute one of the items with this "first" item and achieve improvement. Case 2 indicates that the solution must contain a number of elements of width ρ^* that is a multiple of D. If this number is 0, none of the items in $\mathcal{P}^*$ are in the solution. If it is not, then they all are. Thus, if $\mathcal{P}^* = \emptyset$, the number is 0, and we have Case 2a. If not, we may "package" the items, considering the replaced package as one item, as in Case 2b. Thus the inductive hypothesis holds. ■

Fig. 3 presents a simple example of this algorithm at work for $D = 3$, finding minimum total weight items of total width $\rho_{\rm tot} = 5$ (or, in ternary, 12_3). In the figure, item width represents numeric width and item area represents numeric weight. Initially, as shown in the top row, the minimum weight item has width $\rho^* = \rho_{i^*} = \rho_{\rm pow} = 1$. This item is put into the solution set, and the next step repeats the task on the items remaining outside the solution set. Then, the remaining minimum width items are packaged into a merged item of width 3 (10_3), as in the middle row. Finally, the minimum weight item/package with width $\rho^* = \rho_{i^*} = \rho_{\rm pow} = 3$ is added to complete the solution set, which is now of weight 7. The remaining packaged item is left out in this case; when the algorithm is used for coding, several items are usually left out of the optimal set. Given input sorted first by width then weight, the resulting algorithm is $O(m)$ time and space.

V. A General Algorithm

We now formalize the reduction from the coding problem to the Coin Collector's problem. This generalizes the similar reduction shown in [19] for binary codes with only a limit on maximum length, which is in turn a generalization of [15] for length-limited binary codes with linear φ, the traditional penalty function.

We assert that any optimal solution N of the Coin Collector's problem with total width

$$\rho_{\rm tot} = \frac{n - D^{l_{\min}}}{D-1} D^{-l_{\min}}$$

on coins $\mathcal{I}$ (identical to the set of all possible nodes I) is a nodeset for an optimal solution of the coding problem. This yields a suitable method for solving the problem.

To show this reduction, we first define $\rho(N)$ in a natural manner for any $N = \eta(\boldsymbol{l})$:

$$\begin{aligned} \rho(N) &\triangleq \sum_{(i,l) \in N} \rho(i,l) \\ &= \sum_{i=1}^{n} \sum_{l=l_{\min}+1}^{l_i} D^{-l} \\ &= \sum_{i=1}^{n} \frac{D^{-l_{\min}} - D^{-l_i}}{D-1} \\ &= \frac{nD^{-l_{\min}} - \kappa(\boldsymbol{l})}{D-1} \end{aligned}$$

where $\kappa(\boldsymbol{l})$ is the Kraft sum (2). Given $n \bmod (D-1) \equiv 1$, all optimal codes have the Kraft inequality satisfied with equality; otherwise, the longest codeword length could be shortened by one, strictly decreasing the penalty without

violating the inequality. Thus the optimal solution has $\kappa(\boldsymbol{l}) = 1$ and
$$\rho(N) = \frac{n - D^{l_{\min}}}{D-1} D^{-l_{\min}}.$$

Also define:
$$\begin{array}{rcl} \delta_\varphi(l) & \triangleq & \varphi(l) - \varphi(l-1) \\ \mu(N) & \triangleq & \sum_{(i,l)\in N} \mu(i,l). \end{array}$$

Note that
$$\begin{array}{rcl} \mu(N) & = & \sum_{(i,l)\in N} \mu(i,l) \\ & = & \sum_{i=1}^{n} \sum_{l=l_{\min}+1}^{l_i} p_i \delta_\varphi(l - l_{\min}) \\ & = & \sum_{i=1}^{n} p_i \varphi(l_i - l_{\min}) - \sum_{i=1}^{n} p_i \varphi(0). \end{array}$$

Since the subtracted term is a constant, if the optimal nodeset corresponds to a valid code, solving the Coin Collector's problem solves this coding problem. To prove the reduction, we need to prove that the optimal nodeset indeed corresponds to a valid code. We begin with the following lemma:

Lemma 1: Suppose that N is a nodeset of width $xD^{-k} + r$ where k and x are integers and $0 < r < D^{-k}$. Then N has a subset R with width r.

Proof: Let us use induction on the cardinality of the set. The base case $|N| = 1$ is trivial since then $x = 0$. Assume the lemma holds for all $|N| < n$, and suppose $|\tilde{N}| = n$. Let $\rho^* = \min_{j\in\tilde{N}} \rho_j$ and $j^* = \arg\min_{j\in\tilde{N}} \rho_j$. We can view item j^* of width $\rho^* \in D^{\mathbb{Z}}$ as the smallest contributor to the width of $\tilde{N}$ and r as the portion of the D-ary expansion of the width of $\tilde{N}$ to the right of D^{-k}. Then r must be an integer multiple of ρ^*. If $r = \rho^*$, $R = \{j^*\}$ is a solution. Otherwise let $N' = \tilde{N}\backslash\{j^*\}$ (so $|N'| = n-1$) and let R' be the subset obtained from solving the lemma for set N' of width $r - \rho^*$. Then $R = R' \cup \{j^*\}$. ■

We now prove the reduction:

Theorem 2: Any N that is a solution of the Coin Collector's problem for
$$\rho_{\text{tot}} = \rho(N) = \frac{n - D^{l_{\min}}}{D-1} D^{-l_{\min}}$$
has a corresponding length vector $\boldsymbol{l}^N$ such that $N = \eta(\boldsymbol{l}^N)$ and $\mu(N) = \min_{\boldsymbol{l}} \sum_i p_i \varphi(l_i - l_{\min}) - \varphi(0) \sum_i p_i$.

Proof: Any optimal length vector nodeset has $\rho(\eta(\boldsymbol{l})) = \rho_{\text{tot}}$. Suppose N is a solution to the Coin Collector's problem but is not a valid nodeset of a length vector. Then there exists an (i,l) with $l \in [l_{\min}+2, l_{\max}]$ such that $(i,l) \in N$ and $(i, l-1) \in I\backslash N$. Let $R' = N \cup \{(i,l-1)\}\backslash\{(i,l)\}$. Then $\rho(R') = \rho_{\text{tot}} + (D-1)D^{-l}$ and, due to convexity, $\mu(R') \leq \mu(N)$. Using $n \bmod (D-1) \equiv 1$, we know that ρ_{tot} is an integer multiple of $D^{-l_{\min}}$. Thus, using Lemma 1 with $k = l_{\min}$, $x = \rho_{\text{tot}} D^{l_{\min}}$, and $r = (D-1)D^{-l}$, there exists an $R \subset R'$ such that $\rho(R) = r$. Since $\mu(R) > 0$, $\mu(R'\backslash R) < \mu(R') \leq \mu(N)$. This is a contradiction to N being an optimal solution to the Coin Collector's problem, and thus any optimal solution of the Coin Collector's problem corresponds to an optimal length vector. ■

Because the Coin Collector's problem is linear in time and space — same-width inputs are presorted by weight, numerical operations and comparisons are constant time — the overall algorithm finds an optimal code in $O(n(l_{\max} - l_{\min}))$ time and space. Space complexity, however, can be decreased.

VI. A Deterministic $O(n)$-Space Algorithm

If $p_i = p_j$, we are guaranteed no particular inequality relation between l_i and l_j since we did not specify a method for breaking ties. Thus the length vector returned by the algorithm need not have the property that $l_i \leq l_j$ whenever $i < j$. We would like to have an algorithm that has such a monotonicity property.

Definition 2: A *monotonic* nodeset, N, is one with the following properties:
$$(i,l) \in N \Rightarrow (i+1, l) \in N \quad \text{for } i < n \tag{6}$$
$$(i,l) \in N \Rightarrow (i, l-1) \in N \quad \text{for } l > l_{\min}+1. \tag{7}$$

In other words, a nodeset is monotonic if and only if it corresponds to a length vector $\boldsymbol{l}$ with lengths sorted in increasing order; this definition is equivalent to that given in [15].

Examples of monotonic nodesets include the sets of nodes enclosed by dashed lines in Fig. 2 and Fig. 4. In the latter case, $n = 21$, $D = 3$, $l_{\min} = 2$, and $l_{\max} = 8$, so $\rho_{\text{tot}} = 2/3$. As indicated, if $p_i = p_j$ for some i and j, then an optimal nodeset need not be monotonic. However, if all probabilities are distinct, the optimal nodeset is monotonic.

Lemma 2: If $\boldsymbol{p}$ has no repeated values, then any optimal solution $N = \text{CC}(I, n-1)$ is monotonic.

Proof: The second monotonic property (7) was proved for optimal nodesets in Theorem 2. The first property (6) can be shown via a simple exchange argument. Consider optimal $\boldsymbol{l}$ with $i > j$ so that $p_i < p_j$, and also consider $\boldsymbol{l}'$ with lengths for inputs i and j interchanged, as in [34, pp. 97–98]. Then
$$\begin{array}{l} \sum_k p_k \varphi(l'_k - l_{\min}) - \sum_k p_k \varphi(l_k - l_{\min}) \\ \quad = (p_j - p_i)\left[\varphi(l_i - l_{\min}) - \varphi(l_j - l_{\min})\right] \\ \quad \leq 0 \end{array}$$
where the inequality is to due to the optimality of $\boldsymbol{l}$. Since $p_j - p_i > 0$ and φ is monotonically increasing, $l_i \geq l_j$ for all $i > j$ and an optimal nodeset without repeated $\boldsymbol{p}$ must be monotonic. ■

Taking advantage of monotonicity in a Package-Merge coding implementation to trade off a constant factor of time for drastically reduced space complexity is done in [12] for length-limited binary codes. We extend this to the length-bounded problem, first for $\boldsymbol{p}$ without repeated values, then for arbitrary $\boldsymbol{p}$.

Note that the total width of items that are each less than or equal to width ρ is less than $2n\rho$. Thus, when we are processing items and packages of width ρ, fewer than $2n$ packages are kept in memory. The key idea in reducing space complexity is to keep only four attributes of each package in memory instead of the full contents. In this manner, we use $O(n)$ space while retaining enough information to reconstruct the optimal nodeset in algorithmic postprocessing.

Define

$$l_{\mathrm{mid}} \triangleq \left\lfloor \frac{1}{2}(l_{\max} + l_{\min} + 1) \right\rfloor .$$

For each package S, we retain only the following attributes:

1) $\mu(S) \triangleq \sum_{(i,l)\in S} \mu(i,l)$
2) $\rho(S) \triangleq \sum_{(i,l)\in S} \rho(i,l)$
3) $\nu(S) \triangleq |S \cap I_{\mathrm{mid}}|$
4) $\psi(S) \triangleq \sum_{(i,l)\in S\cap I_{\mathrm{hi}}} \rho(i,l)$

where $I_{\mathrm{hi}} \triangleq \{(i,l) \mid l > l_{\mathrm{mid}}\}$ and $I_{\mathrm{mid}} \triangleq \{(i,l) \mid l = l_{\mathrm{mid}}\}$. We also define $I_{\mathrm{lo}} \triangleq \{(i,l) \mid l < l_{\mathrm{mid}}\}$.

With only these parameters, the "first run" of the algorithm takes $O(n)$ space. The output of this run is the package attributes of the optimal nodeset N. Thus, at the end of this first run, we know the value for $n_\nu \triangleq \nu(N)$, and we can consider N as the disjoint union of four sets, shown in Fig. 4:

1) A = nodes in $N \cap I_{\mathrm{lo}}$ with indices in $[1, n - n_\nu]$,
2) B = nodes in $N \cap I_{\mathrm{lo}}$ with indices in $[n - n_\nu + 1, n]$,
3) $\mathit{\Gamma}$ = nodes in $N \cap I_{\mathrm{mid}}$,
4) $\mathit{\Delta}$ = nodes in $N \cap I_{\mathrm{hi}}$.

Due to the monotonicity of N, it is clear that $B = [n - n_\nu + 1, n] \times [l_{\min} + 1, l_{\mathrm{mid}} - 1]$ and $\mathit{\Gamma} = [n - n_\nu + 1, n] \times \{l_{\mathrm{mid}}\}$. Note then that $\rho(B) = (n_\nu)(D^{-l_{\min}} - D^{1-l_{\mathrm{mid}}})/(D-1)$ and $\rho(\mathit{\Gamma}) = n_\nu D^{-l_{\mathrm{mid}}}$. Thus we need merely to recompute which nodes are in A and in $\mathit{\Delta}$.

Because $\mathit{\Delta}$ is a subset of I_{hi}, $\rho(\mathit{\Delta}) = \psi(N)$ and $\rho(A) = \rho(N) - \rho(B) - \rho(\mathit{\Gamma}) - \rho(\mathit{\Delta})$. Given their respective widths, A is a minimal weight subset of $[1, n - n_\nu] \times [l_{\min} + 1, l_{\mathrm{mid}} - 1]$ and $\mathit{\Delta}$ is a minimal weight subset of $[n - n_\nu + 1, n] \times [l_{\mathrm{mid}} + 1, l_{\max}]$. These are monotonic if the overall nodeset is monotonic. The nodes at each level of A and $\mathit{\Delta}$ can thus be found by recursive calls to the algorithm. This approach uses only $O(n)$ space while preserving time complexity; one run of an algorithm on $n(l_{\max} - l_{\min})$ nodes is replaced with a series of runs, first one on $n(l_{\max} - l_{\min})$ nodes, then two on an average of at most $n(l_{\max} - l_{\min})/4$ nodes each, then four on an average of at most $n(l_{\max} - l_{\min})/16$, and so forth. An optimization of the same complexity is made in [15], where it is proven that this yields $O(n(l_{\max} - l_{\min}))$ time complexity with a linear space requirement. Given the hard bounds for $l_{\max}$ and $l_{\min}$, this is always $O(n^2/D)$.

The assumption of distinct p_i's puts an undesirable restriction on our input that we now relax. In doing so, we make the algorithm deterministic, resolving ties that make certain minimization steps of the algorithm implementation dependent. This results in what in some sense is the "best" optimal code if multiple monotonic optimal codes exist.

Recall that $\boldsymbol{p}$ is a nonincreasing vector. Thus items of a given width are sorted for use in the Package-Merge algorithm; this order is used to break ties. For example, if we look at the problem in Fig. 2 — $\varphi(\delta) = \delta^2$, $n = 7$, $D = 3$, $l_{\min} = 1$, $l_{\max} = 4$ — with probability vector $\boldsymbol{p} = (0.4, 0.3, 0.14, 0.06, 0.06, 0.02, 0.02)$, then nodes $(7,4)$, $(6,4)$, and $(5,4)$ are the first to be grouped, the tie between $(5,4)$ and $(4,4)$ broken by order. Thus, at any step, all identical-width items in one package have adjacent indices. Recall that packages of items will be either in the final nodeset or absent from it as a whole. This scheme then prevents any of the nonmonotonicity that identical p_i's might bring about.

In order to assure that the algorithm is fully deterministic, the manner in which packages and single items are merged must also be taken into account. We choose to combine nonmerged items before merged items in the case of ties, in a similar manner to the two-queue bottom-merge method of Huffman coding [5], [35]. Thus, in our example, there is a point at which the node $(2,2)$ is chosen (to be merged with $(3,2)$ and $(4,2)$) while the identical-weight package of items $(5,3)$, $(6,3)$, and $(7,3)$ is not. This leads to the optimal length vector $\boldsymbol{l} = (1,2,2,2,2,2,2)$, rather than $\boldsymbol{l} = (1,1,2,2,3,3,3)$ or $\boldsymbol{l} = (1,1,2,3,2,3,3)$, which are also optimal. The corresponding nodeset is enclosed within the dashed line in Fig. 2, and the resulting monotonic code tree is the code tree shown in Fig. 1.

This approach also enables us to set ϵ, the value for dummy variables, equal to 0 without violating monotonicity. As in bottom-merge Huffman coding, the code with the minimum reverse lexicographical order among optimal codes (and thus the one with minimum height) is the one produced; reverse lexicographical order is the lexicographical order of lengths after their being sorted largest to smallest. An identical result can be obtained by using the position of the "largest" node in a package (in terms of position number $nl+i$) in order to choose those with lower values, as in [32]. However, our approach, which can be shown to be equivalent via simple induction, eliminates the need for keeping track of the maximum value of $nl+i$ for each package.

VII. Further Refinements

There are changes we can make to the algorithm that, for certain inputs, result in even better performance. For example, if $l_{\max} \approx \log_D n$, then, rather than minimizing the weight of nodes of a certain total width, it is easier to maximize weight over a complementary total width and find the complementary set of nodes. Similarly, if most input symbols have one of a handful of probability values, one can consider this and simplify calculations. These and other similar optimizations have been done in the past for the special case $\varphi(\delta) = \delta$, $l_{\min} = 0$, $D = 2$ [36]–[40], though we do not address or extend such improvements here.

So far we have assumed that $l_{\max}$ is the best upper bound on codeword length we could obtain. However, there are many cases in which we can narrow the range of codeword lengths, thus making the algorithm faster. For example, since, as stated previously, we can assume without loss of generality that $l_{\max} \leq \lceil (n-1)/(D-1) \rceil$, we can eliminate the bottom row of nodes from consideration in Fig. 2.

Consider also when $l_{\min} = 0$. An upper bound on $\{l_i\}$ can be derived from a theorem and a definition due to Larmore:

Definition 3: Consider penalty functions φ and χ. We say that χ is *flatter* than φ if, for positive integers $l' > l$, $(\chi(l) - \chi(l-1))(\varphi(l') - \varphi(l'-1)) \leq (\varphi(l) - \varphi(l-1))(\chi(l') - \chi(l'-1))$. [12].

A consequence of the Convex Hull Theorem of [12] is that, given χ flatter than φ, for any $\boldsymbol{p}$, there exist φ-optimal $\boldsymbol{l}^{(\varphi)}$

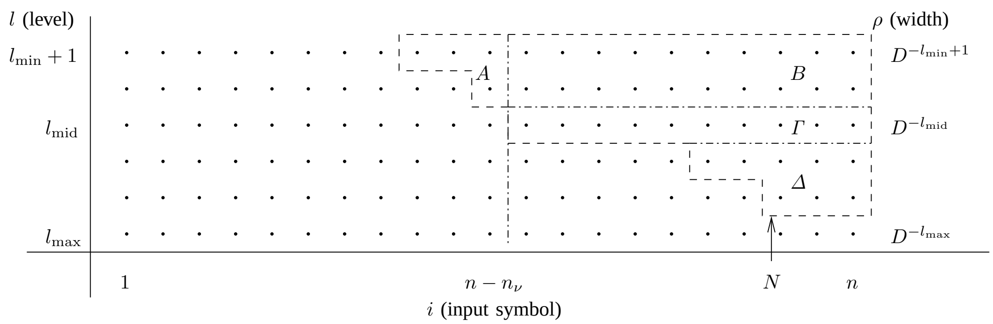

Fig. 4. The set of nodes I, an optimal nodeset N, and disjoint subsets A, B, Γ, Δ

and χ-optimal $\boldsymbol{l}^{(\chi)}$ such that $\boldsymbol{l}^{(\varphi)}$ is greater than $\boldsymbol{l}^{(\chi)}$ in terms of reverse lexicographical order. This explains why the word "flatter" is used.

Penalties flatter than the linear penalty — i.e., convex φ — can therefore yield a useful upper bound, reducing complexity. Thus, if $l_{\min}=0$, we can use the results of a pre-algorithmic Huffman coding of the input symbols to find an upper bound on codeword length in linear time, one that might be better than $l_{\max}$. Alternatively, we can use the least probable input to find a looser upper bound, as in [41].

When $l_{\min}>1$, one can still use a modified pre-algorithmic Huffman coding to find an upper bound as long as $\varphi(\delta)=\delta$. This is done via a modification of the Huffman algorithm allowing an arbitrary minimum $l_{\min}$ and a trivial maximum (e.g., $l_{\max}=n$ or $\lceil (n-1)/(D-1)\rceil$):

Procedure for length-lower-bounded ("truncated Huffman") coding

1) Add $(D-n) \bmod (D-1)$ dummy items of probability 0.
2) Combine the items with the D smallest probabilities $p_{i_1}, p_{i_2}, \ldots, p_{i_D}$ into one item with the combined probability $\tilde{p}_i = \sum_{i=1}^{D} p_{i_1}$. This item has codeword $\tilde{c}_i$, to be determined later, while these D smallest items are assigned concatenations of this yet-to-be-determined codeword and every possible output symbol, that is, $c_{i_1}=\tilde{c}_i 0, c_{i_2}=\tilde{c}_i 1, \ldots, c_{i_D}=\tilde{c}_i(D-1)$. Since these have been assigned in terms of $\tilde{c}_i$, replace the smallest D items with $\tilde{p}_i$ in $\boldsymbol{p}$ to form $\tilde{\boldsymbol{p}}$.
3) Repeat previous step, now with the remaining $n-D+1$ codewords and corresponding probabilities, until only $D^{l_{\min}}$ items are left.
4) Assign all possible $l_{\min}$ long codewords to these items, thus defining the overall code based on the fixed-length code assigned to these combined items.

This procedure is Huffman coding truncated midway through coding, the resulting trees serving as subtrees of nodes of identical depth. Excluding the last step, the algorithm is identical to that shown in [42] to result in an optimal Huffman forest. The optimality of the algorithm for length-lower-bounded coding is an immediate consequence of the optimality of the forest, as both have the same constraints and the same value to minimize. As with the usual Huffman algorithm, this can be made linear time given sorted inputs [35] and can be made to find a code with the minimum reverse lexicographical order among optimal codes via the bottom-merge variant.

Clearly, this algorithm finds the optimal code for the length-bounded problem if the resulting code has no codeword longer than $l_{\max}$, whether this be because $l_{\max}$ is trivial or because of other specifications of the problem. If this truncated Huffman algorithm fails, then we know that $l_n = l_{\max}$, that is, we cannot have that $l_n < l_{\max}$ for the length-bounded code. This is an intuitive result, but one worth stating and proving, as it is used in the next section:

Lemma 3: If a (truncated) Huffman code ($\varphi(\delta)=\delta$) for $l_{\min}$ has a codeword longer than some l_{ub}, then there exists an optimal length-bounded code for bound $[l_{\min}, l_{\text{ub}}]$ with codewords of length l_{ub}.

Proof: It suffices to show that, if an optimal code for the bound $[l_{\min}, l_{\max}]$ has a codeword with length $l_{\max}$, then an optimal code for the bound $[l_{\min}, l_{\max}-1]$ has a codeword with length $l_{\max}-1$, since this can be applied inductively from $l_{\max}=l_n$ (assuming l_n is the length of the longest codeword of the truncated Huffman code) to l_{ub}, obtaining the desired result. The optimal nodeset N for the bound $[l_{\min}, l_{\max}]$ has width $D^{-l_{\min}}(n-D^{l_{\min}})/(D-1)$. Therefore, in the course of the Package-Merge algorithm, we at one point have $(n-D^{l_{\min}})/(D-1)$ packages of width $D^{-l_{\min}}$ which will eventually comprise optimal nodeset N, these packages having weight no larger than the remaining packages of the same width.

Consider the nodeset N' formed by making each (i,l) in N into $(i, l-1)$. This nodeset is the solution to the Package-Merge algorithm for the total width $D^{-l_{\min}+1}(n-D^{l_{\min}})/(D-1)$ with bounds $l_{\min}-1$ and $l_{\max}-1$. Let $i(l)$ denote the number of nodes on level l. Then $i(l_{\min}) \geq n-D^{l_{\min}}$ since at most $D^{l_{\min}}$ nodes can have length $l_{\min}$. The subset of N' not of depth $l_{\min}-1$ is thus an optimal solution for bounds $l_{\min}$ and $l_{\max}-1$ with total width

$$D^{-l_{\min}}\left(\frac{D(n-D^{l_{\min}})}{D-1} - Di(l_{\min})\right)$$

that is, at one point in the algorithm this solution corresponds to the $D(n-D^{l_{\min}})/(D-1)-i(l_{\min})$ least weighted packages of width $D^{-l_{\min}}$. Due to the bounds on $i(l_{\min})$, this number of packages is less than the number of packages of the same width in the optimal nodeset for bounds $l_{\min}$ and $l_{\max}-1$

(with total width $D^{l_{\min}}(n-D^{l_{\min}})/(D-1)$). Thus an optimal nodeset to the shortened problem can contain the (shifted-by-one) original nodeset and must have its maximum length achieved for all input symbols for which the original nodeset achieves maximum length. ■

Thus we can find whether $l_n = l_{\max}$ by merely doing pre-algorithmic bottom-merge Huffman coding (which, when $l_n \neq l_{\max}$, results in reduced computation). This is useful in finding a faster algorithm for large $l_{\max} - l_{\min}$ and linear φ.

VIII. A Faster Algorithm for the Linear Penalty

A somewhat different reduction, one analogous to the reduction of [33], is applicable if $\varphi(\delta) = \delta$. This more specific algorithm has similar space complexity and strictly better time complexity unless $l_{\max} - l_{\min} = O(\log n)$. However, we only sketch this approach here roughly compared to our previous explanation of the simpler, more general approach.

Consider again the code tree representation, that using a D-ary tree to represent the code. A codeword is represented by successive splits from the root to a leaf — one split for each output symbol — so that the length of a codeword is represented by the length of the path to its corresponding leaf. A vertex that is not a leaf is called an *internal vertex*; each internal vertex of the tree in Fig. 1 is shown as a black circle. We continue to use dummy variables to ensure that $n \bmod (D-1) \equiv 1$, and thus an optimal tree has $\kappa(\boldsymbol{l}) = 1$; equivalently, all internal vertices have D children. We also continue to assume without loss of generality that the output tree is monotonic. An optimal tree given the constraints of the problem will have no internal vertices at level $l_{\max}$, $(n - D^{l_{\min}})/(D-1)$ internal vertices in the $l_{\max} - l_{\min}$ previous levels, and $(D^{l_{\min}} - 1)/(D-1)$ internal vertices — with no leaves — in the levels above this, if any. The solution to a linear length-bounded problem can be expressed by the number of internal vertices in the unknown levels, that is, by

$$\alpha_i \triangleq \begin{array}{l}\text{number of internal vertices} \\ \text{in levels } [l_{\max} - i, l_{\max}]\end{array} \tag{8}$$

so that we know that

$$\alpha_0 = 0 \quad \text{and} \quad \alpha_{l_{\max}-l_{\min}} = \frac{n - D^{l_{\min}}}{D-1}.$$

If the truncated Huffman coding algorithm (as in the previous section) fails to find a code with all $l_i \leq l_{\max}$, then we are assured that there exists an $l_i = l_{\max}$, so that α_i can be assumed to be a sequence of strictly increasing integers. A strictly increasing sequence can be represented by a path on a different type of graph, a directed acyclic graph with vertices numbered 0 to $(n-D^{l_{\min}})/(D-1)$, e.g., the graph of vertices in Fig. 5. The ith edge of the path begins at α_{i-1} and ends at α_i, and each α_i represents the number of internal vertices at and below the corresponding level of the tree according to (8). Fig. 1 shows a code tree with corresponding α_i's as a count of internal vertices. The path length is identical to the height of the corresponding tree, and the path weight is

$$\sum_{i=1}^{l_{\max}-l_{\min}} w(\alpha_{i-1}, \alpha_i)$$

for edge weight function w, to be determined. Larmore and Przytycka used such a representation for binary codes [33]; here we use the generalized representation for D-ary codes.

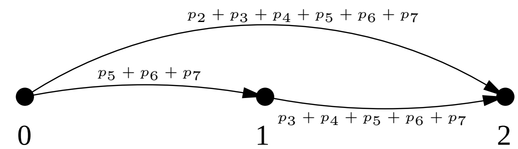

Fig. 5. The directed acyclic graph for coding $n = 7$, $D = 3$, $l_{\min} = 1$, $l_{\max} = l_n = 4$ ($\varphi(\delta) = \delta$)

In order to make this representation correspond to the above problem, we need a way of making weighted path length correspond to coding penalty and a way of assuring a one-to-one correspondence between valid paths and valid monotonic code trees. First let us define the cumulative probabilities

$$s_i \triangleq \sum_{k=n-i+1}^{n} p_k$$

so that there are $n+1$ possible values for s_i, each of which can be accessed in constant time after $O(n)$-time preprocessing. We then use these values to weigh paths such that

$$w(\alpha', \alpha'') \triangleq \begin{cases} s_{(D\alpha''^{+} - \alpha'^{+})}, & D\alpha''^{+} - \alpha'^{+} \leq n \\ \infty, & D\alpha''^{+} - \alpha'^{+} > n \end{cases}$$

where we recall that x^+ denotes $\max(x, 0)$ and ∞ is necessary for cases in which the numbers of internal vertices are incompatible; this rules out paths not corresponding to valid trees. Thus path length and penalty are equal, that is,

$$\sum_{i=1}^{l_{\max}-l_{\min}} w(\alpha_{i-1}, \alpha_i) = \sum_{j=1}^{n} p_j (l_j - l_{\min}).$$

This graph weighting has the *concave Monge property* or *quadrangle inequality*,

$$\begin{aligned} &w(\alpha', \alpha'') + w(\alpha'+1, \alpha''+1) \\ &\quad \leq w(\alpha', \alpha''+1) + w(\alpha'+1, \alpha'') \end{aligned}$$

for all $0 < \alpha' + 1 < \alpha'' \leq (n - D^{l_{\min}})/(D-1)$, since this inequality reduces to the already-assumed $p_{n-D\alpha''+\alpha'+1-D} \geq p_{n-D\alpha''+\alpha'+2}$ (where $p_i \triangleq 0$ for $i > n$). Fig. 5 shows such a graph. A single-edge path corresponds to $\boldsymbol{l} = (1, 2, 2, 2, 2, 2, 2)$ while the two-edge path corresponds to $\boldsymbol{l} = (1, 1, 2, 2, 3, 3, 3)$. In practice, only the latter would be under consideration using the algorithm in question, since the pre-algorithmic Huffman coding assured that $l_n = l_{\max} = 3$.

Thus, if

$$k \triangleq l_{\max} - l_{\min}$$

and

$$n' \triangleq 1 + \frac{n - D^{l_{\min}}}{D-1}$$

we wish to find the minimum k-link path from 0 to $(n - D^{l_{\min}})/(D-1)$ on this weighted graph of n' vertices. Given the concave Monge property, an $n' 2^{O(\sqrt{\log k \log\log n'})}$-time $O(n')$-space algorithm for solving this problem is presented in [18]. Thus the problem in question can be solved in

$n2^{O(\sqrt{\log(l_{\max}-l_{\min})\log\log n})}/D$ time and $O(n/D)$ space — $O(n)$ space if one counts the pre-algorithmic Huffman coding and/or necessary reconstruction of the Huffman code or codeword lengths — an improvement on the Package-Merge-based approach except for $k = O(\log n)$.

IX. Extensions

One might wonder whether the time complexity of the aforementioned algorithms is the minimum achievable. Special cases (e.g., $l_{\max} \approx \log_D n$ for $\varphi(\delta) = \delta$, $l_{\min} = 0$, and $D = 2$) can be addressed using modifications of the Package-Merge approach [36]–[40]. Also, $\boldsymbol{p}$ often implies ranges of values, obtainable without coding, for l_1 and l_n. This enables one to use values of $l_{\min}$ and $l_{\max}$ that result in a significant improvement, as in [19] for $l_{\min} = 0$.

An important problem that can be solved with the techniques in this paper is that of finding an optimal code given an upper bound on fringe, the difference between minimum and maximum codeword length. One might, for example, wish to find a fringe-limited prefix code in order to have a near-optimal code that can be simply implemented with minimal memory and decompression time, as in Section VIII of [43]. Such a problem is mentioned in [8, p. 121], where it is suggested that if there are $b-1$ codes better than the best code having fringe at most d, one can find this b-best code with the $O(bn^3)$-time algorithm in [44, pp. 890–891], thus solving the fringe-limited problem. However, this presumes we know an upper bound for b before running this algorithm. More importantly, if a probability vector is far from uniform, b can be very large, since the number of viable code trees is $\Theta(1.794\ldots^n)$ [45]. Thus this is a poor approach in general.

Instead, we can use the aforementioned algorithms for finding the optimal length-bounded code with codeword lengths restricted to $[l'-d, l']$ for each $l' \in \{\lceil \log_D n \rceil, \lceil \log_D n \rceil + 1, \ldots, \lfloor \log_D n \rfloor + d\}$, keeping the best of these codes; this covers all feasible cases of fringe upper bounded by d. (Here we again assume, without loss of generality, that $n \bmod (D-1) \equiv 1$.) The overall procedure thus has time complexity $O(nd^2)$ for the general convex quasiarithmetic case and $nd2^{O(\sqrt{\log d \log\log n})}/D$ when applying the algorithm of Section VIII to the most common penalty of expected length; the latter approach is of lower complexity unless $d = O(\log n)$. Both algorithms operate with only $O(n)$ space complexity.

Acknowledgments

The author wishes to thank Z. Zhang for first bringing a related problem to his attention and J.D. Morgenthaler for constructive discussions on this topic.

References

[1] C. Dickens, *A Christmas Carol*. London, UK: Chapman and Hall, 1843, available from http://www.gutenberg.org/etext/46.

[2] A. Rényi, *A Diary on Information Theory*. New York, NY: John Wiley & Sons Inc., 1987, original publication: *Naplò az információelméletről*, Gondolat, Budapest, Hungary, 1976.

[3] S. Cass, "Holiday gifts," *IEEE Spectrum*, vol. 42, no. 11, pp. 59–68, Nov. 1994, available from http://www.spectrum.ieee.org/nov05/2133/3.

[4] I. Newton, *Opticks*. London, UK: Smith and Walford, 1704, available from http://burndy.mit.edu/Collections/Babson/Online/Opticks.

[5] E. S. Schwartz, "An optimum encoding with minimum longest code and total number of digits," *Inf. Contr.*, vol. 7, no. 1, pp. 37–44, Mar. 1964.

[6] A. Moffat and A. Turpin, "On the implementation of minimum redundancy prefix codes," *IEEE Trans. Commun.*, vol. 45, no. 10, pp. 1200–1207, Oct. 1997.

[7] I. H. Witten, A. Moffat, and T. Bell, *Managing Gigabytes*, 2nd ed. San Francisco, CA: Morgan Kaufmann Publishers, 1999.

[8] J. Abrahams, "Code and parse trees for lossless source encoding," *Communications in Information and Systems*, vol. 1, no. 2, pp. 113–146, Apr. 2001.

[9] T. C. Hu and A. C. Tucker, "Optimal computer search trees and variable-length alphabetic codes," *SIAM J. Appl. Math.*, vol. 21, no. 4, pp. 514–532, Dec. 1971.

[10] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, Sept. 1952.

[11] B. McMillan, "Two inequalities implied by unique decipherability," *IRE Trans. Inf. Theory*, vol. IT-2, no. 4, pp. 115–116, Dec. 1956.

[12] L. L. Larmore, "Minimum delay codes," *SIAM J. Comput.*, vol. 18, no. 1, pp. 82–94, Feb. 1989.

[13] P. A. Humblet, "Generalization of Huffman coding to minimize the probability of buffer overflow," *IEEE Trans. Inf. Theory*, vol. IT-27, no. 2, pp. 230–232, Mar. 1981.

[14] J. Cheng, S. Dolinar, M. Effros, and R. McEliece, "Data expansion with Huffman codes," in *Proc., 1995 IEEE Int. Symp. on Information Theory*, Sept. 1995, p. 325.

[15] L. L. Larmore and D. S. Hirschberg, "A fast algorithm for optimal length-limited Huffman codes," *J. ACM*, vol. 37, no. 2, pp. 464–473, Apr. 1990.

[16] A. Itai, "Optimal alphabetic trees," *SIAM J. Comput.*, vol. 5, no. 1, pp. 9–18, Mar. 1976.

[17] L. Gotlieb and D. Wood, "The construction of optimal multiway search trees and the monotonicity principle," *Intern. J. Computer Maths, Section A*, vol. 9, no. 1, pp. 17–24, 1981.

[18] B. Schieber, "Computing a minimum-weight k-link path in graphs with the concave Monge property," *Journal of Algorithms*, vol. 29, no. 2, pp. 204–222, Nov. 1998.

[19] M. B. Baer, "Source coding for quasiarithmetic penalties," *IEEE Trans. Inf. Theory*, vol. IT-52, no. 10, pp. 4380–4393, Oct. 2006.

[20] A. De Santis and G. Persiano, "An optimal algorithm for the construction of optimal prefix codes with given fringe," in *Proc., IEEE Data Compression Conf.*, Apr. 8–11, 1991, pp. 297–306.

[21] C. Chang and J. Thomas, "Huffman algebras for independent random variables," *Disc. Event Dynamic Syst.*, vol. 4, no. 1, pp. 23–40, Feb. 1994.

[22] A. M. Garsia and M. L. Wachs, "A new algorithm for minimum cost binary trees," *SIAM J. Comput.*, vol. 6, no. 4, pp. 622–642, Dec. 1977.

[23] T. C. Hu, D. J. Kleitman, and J. K. Tamaki, "Binary trees optimum under various criteria," *SIAM J. Appl. Math.*, vol. 37, no. 2, pp. 246–256, Apr. 1979.

[24] D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, 1st ed. Reading, MA: Addison-Wesley, 1973.

[25] T. C. Hu and J. D. Morgenthaler, "Optimum alphabetic binary trees," in *Combinatorics and Computer Science*, ser. Lecture Notes in Computer Science, vol. 1120. Springer-Verlag, Aug. 1996, pp. 234–243.

[26] T. C. Hu, L. L. Larmore, and J. D. Morgenthaler, "Optimal integer alphabetic trees in linear time," in *Proc. 13th Annual European Symposium on Algorithms*. Springer-Verlag, Oct. 2005, pp. 226–237.

[27] D. E. Knuth, "Optimum binary search trees," *Acta Informatica*, vol. 1, pp. 14–25, 1971.

[28] T. C. Hu and K. C. Tan, "Path length of binary search trees," *SIAM J. Appl. Math.*, vol. 22, no. 2, pp. 225–234, Mar. 1972.

[29] M. R. Garey, "Optimal binary search trees with restricted maximal depth," *SIAM J. Comput.*, vol. 3, no. 2, pp. 101–110, June 1974.

[30] S.-L. Chan and M. J. Golin, "A dynamic programming algorithm for constructing optimal "1"-ended binary prefix-free codes," *IEEE Trans. Inf. Theory*, vol. IT-46, no. 4, pp. 1637–1644, July 2000.

[31] M. J. Golin and G. Rote, "A dynamic programming algorithm for constructing optimal prefix-free codes for unequal letter costs," *IEEE Trans. Inf. Theory*, vol. IT-44, no. 5, pp. 1770–1781, Sept. 1998.

[32] L. L. Larmore and T. M. Przytycka, "A fast algorithm for optimal height-limited alphabetic binary-trees," *SIAM J. Comput.*, vol. 23, no. 6, pp. 1283–1312, Dec. 1994.

[33] ——, "Parallel construction of trees with optimal weighted path length," in *Proc. 3nd Annual Symposium on Parallel Algorithms and Architectures*, 1991, pp. 71–80.

[34] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 1st ed. New York, NY: Wiley-Interscience, 1991.

[35] J. van Leeuwen, "On the construction of Huffman trees," in *Proc. 3rd Int. Colloquium on Automata, Languages, and Programming*, July 1976, pp. 382–410.

[36] J. Katajainen, A. Moffat, and A. Turpin, "A fast and space-economical algorithm for length-limited coding," in *Proc., Int. Symp. on Algorithms and Computation*, Dec. 1995, p. 1221.

[37] M. Liddell and A. Moffat, "Incremental calculation of optimal length-restricted codes," in *Proc., IEEE Data Compression Conf.*, Apr. 2–4, 2002, pp. 182–191.

[38] A. Moffat, A. Turpin, and J. Katajainen, "Space-efficient construction of optimal prefix codes," in *Proc., IEEE Data Compression Conf.*, Mar. 28–30, 1995, pp. 192–202.

[39] A. Turpin and A. Moffat, "Practical length-limited coding for large alphabets," *The Comput. J.*, vol. 38, no. 5, pp. 339–347, 1995.

[40] ——, "Efficient implementation of the package-merge paradigm for generating length-limited codes," in *Proc., Computing: The Australasian Theory Symposium*, Jan. 29–30, 1996, pp. 187–195.

[41] R. M. Capocelli and A. De Santis, "A note on D-ary Huffman codes," *IEEE Trans. Inf. Theory*, vol. IT-37, no. 1, pp. 174–179, Jan. 1991.

[42] I. Tomescu, "Optimum Huffman forests," *J. Universal Comput. Sci.*, vol. 3, no. 7, pp. 813–820, July 1997.

[43] M. Khosravifard, H. Saidi, M. Esmaeili, and T. A. Gulliver, "The minimum average code for finite memoryless monotone sources," *IEEE Trans. Inf. Theory*, vol. IT-53, no. 3, pp. 955–975, Mar. 2007.

[44] S. Anily and R. Hassin, "Ranking the best binary trees," *SIAM J. Comput.*, vol. 18, no. 5, pp. 882–892, Oct. 1989.

[45] D. W. Boyd, "The asymptotic number of solutions of a diophantine equation from coding theory," *J. Comb. Theory, Ser. A*, vol. 18, no. 2, pp. 210–215, Mar. 1975.