Optimal Prefix Codes for Infinite Alphabets with Nonlinear Costs

Michael B. Baer, *Member, IEEE*

Abstract—Let $P = \{p(i)\}$ **be a measure of strictly positive probabilities on the set of nonnegative integers. Although the countable number of inputs prevents usage of the Huffman algorithm, there are nontrivial** P **for which known methods find a source code that is optimal in the sense of minimizing expected codeword length. For some applications, however, a source code should instead minimize one of a family of nonlinear objective functions,** β**-exponential means, those of the form** $\log_a \sum_i p(i) a^{n(i)}$**, where** $n(i)$ **is the length of the** i**th codeword and** a **is a positive constant. Applications of such minimizations include a novel problem of maximizing the chance of message receipt in single-shot communications** $(a < 1)$ **and a previously known problem of minimizing the chance of buffer overflow in a queueing system** $(a > 1)$**. This paper introduces methods for finding codes optimal for such exponential means. One method applies to geometric distributions, while another applies to distributions with lighter tails. The latter algorithm is applied to Poisson distributions and both are extended to alphabetic codes, as well as to minimizing maximum pointwise redundancy. The aforementioned application of minimizing the chance of buffer overflow is also considered.**

Index Terms—**Communication networks, generalized entropies, generalized means, Golomb codes, Huffman algorithm, optimal prefix codes, queueing, worst case minimax redundancy.**

I. Introduction, Motivation, and Main Results

If probabilities are known, optimal lossless source coding of individual symbols (and blocks of symbols) is usually done using David Huffman's famous algorithm [1]. There are, however, cases that this algorithm does not solve. Problems with an infinite number of possible inputs — e.g., geometrically-distributed variables — are not covered. Also, in some instances, the optimality criterion — or *penalty* — is not the linear penalty of expected length. Both variants of the problem have been considered in the literature, but not simultaneously. This paper discusses cases which are both infinite and nonlinear.

An infinite-alphabet source emits symbols drawn from the alphabet $\mathcal{X}_\infty = \{0, 1, 2, \ldots\}$. (More generally, we use $\mathcal{X}$ to denote an input alphabet whether infinite or finite.) Let $P = \{p(i)\}$ be the sequence of probabilities for each symbol, so that

the probability of symbol i is $p(i) > 0$. The source symbols are coded into binary codewords. The codeword $c(i) \in \{0, 1\}^*$ in code C, corresponding to input symbol i, has length $n(i)$, thus defining length distribution N. Such codes are called *integer codes* (as in, e.g., [2]).

Perhaps the most well-known integer codes are the codes derived by Golomb for geometric distributions [3], [4], and many other types of integer codes have been considered by others [5]. There are many reasons for using such integer codes rather than codes for finite alphabets, such as Huffman codes. The most obvious use is for cases with no upper bound — or at least no known upper bound — on the number of possible items. In addition, for many cases it is far easier to come up with a general code for integers rather than a Huffman code for a large but finite number of inputs. Similarly, it is often faster to encode and decode using such well-structured codes. For these reasons, integer codes and variants of them are widely used in image and video compression standards [6], [7], as well as for compressing text, audio, and numerical data.

To date, the literature on integer codes has considered only finding efficient uniquely decipherable codes with respect to minimizing expected codeword length $\sum_i p(i) n(i)$. Other utility functions, however, have been considered for finite-alphabet codes. Campbell [8] introduced a problem in which the penalty to minimize, given some continuous (strictly) monotonic increasing *cost function* $\varphi(x) : \mathbb{R}_+ \to \mathbb{R}_+$, is

$$L(P, N, \varphi) = \varphi^{-1}\left(\sum_i p(i) \varphi(n(i))\right)$$

and specifically considered the exponential subcases with exponent $a > 1$:

$$L_a(P, N) \triangleq \log_a \sum_i p(i) a^{n(i)} \qquad (1)$$

that is, $\varphi(x) = a^x$. Note that minimizing penalty L is also an interesting problem for $0 < a < 1$ and approaches the standard penalty $\sum_i p(i) n(i)$ for $a \to 1$ [8]. While $\varphi(x)$ decreases for $a < 1$, one can map decreasing φ to a corresponding increasing function $\tilde{\varphi}(l) = \varphi_{\max} - \varphi(l)$ (e.g., for $\varphi_{\max} = 1$) without changing the penalty value. Thus this problem, equivalent to maximizing $\sum_i p(i) a^{n(i)}$, is a subset of those considered by Campbell. All penalties of the form (1) are called β-exponential means, where $\beta = \log_2 a$ [9, p. 158].

Campbell noted certain properties for β-exponential means, but did not consider applications for these means. Applications were later found for the problem with $a > 1$ [10]–[12]; these applications all relate to a buffer overflow problem discussed in Section V.

Here we introduce a novel application for problems of the form $a < 1$. Consider a situation related by Alfred Rényi, an ancient scenario in which a rebel fortress was besieged by Romans. The rebels' only hope was the knowledge gathered by a mute, illiterate spy, one who could only nod and shake his head [13, pp. 13-14]. This apocryphal tale — based upon a historical siege — is the premise behind the Hungarian version of the spoken parlor game Twenty Questions. A modern parallel in the 21st century occurred when Russian forces gained the knowledge needed to defeat hostage-takers by asking hostages "yes" or "no" questions over mobile phones [14], [15].

Rényi presented this problem in narrative form in order to motivate the relation between Shannon entropy and binary prefix coding. Note however that Twenty Questions, traditional prefix coding, and the siege scenario actually have three different objectives. In Twenty Questions, the goal is to be able to determine the symbol (i.e., the item or message) by asking at most twenty questions. In prefix coding, the goal is to minimize the expected number of questions — or, equivalently, bits — necessary to determine the message. For the siege scenario, the goal is survival; that is, assuming partial information is not useful, the besieged would wish to maximize the probability that the message is successfully transmitted within a certain window of opportunity. When this window closes — e.g., when the fortress falls — the information becomes worthless. An analogous situation occurs when a wireless device is losing power or is temporarily within range of a base station; one can safely assume that the channel, when available, will transmit at the lowest (constant) bitrate, and will be lost after a nondeterministic time period.

Assume that the duration of the window of opportunity is independent of the communicated message and is memoryless, the latter being a common assumption — due to both its accuracy and expedience — of such stochastic phenomena. Memorylessness implies that the window duration is distributed exponentially. Therefore, quantizing time in terms of the number of bits T that we can send within our window,

$$\mathbb{P}(T = t) = (1 - a)a^t, \ t = 0, 1, 2, \ldots$$

with known positive parameter $a < 1$. We then wish to maximize the probability of success, i.e., the probability that the message length does not exceed the quantized window length:

$$
\begin{aligned}
\mathbb{P}[n(X) \leq T] &= \sum_{t=0}^{\infty} \mathbb{P}(T = t) \cdot \mathbb{P}[n(X) \leq t] \\
&= \sum_{t=0}^{\infty} (1 - a)a^t \cdot \sum_{i \in \mathcal{X}} p(i) 1_{n(i) \leq t} \\
&= \sum_{i \in \mathcal{X}} p(i) \cdot (1 - a) \sum_{t=n(i)}^{\infty} a^t \\
&= \sum_{i \in \mathcal{X}} p(i) a^{n(i)} \cdot (1 - a) \sum_{t=0}^{\infty} a^t \\
&= \sum_{i \in \mathcal{X}} p(i) a^{n(i)}
\end{aligned}
$$

where $1_{n(i) \leq t}$ is 1 if $n(i) \leq t$, 0 otherwise. Minimizing (1) is an equivalent objective.

Note that this problem can be constrained or otherwise modified for the application in question. For example, in some cases, we might need some extra time to send the first bit, or, alternatively, the window of opportunity might be of at least a certain duration, increasing or reducing the probability that no bits can be sent, respectively. Thus we might have

$$\mathbb{P}(T = t) = \begin{cases} t_0, & t = 0 \\ (1 - t_0)(1 - a)a^{t-1}, & t = 1, 2, \ldots \end{cases}$$

for some $t_0 \in (0, 1)$. In this case,

$$\mathbb{P}[n(X) \leq T] = \frac{(1 - t_0)}{a} \sum_{i \in \mathcal{X}} p(i) a^{n(i)}$$

and the maximizing code is identical to that of the more straightforward case. Likewise, if we need to send multiple messages, the same code maximizes the expected number of independent messages we can send within the window, due to the memoryless property.

We must be careful regarding the meaning of an "optimal code" when there are an infinite number of possible codes under consideration. One might ask whether there must exist an optimal code or if there can be an infinite sequence of codes of decreasing penalty without any code achieving the limit penalty value. Fortunately the answer is the former, the proof being a special case of Theorem 2 in [16] (a generalization of the result for the expected-length penalty [17]). The question is then how to find one of these optimal source codes given parameter a and probability measure P.

As in the linear case, a general solution for (1) is not known for general P over a countably infinite number of events, but methods and properties for finite numbers of events — discussed in the next section — can be used to find optimal codes for certain common infinite-item distributions. In Section III, we consider geometric distributions and find that Golomb codes are optimal, although the optimal Golomb code for a given probability mass function varies according to a. The main result of Section III is that, for $p_\theta(i) = (1-\theta)\theta^i$ and $a \in \mathbb{R}_+$, Gk, the Golomb code with parameter k, is optimal, where

$$k = \max\left(1, \lceil -\log_\theta a - \log_\theta(1 + \theta) \rceil\right).$$

In Section IV, we consider distributions that are relatively light-tailed, that is, that decline faster than certain geometric distributions. If there is a nonnegative integer r such that for all $j > r$ and $i < j$,

$$p(i) \geq \max\left(p(j), \sum_{k=j+1}^{\infty} p(k) a^{k-j}\right)$$

then an optimal binary prefix code tree exists which consists of a unary code tree appended to a leaf of a finite code tree. A specific case of this is the Poisson distribution, $p_\lambda(i) = \lambda^i e^{-\lambda}/i!$, where e is the base of the natural logarithm ($e \approx 2.71828$). We show that in this case the aforementioned r is given by $r = \max(\lceil 2a\lambda \rceil - 2, \lceil e\lambda \rceil - 1)$. An application, that of minimizing probability of buffer overflow, as in [11], is

considered in Section V, where we show that the algorithm developed in [11] readily extends to coding geometric and light-tailed distributions. Section VI discusses the maximum pointwise redundancy penalty, which has a similar solution for light-tailed distributions and for which the Golomb code Gk with $k = \lceil -1/\log_2\theta \rceil$ is optimal for with geometric distributions. We conclude with some remarks on possible extensions to this work.

Throughout the following, a set or sequence of items $x(i)$ is represented by its uppercase counterpart, X. A glossary of terms is given in Appendix II.

II. BACKGROUND: FINITE ALPHABETS

If a finite number of events comprise P (i.e., $|\mathcal{X}| < \infty$), the exponential penalty (1) is minimized using an algorithm found independently by Hu *et al.* [18, p. 254], Parker [19, p. 485], and Humblet [20, p. 25], [11, p. 231], although only the last of these considered $a < 1$. (The simultaneity of these lines of research was likely due to the appearance of the first paper on adapting the Huffman algorithm to a nonlinear penalty, $\max_i(p(i) + n(i))$ for given $p(i) \in \mathbb{R}_+$, in 1976 [21].) We will use this finite-alphabet exponential-penalty algorithm in the sections that follow in order to prove optimally for infinite distributions, so let us reproduce the algorithm here:

Procedure for Exponential Huffman Coding (finite alphabets): This procedure finds the optimal code whether $a > 1$ (a minimization of the average of a growing exponential) or $a < 1$ (a maximization of the average of a decaying exponential). Note that it minimizes (1), even if the "probabilities" do not add to 1. We refer to such arbitrary positive inputs as *weights*, denoted by $w(i)$ instead of $p(i)$:

1) Each item i has weight $w(i) \in W_{\mathcal{X}}$, where $\mathcal{X}$ is the (finite) alphabet and $W_{\mathcal{X}} = \{w(i)\}$ is the set of all such weights. Assume each item i has codeword $c(i)$, to be determined later.

2) Combine the items with the two smallest weights $w(j)$ and $w(k)$ into one compound item with the combined weight $\tilde{w}(j) = a \cdot (w(j) + w(k))$. This item has codeword $\tilde{c}(j)$, to be determined later, while item j is assigned codeword $c(j) = \tilde{c}(j)0$ and k codeword $c(k) = \tilde{c}(j)1$. Since these have been assigned in terms of $\tilde{c}(j)$, replace $w(j)$ and $w(k)$ with $\tilde{w}(j)$ in $W_{\mathcal{X}}$ to form $W_{\tilde{\mathcal{X}}}$.

3) Repeat procedure, now with the remaining codewords (reduced in number by 1) and corresponding weights, until only one item is left. The weight of this item is $\sum_i w(i)a^{n(i)}$. All codewords are now defined by assigning the null string to this trivial item.

This algorithm assigns a weight to each node of the resulting implied code tree by having each item represented by a node with its parent representing the items combined into its subtree, as in Fig. 1: If a node is a leaf, its weight is given by the associated probability; otherwise its weight is defined recursively as a times the sum of its children. This concept is useful in visualizing both the coding procedure and its output.

Van Leeuwen implemented the Huffman algorithm in linear time (to input size) given sorted weights in [22], and this implementation was extended to the exponential problem in [23] as follows:

Two-Queue Implementation of Exponential Huffman Coding: The two-queue method of implementing the Huffman algorithm puts nodes/items in two queues, the first of which is initialized with the input items (eventual leaf nodes) arranged from head to tail in order of nondecreasing weight, and the second of which is initially empty. At any given step, a node with lowest weight among all nodes in both queues is at the head of one of the two queues, and thus two lowest-weighted nodes can be combined in constant time. This compound node is then inserted into (the tail of) the second queue, and the algorithm progresses until only one node is left. This node is the root of the coding tree and is obtained in linear time.

The presentation of the algorithm in [23] did not include a formal proof, so we find it useful to present one here:

Lemma 1: The two-queue method using the exponential combining rule results in an optimal exponential Huffman code given a finite number of input items.

Proof: The method is clearly a valid implementation of the exponential Huffman algorithm so long as both queues' sets of nodes remain in nondecreasing order. This is clearly satisfied prior to the first combination step. Here we show that, if nodes are in order at all points prior to a given combination step, they must be in order at the end of that step as well, inductively proving the correctness of the algorithm. It is obvious that order is preserved in the single-item queue, since nodes are only removed from it, not added to it. In the compound-node queue, order is only a concern if there is already at least one node in it at the beginning of this step, a step that combines nodes we call node i_{-1} and node i_{-2}. If so, the item at the tail of the compound-node queue at the beginning of the step was two separate items, i_{-3} and i_{-4}, at the beginning of the prior step. At the beginning of this prior step, all four items must have been distinct — i.e., corresponding to distinct sets of (possibly combined) leaf nodes — and, because the algorithm chooses the smallest two nodes to combine, neither i_{-3} nor i_{-4} can have a greater weight than either i_{-1} or i_{-2}. Thus — since $a \cdot (w(i_{-3}) + w(i_{-4})) \le a \cdot (w(i_{-1}) + w(i_{-2}))$ and the node with weight $a \cdot (w(i_{-3}) + w(i_{-4}))$ is the compound node with the largest weight in the compound-node queue at the beginning of the step in question — the queues remain properly ordered at the end of the step in question. ∎

If $a < 0.5$, the compound-node queue will never have more than one item. At each step after the first, the sole compound item will be removed from its queue since it has a weight less than the maximum weight of each of the two nodes combined to create it, which in turn is no greater than the weight of any node in the single-item queue. It is replaced by the new (sole) compound item. This extends to $a = 0.5$ if we prefer to merge combined nodes over single items of the same weight. Thus, any finite input distribution can be optimally coded for $a \le 0.5$ using a *truncated unary code*, a truncated version of the *unary code*, the latter of which has codewords of the form $\{1^j 0 : j \ge 0\}$. The truncated unary code has identical codewords as the unary code except for the longest codeword, which is of the form $\{1^{|\mathcal{X}|-1}\}$. This results from each compound node being formed using at least one single item (leaf). Taking limits, informally speaking, results in a unary limit code. Formally, this is a straightforward corollary of Theorem 2 in Section IV.

If $a > 0.5$, a code with finite penalty exists if and only if Rényi entropy of order $\alpha(a) = (1 + \log_2 a)^{-1}$ is finite, as shown in [16]. It was Campbell who first noted the connection between the optimal code's penalty, $L_a(P, N^*)$, and Rényi entropy

$$H_\alpha(P) \triangleq \frac{1}{1-\alpha} \log_2 \sum_{i \in \mathcal{X}} p(i)^\alpha$$

$$\Rightarrow H_{\alpha(a)}(P) = \frac{1 + \log_2 a}{\log_2 a} \log_2 \sum_{i \in \mathcal{X}} p(i)^{(1+\log_2 a)^{-1}}.$$

This relationship is

$$H_{\alpha(a)}(P) \leq L_a(P, N^*) < H_{\alpha(a)}(P) + 1$$

which should not be surprising given the similar relationship between Huffman-optimal codes and Shannon entropy [24], which corresponds to $a \to 1$ ($\alpha \to 1$) [8], [25]; due to this correspondence, Shannon entropy is sometimes expressed as $H_1(P)$.

III. GEOMETRIC DISTRIBUTION WITH EXPONENTIAL PENALTY

Consider the geometric distribution

$$p_\theta(i) = (1-\theta)\theta^i$$

for parameter $\theta \in (0,1)$. This distribution arises in run-length coding among other circumstances [3], [4].

For the traditional linear penalty, a Golomb code with parameter k — or Gk — is optimal for $\theta^k + \theta^{k+1} \leq 1 < \theta^{k-1} + \theta^k$. Such a code consists of a unary prefix followed by a binary suffix, the latter taking one of k possible values. If k is a power of two, all binary suffix possibilities have the same length; otherwise, their lengths $\sigma(i)$ differ by at most 1 and $\sum_i 2^{-\sigma(i)} = 1$. Binary codes such as these suffix codes are called *complete* codes. This defines the Golomb code; for example, the Golomb code for $k = 3$ is:

i	$p(i)$	$c(i)$
0	$1-\theta$	0 0
1	$(1-\theta)\theta$	0 10
2	$(1-\theta)\theta^2$	0 11
3	$(1-\theta)\theta^3$	10 0
4	$(1-\theta)\theta^4$	10 10
5	$(1-\theta)\theta^5$	10 11
6	$(1-\theta)\theta^6$	110 0
7	$(1-\theta)\theta^7$	110 10
8	$(1-\theta)\theta^8$	110 11
9	$(1-\theta)\theta^9$	1110 0
$\vdots$	$\vdots$	$\vdots$

where the space in the code separates the unary prefix from the complete suffix. In general, codeword j for Gk is of the form $\{1^{\lfloor j/k \rfloor} 0 b(j \bmod k, k) : j \geq 0\}$, where $b(j \bmod k, k)$ is a complete binary code for the $(j - k\lfloor j/k \rfloor + 1)$th of k items.

It turns out that such codes are optimal for the exponential penalty:

Theorem 1: For $a \in \mathbb{R}_+$, if

$$\theta^k + \theta^{k+1} \leq \frac{1}{a} < \theta^{k-1} + \theta^k \qquad (2)$$

for $k \geq 1$, then the Golomb code Gk is the optimal code for P_θ. If no such k exists, the unary code G1 is optimal.

Remark: This rule for finding an optimal Golomb Gk code is equivalent to

$$k = \max\left(1, \lceil -\log_\theta a - \log_\theta(1 + \theta) \rceil \right).$$

This is a generalization of the traditional linear result, which corresponds to $a \to 1$. Cases in which the left inequality is an equality have multiple solutions, as with linear coding; see, e.g., [26, p. 289]. The proof of the optimality of the Golomb code for exponential penalties is somewhat similar to that of [4], although it must be significantly modified due to the nonlinearity involved.

Before proving Theorem 1, we need the following lemma:

Lemma 2: Consider a Huffman combining procedure, such as the exponential Huffman coding procedure, implemented using the two-queue method presented in the previous section just prior to Lemma 1. Now consider a step at which the first (single-item) queue is empty, so that remaining are only compound items, that is, items representing internal nodes rather than leaves in the final Huffman coding tree. Then, in this final tree, the nodes corresponding to these compound items will be on levels differing by at most one; that is, the nodes will form a complete tree. Furthermore, if n is the number of items remaining at this point, all items that finish at level $\lceil \log_2 n \rceil$ appear closer to the head of the (second, nonempty) queue than any item at level $\lceil \log_2 n \rceil - 1$ (if any).

Proof: [Lemma 2] We use an inductive proof, in which the base cases of one and two compound items (i.e., internal nodes) are trivial. Suppose the lemma is true for every case with $n - 1$ items for $n > 2$, that is, that all nodes are at levels $\lfloor \log_2(n-1) \rfloor$ or $\lceil \log_2(n-1) \rceil$, with the latter items closer to the head of the queue than the former. Consider now a case with n nodes. The first step of coding is to merge two nodes, resulting in a combined item that is placed at the end of the combined-item queue. Because it is at the end of the queue in the reduced problem of size $n - 1$, this combined node is at level $\lfloor \log_2(n-1) \rfloor$ in the final tree, and its children are at level $1 + \lfloor \log_2(n-1) \rfloor = \lceil \log_2 n \rceil$. If n is a power of two, the remaining items end up on level $\log_2 n = \lceil \log_2(n-1) \rceil$, satisfying this lemma. If $n - 1$ is a power of two, they end up on level $\log_2(n-1) = \lfloor \log_2 n \rfloor$, also satisfying the lemma. Otherwise, there is at least one item ending up at level $\lceil \log_2 n \rceil = \lceil \log_2(n-1) \rceil$ near the head of the queue, followed by the remaining items, which end up at level $\lfloor \log_2 n \rfloor = \lfloor \log_2(n-1) \rfloor$. In any case, the lemma is satisfied for n items, and thus, inductively, for any number of items. ∎

This lemma applies to any problem in which a two-queue Huffman algorithm provides an optimal solution, including the original Huffman problem and the tree-height problem of [19]. Here we apply the lemma to the exponential Huffman algorithm to prove Theorem 1:

Proof: [Theorem 1] We start with an optimal exponential Huffman code for a sequence of similar finite weight distributions. These finite weight distributions, called *m-reduced*
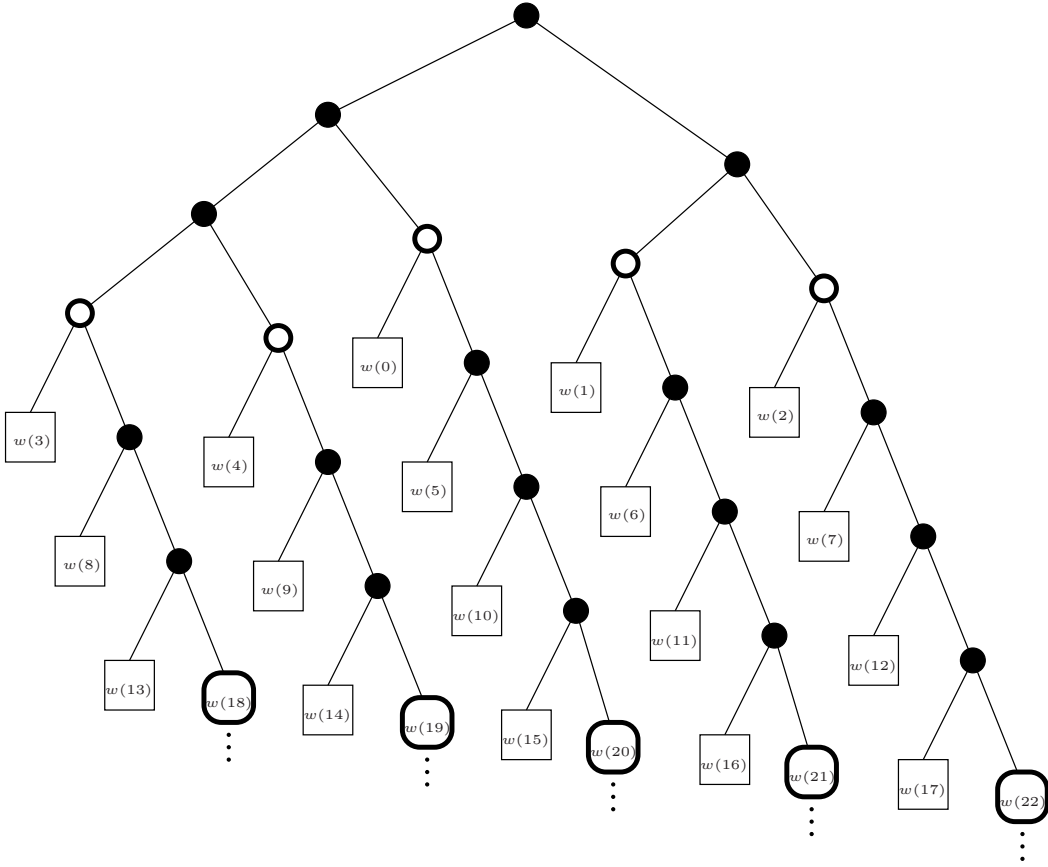
Fig. 1. Formation of a Golomb code using a code for an m-reduced source. In this illustration, $m = 17$ and $k = 5$, and smaller weights are pictorially lower. Weights are merged bottom-up, in a manner consistent with the exponential Huffman algorithm, first in separate (truncated) unary subtrees, then in a (five-leaf) complete tree.

geometric sources W_m, are defined as:

$$w_m(i) \triangleq \begin{cases} (1-\theta)\theta^i, & 0 \le i \le m \\ \dfrac{(1-\theta)a\theta^i}{1-a\theta^k}, & m < i \le m+k. \end{cases}$$

where k is as given in the statement of the theorem, or 1 if no such k exists.

Weights $w_m(0)$ through $w_m(m)$ are decreasing, as are $w_m(m+1)$ through $w_m(m+k)$. Thus we can combine the nodes with weights $w_m(m)$ and $w_m(m+k)$ if

$$\frac{(1-\theta)a\theta^{m+k}}{1-a\theta^k} \le (1-\theta)\theta^{m-1}$$

and

$$\frac{(1-\theta)a\theta^{m+k-1}}{1-a\theta^k} > (1-\theta)\theta^m \text{ or } k = 1.$$

These conditions are equivalent to the left and right sides, respectively, of (2). Thus the combined item is

$$w_{m-1}(m) = \frac{(1-\theta)a\theta^m}{1-a\theta^k}$$

and the code is reduced to the W_{m-1} case.

After merging the two smallest weights for $m = 0$, the reduced source is

$$w_{-1}(i) = \frac{(1-\theta)a\theta^i}{1-a\theta^k}, \ 0 \le i \le k-1.$$

For $k = 1$ (including all instances of the degenerate $a \le 0.5$ case and all instances in which (2) cannot be satisfied), this proves that the optimal tree is the truncated unary tree. Considering now only $k > 1$ for $m \ge k - 1$, the two-queue algorithm assures that, when the problem is reduced to weights $\{w_{-1}(i)\}$, all corresponding nodes are in the combined-item queue. Lemma 2 thus proves that these nodes form a complete code. The overall optimal tree for any m-reduced code with $m \ge k - 1$ is then a truncated Golomb tree, as pictorially represented in Fig. 1, where $m = 17$ and $k = 5$. Note that $m + 1$ is the number of leaves in common with what we call the "Golomb tree," the tree we show to be optimal for the original geometric source. The number of remaining leaves in the truncated tree is k, which is thus the number of distinct unary subtrees in the Golomb tree.

Fig. 1 represents both the truncated and full Golomb trees, along with how to merge the weights. Squares represent items to code, while circles represent other nodes of the tree. Smaller weights are below larger ones, so that items are merged as pictured. Rounded squares are items $m + 1$ through $m + k$, the items which are replaced in the Golomb tree by unary subtrees, that is, subtrees representing the unary code. Other squares are items 0 through m, those corresponding to single items in the integer code. White circles are the leaves used for the complete tree.
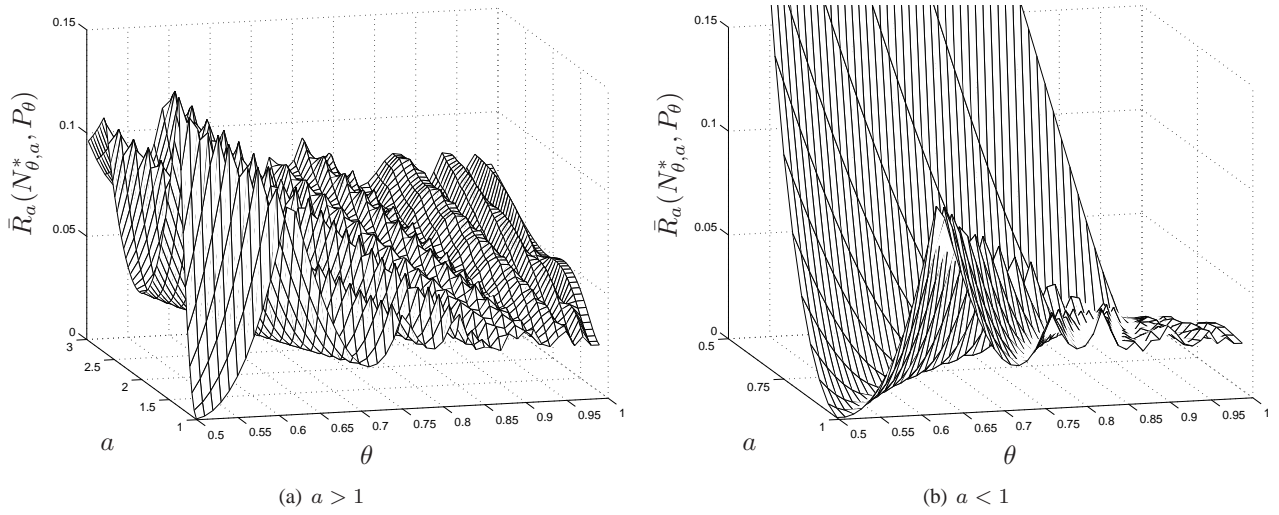
(a) $a > 1$

(b) $a < 1$

Fig. 2. Redundancy of the optimal code for the geometric distribution with the exponential penalty (parameter a). $\bar{R}_a(N_{\theta,a}^*, P_\theta) = L_a(P_\theta, N_{\theta,a}^*) - H_{\alpha(a)}(P_\theta)$, where $\alpha(a) = (1 + \log_2 a)^{-1}$, P_θ is the geometric probability sequence implied by θ, and $N_{\theta,a}^*$ is the optimal length sequence for distribution P_θ and parameter a.

It is equivalent to follow the complete portion of the code with the unary portion — as in the exponential Huffman tree in Fig. 1 — or to reorder the bits and follow the unary portion with the complete portion — as in the Golomb code [3]. The latter is more often used in practice and has the advantage of being alphabetic, that is, $i > j$ if and only if $c(i)$ is lexicographically after $c(j)$.

The truncated Golomb tree for any $m \geq k - 1$ represents a code that has the same penalty for the m-reduced distribution as does the Golomb code with the corresponding geometric distribution. We now show that this is the minimum penalty for any code with this geometric distribution.

Let $N_{\theta,a}^*$ (or N^* if there is no ambiguity) be codeword lengths that minimize the penalty for the geometric distribution (which, as we noted, exist as shown in Theorem 2 of [16]). Let N_m be codeword lengths for the m-reduced distribution found earlier; that is, $n_m(i)$ is the Golomb length for $i \leq m$ and $n_m(i) = n_m(i - k)$ for the remaining values. Finally, let N_∞ be the lengths of the code implied by $m \to \infty$, that is, the lengths of the Golomb code Gk. Then

$$\log_a \sum_{i=0}^{\infty} p(i) a^{n^*(i)} \leq \log_a \sum_{i=0}^{\infty} p(i) a^{n_\infty(i)}$$
$$= \log_a \sum_{i=0}^{m+k} w_m(i) a^{n_m(i)} \quad (3)$$
$$\leq \log_a \sum_{i=0}^{m+k} w_m(i) a^{n^*(i)}$$

where the inequalities are due to the optimality of the respective codes and the facts that $w_m(i) = p(i)$ for $i \leq m$ and

$$w_m(i) = \sum_{j=0}^{\infty} (1 - \theta) \theta^{i+jk} a^{j+1} = \sum_{j=0}^{\infty} a^{j+1} p(i + jk)$$

for $i \in (m, m + k]$. The difference between the exponent of the first and the last of the expressions in (3) is

$$\sum_{i=0}^{\infty} p(i) a^{n^*(i)} - \sum_{i=0}^{m+k} w_m(i) a^{n^*(i)}$$
$$= \sum_{i=m+1}^{\infty} p(i) a^{n^*(i)} - \sum_{i=m+1}^{m+k} w_m(i) a^{n^*(i)}.$$

As $m \to \infty$ for $m \geq k - 1$, the sums on the right-hand side approach 0; the first is the difference between a limit (an infinite sum) and its approaching sequence of finite sums, all upper bounded in (3), and each of the terms in the second summation is upper-bounded by a multiplicative constant of the corresponding term in the first. (In the latter finite summation, terms are 0 for $i > m + k$.) Their difference therefore also approaches zero, so the summations on the left-hand side approach equality, as do those in (3), and the Golomb code must be optimal. ∎

It is equivalent for the bits of the unary portion to be complemented, that is, to use $\{0^{\lfloor j/k \rfloor} 1 b(j \bmod k, k) : j \geq 0\}$ (as in [4]) instead of $\{1^{\lfloor j/k \rfloor} 0 b(j \bmod k, k) : j \geq 0\}$ (as in [3]). It is also worth noting that Golomb originally proposed his code in the context of a spy reporting run lengths; this is similar to Rényi's context for communications, related in Section I as a motivation for the nonlinear penalty with $a < 1$.

A little algebra reveals that, for a distribution P_θ and a Golomb code with parameter k (lengths N_k),

$$L_a(P_\theta, N_k) = \log_a \sum_{i=0}^{\infty} (1 - \theta) \theta^i a^{\left(\lceil \frac{i+1-z}{k} \rceil + g\right)}$$
$$= g + \log_a \left(1 + \frac{(a-1)\theta^z}{1 - a\theta^k}\right) \quad (4)$$

where $g = \lfloor \log_2 k \rfloor + 1$ and $z = 2^g - k$. Therefore, Theorem 1 provides the k that minimizes (4). If $a > 0.5$, the corresponding Rényi entropy is

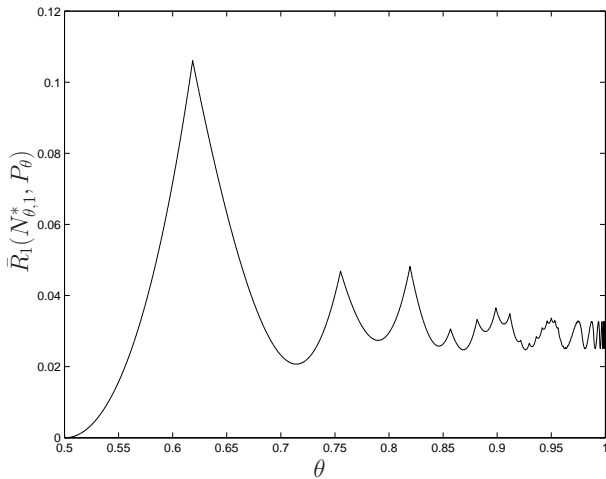$$H_{\alpha(a)}(P_\theta) = \log_a \frac{1 - \theta}{(1 - \theta^{\alpha(a)})^{1/\alpha(a)}} \quad (5)$$

Fig. 3. Redundancy of the optimal code for the geometric distribution with the traditional linear penalty.

where we recall that $\alpha(a) = (1+\log_2 a)^{-1}$. (Again, $a \le 0.5$ is degenerate, an optimal code being unary with no corresponding Rényi entropy.)

In evaluating the effectiveness of the optimal code, one might use the following definition of *average pointwise redundancy* (or just *redundancy*):

$$\bar{R}_a(N, P) \triangleq L_a(P, N) - H_{\alpha(a)}(P).$$

For nondegenerate values, we can plot the $\bar{R}_a(N_{\theta,a}^*, P_\theta)$ obtained from the minimization. This is done for $a > 1$ and $a < 1$ in Fig. 2. Note that as $a \to 1$, the plot approaches the redundancy plot for the linear case, e.g., [4], reproduced as Fig. 3.

In many potential applications of nonlinear penalties — such as the aforementioned for $a > 1$ [10]–[12] and $a < 1$ (Section I) — a is very close to 1. Since the preceding analysis shows that the Golomb code that is optimal for given a and θ is optimal not only for these particular values, but for a range of a (fixing θ) and a range of θ (fixing a), the Golomb code for the traditional linear penalty is, in some sense, much more robust and general than previously appreciated.

IV. OTHER INFINITE SOURCES

Abrahams noted that, in the linear case, slight deviation from the geometric distribution in some cases does not change the optimal code [27, Proposition (2)]. Other extensions to and deviations of the geometric distribution have also been considered [28]–[30], including optimal codes for nonbinary alphabets [27], [29]. Many of these approaches can be adapted to the nonlinear penalties considered here. However, in this section we instead consider another type of probability distribution for binary coding, the type with a light tail.

Humblet's approach [31], later extended in [32], uses the fact that there is an optimal code tree with a unary subtree for any probability distribution with a relatively light tail, one for which there is an r such that, for all $j > r$ and $i < j$, $p(i) \ge p(j)$ and $p(i) \ge \sum_{k=j+1}^{\infty} p(k)$. Due to the additive nature of Huffman coding, items beyond r form the unary

subtree, while the remaining tree can be coded via the Huffman algorithm. Once again, this has to be modified for exponential penalties.

We wish to show that the optimal code can be obtained when there is a nonnegative integer r such that, for all $j > r$ and $i < j$,

$$p(i) \ge \max\left(p(j), \sum_{k=j+1}^{\infty} p(k)a^{k-j} \right).$$

The optimal code is obtained by considering the reduced alphabet consisting of symbols $0, 1, \ldots, r+1$ with weights

$$w(i) = \begin{cases} p(i), & i \le r \\ \sum_{k=r+1}^{\infty} p(k)a^{k-r}, & i = r+1. \end{cases} \tag{6}$$

Apply exponential Huffman coding to this reduced set of weights. For items 0 through r, the Huffman codewords for the reduced and the infinite alphabets are identical. Each other item $i > r$ has a codeword consisting of the reduced codeword for item $r+1$ (which, without loss of generality, consists of all 1's) followed by the unary code for $i-r-1$, that is, $i-r-1$ ones followed by a zero. We call such codes *unary-ended*. A pictorial example is shown in Fig. 4 for a problem instance for which $r = 12$.

Theorem 2: Let $p(\cdot)$ be a probability measure on the set of nonnegative integers, and let a be the parameter of the penalty to be optimized. If there is a nonnegative integer r such that for all $j > r$ and $i < j$,

$$p(i) \ge p(j) \tag{7}$$

and

$$p(i) \ge \sum_{k=j+1}^{\infty} p(k)a^{k-j} \tag{8}$$

then there exists a minimum-penalty binary prefix code with every codeword $j > r$ consisting of $j-x$ 1's followed by one 0 for some fixed nonnegative integer x.

Proof: The idea here is similar to that for geometric distributions, to show a sequence of finite codes which in some sense converges to the optimal code for the infinite alphabet. In this case we consider the infinite sequence of codes implicit in the above; for a given $m \ge -1$, the corresponding codeword weights are

$$w_m(i) = \begin{cases} p(i), & i < r+m+2 \\ \sum_{k=r+m+2}^{\infty} p(k)a^{k-r-m-1}, & i = r+m+2. \end{cases}$$

It is obvious that an optimal code for each m-reduced distribution is identical to the proposed code for the infinite alphabet, except for the item $r+m+2$, which is the code tree sibling of item $r+m+1$.

For $a < 1$, we show, as in the geometric case, that the difference between the penalties for the optimal and the proposed codes approaches 0. In this case, the equivalent of
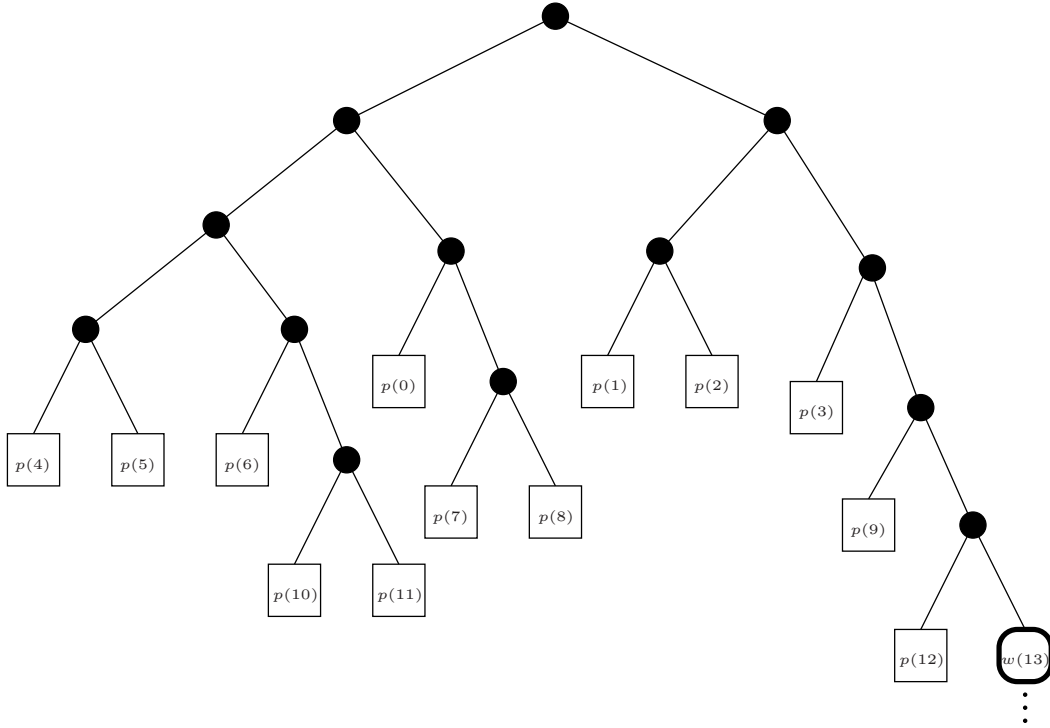
Fig. 4. Formation of a unary-ended infinite code using a Huffman-like code. (Smaller weights are pictorially lower.) Weights are merged bottom-up, in a manner consistent with the exponential Huffman algorithm, first in the (truncated) unary subtree, then as in the exponential Huffman algorithm.

inequality (3) is

$$
\begin{aligned}
\log_a \sum_{i=0}^{\infty} p(i) a^{n^*(i)} &\leq \log_a \sum_{i=0}^{\infty} p(i) a^{n_\infty(i)} \\
&= \log_a \sum_{i=0}^{r+m+2} w_m(i) a^{n_m(i)} \quad (9) \\
&\leq \log_a \sum_{i=0}^{r+m+2} w_m(i) a^{n^*(i)}
\end{aligned}
$$

where in this case $n_\infty(i)$ denotes a codeword of the proposed code, $n_m(i) = n_\infty(i)$ for $i < r+m+2$ and $n_m(i) = n_\infty(i-1)$ for $i = r+m+2$, and, again, $n^*(\cdot)$ denotes the lengths of codewords in an optimal code. The corresponding difference between the exponent of the first and the last expressions of (9) is

$$
\begin{aligned}
&\sum_{i=0}^{\infty} p(i) a^{n^*(i)} - \sum_{i=0}^{r+m+2} w_m(i) a^{n^*(i)} \\
&= \sum_{i=r+m+2}^{\infty} p(i) a^{n^*(i)} - w_m(r+m+2) a^{n^*(r+m+2)}.
\end{aligned}
\tag{10}
$$

As $m \to \infty$, both terms in the difference on the second line of (10) clearly approach 0, so the terms in (9) approach equality, showing the proposed code to be optimal.

For $a > 1$, the same method will work, but it is not so obvious that the terms in the difference on the second line of (10) approach 0. Let us first find an upper bound for $w_m(r+$

$m+2)$ in terms of $p(r+m+2)$:

$$
\begin{aligned}
w_m(r+m+2) &= a p(r+m+2) + a^2 p(r+m+3) + \\
&\quad \sum_{i=r+m+4}^{\infty} p(i) a^{i-r-m-1} \\
&\leq (a^2 + a) p(r+m+2) + a^2 p(r+m+3) \\
&\leq (2a^2 + a) p(r+m+2)
\end{aligned}
$$

where the first equality is due to the definition of $w_m(\cdot)$, the first inequality due to (8), and the second inequality due to (7). Thus $w_m(r+m+2)$ has an upper bound of $(2a^2+a)p(r+m+2)$ for all $m \geq -1$. In addition, since the proposed code has a finite penalty — identical to that of any reduced code — the optimal code has a finite penalty, and the sequence of its terms — each one of which has the form $p(r+m+2)a^{n^*(r+m+2)}$ — approaches 0 as m increases. Thus $w_m(r+m+2)a^{n^*(r+m+2)}$ approaches 0 as well. Due to the optimality of $n^*(\cdot)$, $w_m(r+m+2)a^{n^*(r+m+2)}$ serves as an upper bound for $\sum_{i=r+m+2}^{\infty} p(i) a^{n^*(i)}$, and thus both terms approach 0. As with $a < 1$, then, the terms in (9) approach equality for $m \to \infty$, showing the proposed code to be optimal. ∎

The rate at which $p(\cdot)$ must decrease in order to satisfy condition (8) clearly depends on a. One simple sufficient condition — provable via induction — is that it satisfy $p(i) \geq a p(i+1) + a p(i+2)$ for large i. A less general condition is that $p(i)$ eventually decrease at least as quickly as g^i where $g = (\sqrt{1+4/a} - 1)/2$, the same ratio needed for a unary geometric code for $\theta = g$, as in (2). The ratio g is plotted in Fig. 5.

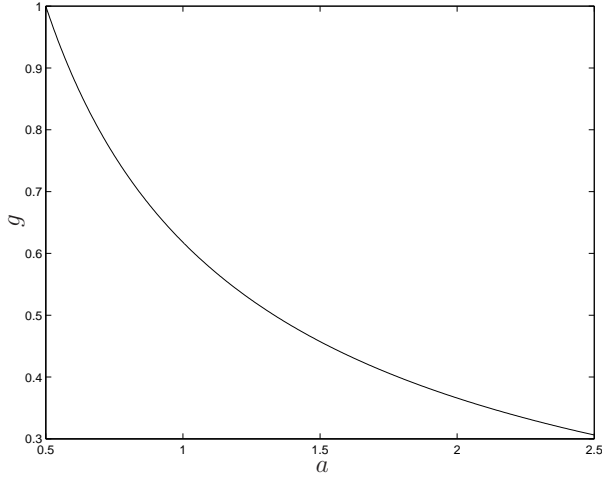For $a \to 1$, these conditions approach those derived in [31].

Fig. 5. Ratio g, probability distribution fall-off sufficient for the optimality of a unary-ended code. Note that $1/g = \Phi \triangleq \frac{1}{2}(1 + \sqrt{5})$, the golden ratio, at $a = 1$.

The stronger results of [32] do not easily extend here due to the nonadditivity of the exponential penalty. An attempt at such an extension in [33, pp. 103–105] gives no criteria for success, so that, while one could produce certain codewords for certain codes, one might fail in producing other codewords for the same codes or for other codes. Thus this extension is not truly a workable algorithm.

Consider the example of optimal codes for the Poisson distribution,

$$p_\lambda(i) = \frac{\lambda^i e^{-\lambda}}{i!}.$$

How does one find a suitable value for r (as in Section IV) in such a case? It has been shown that $r \geq \lceil e\lambda \rceil - 1$ yields $p(i) \geq p(j)$ for all $j > r$ and $i < j$, satisfying the first condition of Theorem 2 [31]. Moreover, if, in addition, $j \geq \lceil 2a\lambda \rceil - 1$ (and thus $j > a\lambda - 1$), then

$$
\begin{aligned}
\sum_{k=1}^{\infty} p(j+k)a^k &= \frac{e^{-\lambda}\lambda^j}{j!}\left[\frac{a\lambda}{j+1} + \frac{a^2\lambda^2}{(j+1)(j+2)} + \cdots\right] \\
&< p(j)\left[\frac{a\lambda}{j+1} + \frac{a^2\lambda^2}{(j+1)^2} + \cdots\right] \\
&= p(j)\frac{\frac{a\lambda}{j+1}}{1 - \frac{a\lambda}{j+1}} \\
&\leq p(j) \\
&\leq p(i).
\end{aligned}
$$

Thus, since we consider $j > r$, $r = \max(\lceil 2a\lambda \rceil - 2, \lceil e\lambda \rceil - 1)$ is sufficient to establish an r such that the above method yields the optimal infinite-alphabet code.

In order to find the optimal reduced code, use

$$w_{-1}(r+1) = \sum_{k=r+1}^{\infty} p(k)a^{k-r} = a^{-r}e^{\lambda(a-1)} - \sum_{k=0}^{r} p(k)a^{k-r}.$$

For example, consider the Poisson distribution with $\lambda = 1$. We code this for both $a = 1$ and $a = 2$. For both values, $r = 2$, so both are easy to code. For $a = 1$, $w_{-1}(3) = 1 - 2.5e^{-1} \approx 0.0803\ldots$, while, for $a = 2$, $w_{-1}(3) = 0.25e - 1.25e^{-1} \approx$

0.2197.... After using the appropriate Huffman procedure on each reduced source of 4 weights, we find that the optimal code for $a = 1$ has lengths $N = \{1, 2, 3, 4, 5, 6, \ldots\}$ — those of the unary code — while the optimal code for $a = 2$ has lengths $N = \{2, 2, 2, 3, 4, 5, \ldots\}$.

It is worthwhile to note that these techniques are easily extensible to finding an optimal alphabetic code — that is, one with $c(i)$'s arranged in lexicographical order — for $a > 1$. One needs only to find the optimal alphabetic code for the reduced code with weights given in equation (6), as in [18], with codewords for $i > r$ consisting of the reduced code's codeword for $r+1$ followed by $i-r-1$ ones and one zero. As previously mentioned, Golomb codes are also alphabetic and thus are optimal alphabetic codes for the geometric distribution.

V. APPLICATION: BUFFER OVERFLOW

The application of the exponential penalty in [11] concerns minimizing the probability of a buffer overflowing. It requires that each candidate code for overall optimality be an optimal code for one of a series of exponential parameters (a's where $a > 1$). An iterative approach yields a final output code by noting that, for the overall utility function, each candidate code is no worse than its predecessor, and there are a finite number of possible candidate codes. Therefore, eventually a candidate code yields the same value as the prior candidate code, and this can be shown to be the optimal code. This application of exponential Huffman coding can, using the above techniques, be extended to infinite alphabets.

In the application, integers with a known distribution P arrive with independent intermission times having a known probability density function. Encoded bits are sent at a given rate, with bits to be sent waiting in a buffer of fixed size. Constant b represents the buffer size in bits, random variable T represents the probability distribution of source integer intermission times measured in units of encoded bit transmission time, and function $A(s)$ is the Laplace-Stieltjes transform of T, $\mathbb{E}[e^{-sT}]$.

When the integers are coded using $N = \{n(i)\}$, the probability per input integer of buffer overflow is of the order of $e^{-s^* b}$, where s^* is the largest s such that

$$f(N, s) \leq 1$$

where

$$f(N, s) \triangleq A(s)\sum_{i=0}^{\infty} p(i)e^{sn(i)}. \tag{11}$$

The previously known algorithm to maximize s^* is as follows:

Procedure for Finding Code with Largest s^* [11]
1) Choose any $s_0 \in \mathbb{R}_+$.
2) $j \leftarrow 0$.
3) $j \leftarrow j + 1$.
4) Find codeword lengths N_j minimizing $\sum_i p(i)e^{s_{j-1}n(i)}$.
5) Compute $s_j \triangleq \max\{s \in \mathbb{R} : f(N_j, s) \leq 1\}$.
6) If $s_j \neq s_{j-1}$ then go to step 3; otherwise stop.

We can use the above methods in order to accomplish step 4, but we still need to examine how to modify steps 1 and 5 for an infinite input alphabet.

First note that, unlike in the finite case, $s^* < \infty$, that is, there always exists an $s^* \in \mathbb{R}_+$ such that, for all $s > s^*$, $f(N, s) > 1$. For any stable system, the buffer cannot receive integers more quickly than it can transit bits, so there is a positive probability that $\mathbb{P}[T \geq 1]$. Thus the Laplace-Stieltjes transform $A(s)$ exceeds $c_1 e^{-s}$ for some constant $c_1 > 0$. Also, without loss of generality, we can assume that $p(i)$ is monotonic nonincreasing and an optimal $n(i)$ is monotonic nondecreasing. This monotonicity means that $n(i) \geq \log_2 i$, and there is no exponential base a_0 and offset constant c_2 for which $\sum_{i=0}^{\infty} p(i)e^{sn(i)} \leq a_0^{s+c_2}$ for all $s \in \mathbb{R}_+$. Thus the summation in (11) must increase superexponentially, and, multiplying the $A(s)$ and summation terms, there is an s such that $f(N, s) > 1$ for $s > s^*$.

For step 1, the initial guess proposed in [11] is an upper bound for all possible values of s^*. The Rényi entropy of P is used to find an initial guess using

$$A(s)\left(\sum_{i=0}^{\infty} p(i)^{\frac{1}{1+\log_2 e^s}}\right)^{1+\log_2 e^s} \leq A(s)\sum_{i=0}^{\infty} p(i)e^{sn(i)}, \quad (12)$$

and choosing s_0 as the largest s such that the left term of (12) is no greater than one. Thus, $s_0 \geq s^*$ for any value of s^* corresponding to step 5.

This technique is well-suited to a geometric distribution, for which entropy has the closed form shown in equation (5), so

$$A(s) \cdot \frac{1 - \theta}{\left(1 - \theta^{(1+\log_2 e^s)^{-1}}\right)^{1+\log_2 e^s}} \leq f(N, s).$$

However, a general distribution with a light tail, such as the Poisson distribution, might have no closed form for this bound. One solution to this is to use more relaxed lower bounds on the sum — such as using a partial sum with a fixed number of terms — yielding looser upper bounds for s^*. Another approach would be to note that, because of the light tail, the infinite sum can usually be quickly calculated to the precision of the architecture used. Note, however, that no matter what the technique, the bound must be chosen so that s_0 is an real number and not infinity. Partial sums may be refined to accomplish this.

In calculating $f(N, s)$ for use in step 5, the geometric distribution has the closed-form value for f obtainable from equation (4), while the other distributions must instead rely on approximations of f. As before, this is easily done due to the light tail of the distribution. Alternatively, a partial sum and a geometric approximation can be used to bound $f(N, s)$ and thus s^*, and these two bounds used to find two codes. If the two codes are identical, the algorithm may proceed; otherwise, we must roll back to the summation and improve the bounds until the codes are identical.

These variations make the steps of the algorithm possible, but the algorithm itself must also be proven correct with the variations.

Theorem 3: Given a geometric distribution or an input distribution satisfying the conditions of Theorem 2 for $a = e^{s_0}$, where s_0 is an upper-bound on s^*, the above Procedure for Finding Code with Largest s^* terminates with an optimal code.

Proof: The number of codes that can be generated in the course of running the algorithm should be bounded so that the algorithm is guaranteed to terminate. Optimality for the algorithm then follows as for the finite case [11]. As in the finite case, $s_{j+1} \geq s_j$ for $j \geq 1$ (but not $j = 0$) due to step 5 [$f(N_j, s_j) \leq 1$], step 4 [$f(N_{j+1}, s_j) \leq f(N_j, s_j)$], and the definition of s_{j+1}.

In the case of a geometric distribution, each N_j is a Golomb code Gk_j for some positive integer k_j. Clearly, if we choose s_0 as detailed above, it is the greatest value of s_j, being either optimal or unachievable due to its derivation as a bound of the problem. Since Gk_i (with lengths N_i) is the optimal code for the code with exponential base $a = e^{s_{i-1}}$, (2) means that $\theta^{k_i} + \theta^{k_i+1} \leq e^{-s_{i-1}} < \theta^{k_i-1} + \theta^{k_i}$, and thus

$$(1 + \theta)\theta^{k_1} \leq e^{-s_0} \leq e^{-s_{j-1}} < (1 + \theta)\theta^{k_j-1}$$

and, since $\theta < 1$, we have $k_j - 1 < k_1$ (or, equivalently, $k_j \leq k_1$) for all $j \geq 1$. Therefore, there are only k_1 possible codes the algorithm can generate.

In the case of a distribution with a lighter tail, the minimum r of Theorem 2 increases with each iteration after the first, and the first r_1 (corresponding to s_0) upper bounds the remaining r_i. Thus all candidate codes can be specified by their first r_1 codeword lengths, none of which is greater than r_1. The number of codes is then bounded for both cases, and the algorithm terminates with the optimal code. ∎

VI. REDUNDANCY PENALTIES

It is natural to ask whether the above results can be extended to other penalties. One penalty discussed in the literature is that of maximal pointwise redundancy [34], which is

$$R^*(N, P) \triangleq \sup_{i \in \mathcal{X}}[n(i) + \log_2 p(i)]$$

where we use sup when we are not assured the existence of a maximum. This can be shown to be a limit of the exponential case, as in [23], allowing us to analyze its minimization using the same techniques as exponential Huffman coding. This limit can be shown by defining *dth exponential redundancy* as follows:

$$
\begin{aligned}
R_d(N, P) &\triangleq \frac{1}{d}\log_2 \sum_{i \in \mathcal{X}} p(i)2^{d(n(i)+\log_2 p(i))} \\
&= \frac{1}{d}\log_2 \sum_{i \in \mathcal{X}} p(i)^{1+d}2^{dn(i)}.
\end{aligned}
$$

Thus $R^*(N, P) = \lim_{d\to\infty} R_d(N, P)$, and the above methods should apply in the limit. In particular:

Theorem 4: The Golomb code Gk for $k = \lceil -1/\log_2 \theta \rceil$ is optimal for minimizing maximal pointwise redundancy for P_θ.

Proof:
Case 1: Consider first when $-1/\log_2 \theta$ is not an integer. We show that $k = \lceil -1/\log_2 \theta \rceil$ is optimal by finding a D such that, for all $d > D$, the optimal code for the dth exponential redundancy penalty is Gk. For a fixed d, (2) implies that such a code should satisfy

$$(\theta^{1+d})^k + (\theta^{1+d})^{k+1} \leq \frac{1}{2^d} < (\theta^{1+d})^{k-1} + (\theta^{1+d})^k, \quad (13)$$
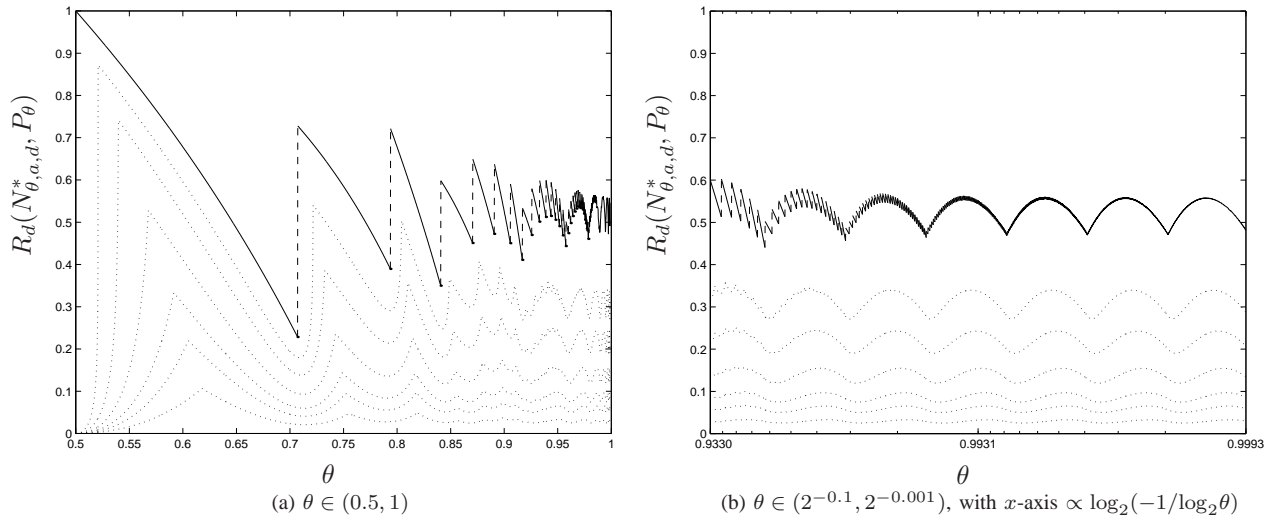
Fig. 6. Maximal pointwise redundancy of the optimal maximal redundancy code for the geometric distribution, solid (with discontinuities represented by dashed); optimal dth exponential redundancy for the geometric distribution, dotted for $d = \{1, 2, 4, 16, 256, 65536\}$, from lowest to highest.

and thus we wish to show that this holds for all $d > D$. Consider $k = \lceil -1/\log_2\theta \rceil$. Clearly, $k > -1/\log_2\theta$, or, equivalently,

$$\theta^k < \frac{1}{2}. \quad (14)$$

Now consider

$$D = -1 + \frac{1}{1 + (k-1)\log_2\theta}$$

so that $(k-1)\log_2\theta \in (-1, 0]$ and therefore $D \geq 0$. Taken together with the fact that $\theta \in (0, 1)$, (14) yields $\theta^{dk} < 2^{-d}$ and $(1 + \theta^{1+d})\theta^k < 2\theta^k < 1$. Multiplication yields the left-hand side of (13) for any $d > D$. For any such d, algebra easily shows that we also have the inequality $(2\theta^{k-1})^{1+d} \geq 2$, yielding

$$\begin{aligned}\left[(\theta^{1+d})^{k-1} + (\theta^{1+d})^k\right] 2^d &= \frac{1}{2}(2\theta^{k-1})^{1+d} + \frac{1}{2}(2\theta^k)^{1+d} \\ &= \frac{1}{2}(2\theta^k)^{1+d}(\theta^{-1-d} + 1) \\ &= \frac{1}{2}(2\theta^{k-1})^{1+d}(1 + \theta^{1+d}) \\ &> 1.\end{aligned}$$

This is equivalent to the right-hand side of inequality (13) for the values implied by the definition of $R_d(N, P)$. Then Gk is an optimal code for $d > D$, and thus for the limit case of maximal pointwise redundancy.

Case 2: Now consider when $-1/\log_2\theta$ is an integer. It should be noted that, for the traditional (linear) penalty, these are precisely the k values that Golomb considered in his original paper [3] and that they are local infima for the minimum maximal pointwise redundancy function in θ, as in Fig. 6. Here we show they are local minima.

Since $\theta = 0.5$ is a dyadic probability distribution and thus trivial, we can assume that $\theta > 0.5$. We wish to show that optimality is preserved in these right limits of Case 1. Note that, for each i with fixed N,

$$\lim_{\theta'\uparrow\theta}[n(i) + \log_2 p_{\theta'}(i)] = n(i) + \log_2 p_\theta(i).$$

This is of particular interest for the value of i maximizing pointwise redundancy for Gk at θ', where $\theta' \in (\theta^{1/\log_2 2\theta}, \theta)$, allowing us to use the right limit of θ. Let $i^{**} \triangleq 2^{\lceil \log_2 k \rceil} - k$, the smallest i which has codeword length exceeding the codeword length for item 0. Clearly the pointwise redundancy for this value is greater than that for all items with $i < i^{**}$, since they are one bit shorter but not more than twice as likely. Similarly, items in (i^{**}, k) have identical length but lower probability, and thus smaller redundancy. For items with $i \geq k$, note that the redundancy of items in the sequence $\{j, j+k, j+2k, \ldots\}$ for any j must be nonincreasing because the difference in redundancy is constant yet redundancy is upper-bounded by the maximum. Thus i^{**} maximizes pointwise redundancy for Gk at θ'.

We know the pointwise redundancy of i^{**} for Gk at θ, although we have yet to show that i^{**} yields the maximal pointwise redundancy for Gk at θ or that Gk minimizes maximal pointwise redundancy. However, for any code, including the optimal code, as a result of pointwise continuity,

$$\begin{aligned}\sup_{i \in \mathcal{X}_\infty}[n(i) + \log_2 p_\theta(i)] &\geq n(i^{**}) + \log_2 p_\theta(i^{**}) \\ &= \lim_{\theta'\uparrow\theta}[n(i^{**}) + \log_2 p_{\theta'}(i^{**})].\end{aligned}$$

From the above discussion, it is clear that the right-hand side is minimized by the Golomb code with $k = -1/\log_2\theta$, so, because the left-hand side achieves same value with this code, the left-hand side is indeed minimized by Gk. Thus this code minimizes maximal pointwise redundancy for θ. The corresponding maximal pointwise redundancy is

$$\begin{aligned}\max_i&[n_\theta^{**}(i) + \log_2 p_\theta(i)] \\ &= n_\theta^{**}(2^{\lceil \log_2 k \rceil} - k) + \log_2 p_\theta(2^{\lceil \log_2 k \rceil} - k) \\ &= \lceil \log_2 k \rceil + 1 + \log_2(1 - \theta) + (2^{\lceil \log_2 k \rceil} - k)\log_2\theta\end{aligned}$$

where $N_\theta^{**} = \{n_\theta^{**}(i)\}$ is defined as the lengths of a code minimizing maximal pointwise redundancy. Note that this is the redundancy for all items $i = 2^{\lceil \log_2 k \rceil} + jk$ with integer $j \geq -1$. ∎

It is worthwhile to observe the behavior of maximal pointwise redundancy in a fixed (not necessarily optimal) Golomb code with length distribution N_k. The maximal pointwise redundancy

$$R^*(N_k, P_\theta) = \sup_{i \in \mathcal{X}_\infty} [n_k(i) + \log_2 p_\theta(i)]$$

decreases with increasing θ — and is an optimal code for $\theta \in (2^{-1/(k-1)}, 2^{-1/k}]$ — until θ exceeds $2^{-1/k}$, after which there is no maximum, that is, pointwise redundancy is unbounded. This explains the discontinuous behavior of minimum maximal redundancy for an optimal code as a function of θ, illustrated in Fig. 6, where each continuous segment corresponds to an optimal code for $\theta \in (2^{-1/(k-1)}, 2^{-1/k}]$.

Note also the oscillating behavior as $\theta \uparrow 1$. We show in Appendix I that $\liminf_{\theta \uparrow 1} R^*(N_\theta^{**}, P_\theta) = 1 - \log_2 \log_2 e$ and $\limsup_{\theta \uparrow 1} R^*(N_\theta^{**}, P_\theta) = 2 - \log_2 e$, and we characterize this oscillating behavior. This technique is extensible to other redundancy scenarios of the kind introduced in [23].

For distributions with light tails, one can use a technique much like the technique of Theorem 2 in Section IV. First note that this requires, as a necessary step, the ability to construct a minimum maximal pointwise redundancy code for finite alphabets. This can be done either with the method in [34] or any of those in [23], the simplest of which uses a variant of the tree-height problem [19], solved via a different extension of Huffman coding. Simply put, the weight combining rule, rather than $w(j) + w(k)$ or $a \cdot (w(j) + w(k))$, is

$$\tilde{w}(j) = 2 \cdot \max(w(j), w(k)). \tag{15}$$

This rule is used to create an optimal code with lengths $N^{(r)}$ for $W^{(r)} \triangleq \{p(0), p(1), \ldots, p(r), 2p(r+1)\}$, assuming a unary subtree for items with index $i \geq r$ (and no other items) is part of an optimal code tree. As in the coding method corresponding to Theorem 2, the codewords for items 0 through r of this reduced code are identical to those of the infinite alphabet. Each other item $i > r$ has a codeword consisting of the reduced codeword for $r + 1$ followed by the unary code for $i - r - 1$, that is, $i - r - 1$ ones followed by a zero.

A sufficient condition for using this method is finding an r such that

$$\text{for all } i < r, \ p(i) \geq p(r)$$

and

$$\text{for all } j \geq r, \ p(j) \geq 2p(j+1).$$

For such j, pointwise redundancy is nonincreasing along a unary subtree, as

$$\begin{aligned} n(j) + \log_2 p(j) &= n(j+1) + \log_2(p(j)/2) \\ &\geq n(j+1) + \log_2 p(j+1). \end{aligned}$$

The aforementioned coding method works because, for each j, an optimal subtree consisting of the items with index $i \geq j$ and higher has $n(i) = n(j) - j + i$; this subtree is optimal because the weight of the root node of *any* subtree cannot be less than $2p(j)$. A formal proof, similar to that of Theorem 2, is omitted in the interest of space.

For a Poisson random variable, $r = \lceil e\lambda \rceil - 1$ satisfies this condition, since, for $i < r \leq j$, $p(i) \geq p(r)$ (as in [31]), and

$$p(j) = \frac{j+1}{\lambda} p(j+1) \geq \frac{r+1}{\lambda} p(j+1) \geq ep(j+1) > 2p(j+1).$$

Thus such a random variable can be coded in this manner.

Note that other sufficient conditions can be obtained through alternative methods. One simple rule is that any code for which $p(i) \leq 2^{-i} p(0)$ for all $i > 0$ will necessarily have $n(0) + \log_2 p(0)$ minimized by letting $n(0) = 1$, and this will be the maximum redundancy if $n(i) = i - 1$ in general. For example, a unary tree optimizes $P = \{0.6, 0.15, 0.15, 0.0375, 0.0375, \ldots\}$, since $\log_2 1.2 \approx 0.263$ is a lower bound on maximal pointwise redundancy for any code given $p(1) = 0.6$, and this bound is achieved for the unary code. If viewed as a rule for a unary subtree, this is looser than the above, since, unlike linear and exponential penalties, not all subtrees of the subtree need be optimal. Other relaxations can be obtained, although, as they are usually not needed, we do not discuss them here.

VII. CONCLUSION

The aforementioned methods for coding integers are applicable to geometric and light-tailed distributions with exponential and related penalties. Although they are not direct applications of Huffman coding, per se, these methods are derived from the properties of generalizations of the Huffman algorithm. This allows examination of subtrees of a proposed optimal code independently of the rest of the code tree, and thus specification of finite codes which in some sense converge to the optimal integer code. Different penalties — e.g., $\varphi(x) = x^2$, implying the minimization of $\sqrt{\sum_i p(i)n(i)^2}$ — do not share this independence property, as an optimal code tree with optimal subtrees need not exist. Thus finding an optimal code for such penalties is more difficult. There should, however, be cases in which this is possible for convex φ which grow more slowly than some exponential.

Another extension of this work would be to find coding algorithms for other probability mass functions under the nonlinear penalties already considered, e.g., to attempt to use the techniques of [33, pp. 103–105] for a more reliable algorithm. Other possible extensions and generalizations involve variants of geometric probability distributions; in addition to the one we mentioned that is analogous to Proposition (2) in [27], there are others in [28]–[30]. Extending these methods to nonbinary codes should also be feasible, following the approaches in [27] and [32]. Finally, as a nonalgorithmic result, it might be worthwhile to characterize *all* optimal codes — not merely finding *an* optimal code — as in [26, p. 289].

APPENDIX I
OPTIMAL MAXIMAL REDUNDANCY GOLOMB CODES FOR LARGE θ

Let us calculate optimal maximal redundancy as a function of $\theta \geq 0.5$:

$$
\begin{aligned}
R^*(N_\theta^{**}, P_\theta) &= \max_i n_\theta^{**}(i) + \log_2 p_\theta(i) \\
&= 1 + \left\lceil \log_2 \left\lceil -\frac{1}{\log_2\theta} \right\rceil \right\rceil + \\
&\quad \log_2(1-\theta) + \\
&\quad \left(2^{\left\lceil \log_2 \left\lceil -\frac{1}{\log_2\theta} \right\rceil \right\rceil} - \left\lceil -\frac{1}{\log_2\theta} \right\rceil \right) \log_2\theta \\
&= 1 - \left\lceil -\frac{1}{\log_2\theta} \right\rceil \log_2\theta + \\
&\quad \log_2\left(-\frac{1-\theta}{\log_2\theta}\right) - \\
&\quad 2^{\left\lceil \log_2 \left(-\frac{1}{\log_2\theta}\right) \right\rceil - \log_2\left(-\frac{1}{\log_2\theta}\right)} + \\
&\quad \left\lceil \log_2\left(-\frac{1}{\log_2\theta}\right) \right\rceil - \log_2\left(-\frac{1}{\log_2\theta}\right) \\
&= 2 + \log_2\left(-\frac{1-\theta}{\log_2\theta}\right) - \left\lceil -\frac{1}{\log_2\theta} \right\rceil \log_2\theta - \\
&\quad 2^{1-\langle \log_2\left(-\frac{1}{\log_2\theta}\right)\rangle} - \left\langle \log_2\left(-\frac{1}{\log_2\theta}\right) \right\rangle,
\end{aligned}
$$

where $\langle x \rangle$ denotes the fractional part of x, i.e., $\langle x \rangle \triangleq x - \lfloor x \rfloor$, since

$$
\left\lceil \log_2 \left\lceil -\frac{1}{\log_2\theta} \right\rceil \right\rceil = \left\lceil \log_2\left(-\frac{1}{\log_2\theta}\right) \right\rceil
$$

for $\theta > 0.25$ (and thus for $\theta \geq 0.5$). Using the Taylor series expansion about $\theta = 1$, we find

$$
\log_2\left(-\frac{1-\theta}{\log_2\theta}\right) = -\log_2\log_2 e - (\log_2\sqrt{e})(1-\theta) + O((1-\theta)^2)
$$

where e is the base of the natural logarithm. Additionally,

$$
-\left\lceil -\frac{1}{\log_2\theta} \right\rceil \log_2\theta = 1 + O(1-\theta).
$$

Note that this actually oscillates between 1 and $1+(1-\theta)\log_2 e$ in the limit, so this first-order asymptotic term cannot be improved upon. However, the remaining terms

$$
2 - 2^{1-\langle \log_2\left(-\frac{1}{\log_2\theta}\right)\rangle} - \left\langle \log_2\left(-\frac{1}{\log_2\theta}\right) \right\rangle \tag{16}
$$

oscillate in the zero-order term. Assigning $x = \langle \log_2(-1/\log_2\theta) \rangle$, we find that (16) achieves its minimum value, 0, at 0 and 1. The maximum point is easily found via a first derivative test. This point is achieved at $x = 1 - \log_2\log_2 e$, at which point (16) achieves the maximum value $1 - \log_2 e + \log_2\log_2 e$. Thus, gathering all terms,

$$
\liminf_{\theta\uparrow 1} R^*(N_\theta^{**}, P_\theta) = 1 - \log_2\log_2 e = 0.4712336270\ldots,
$$

$$
\limsup_{\theta\uparrow 1} R^*(N_\theta^{**}, P_\theta) = 2 - \log_2 e = 0.5573049591\ldots,
$$

and, overall,

$$
\begin{aligned}
R^*(N_\theta^{**}, P_\theta) &= 3 - \log_2\log_2 e - \\
&\quad 2^{1-\langle \log_2\left(-\frac{1}{\log_2\theta}\right)\rangle} - \left\langle \log_2\left(-\frac{1}{\log_2\theta}\right) \right\rangle + \\
&\quad O(1-\theta).
\end{aligned}
$$

This oscillating behavior is similar to that of the average redundancy of a complete tree, as in [35] and [36, p. 192].

Contrast this with the periodicity of the minimum *average* redundancy for a Golomb code: [37]

$$
\begin{aligned}
\bar{R}(N_{\theta,1}^*, P_\theta) &= 1 - \log_2\log_2 e - \log_2 e + \\
&\quad 2^{2-2^{1-\langle \log_2\left(-\frac{1}{\log_2\theta}\right)\rangle}} - \left\langle \log_2\left(-\frac{1}{\log_2\theta}\right) \right\rangle + \\
&\quad O(1-\theta)
\end{aligned}
$$

where $N_{\theta,1}^*$ is the optimal code for the traditional (linear) penalty.

APPENDIX II
GLOSSARY OF TERMS

Notation	Meaning
a	Base of exponential penalty
$b(x,k)$	$(x+1)$th codeword of complete binary code with k items (i.e., the order-preserving [alphabetic] code having the first $2^{\lceil \log_2 k\rceil} - k$ items with length $\lfloor \log_2 k\rfloor$ and the last $2k - 2^{\lceil \log_2 k\rceil}$ items with length $\lceil \log_2 k\rceil$)
$c(i)$	Codeword (for symbol) i
C	Code $\{c(i)\}$
e	Base of the natural logarithm ($e \approx 2.71828$)
Gk	Golomb code with parameter k, one of the form $\{1^{\lfloor j/k\rfloor} 0 b(j \bmod k, k) : j \geq 0\}$
$H_\alpha(P)$	Rényi entropy $(1-\alpha)^{-1}\log_2 \sum_{i\in\mathcal{X}} p(i)^\alpha$ (or, if $\alpha \in \{0, 1, \infty\}$, the limit of this)
i^{**}	Index of the codeword that, among a given code's inputs $i \in \mathcal{X}$, maximizes pointwise redundancy, $n(i) + \log_2 p(i)$
$j \bmod k$	$j - k\lfloor j/k\rfloor$
$L_a(P, N)$	Penalty $\log_a \sum_{i\in\mathcal{X}} p(i) a^{n(i)}$
$n(i)$	Length of codeword (for symbol) i
N	$\{n(i)\}$, the lengths for a given code
$n^{(r)}(i)$	Length of codeword i of an optimal code minimizing maximum redundancy for $W^{(r)}$
$N^{(r)}$	$\{n^{(r)}(i)\}$, the lengths of an optimal code minimizing maximum redundancy for $W^{(r)}$
$n^*(i)$	Length of codeword i of an optimal code for an exponential penalty, L
$(n_{\theta,a}^*(i))$	(...if θ and a are specified)
N^*	$\{n^*(i)\}$, the lengths of an optimal code
$(N_{\theta,a}^*)$	(...if θ and a are specified)
$n_{\theta,a,d}^*(i)$	Length of codeword i of an optimal code minimizing dth exponential redundancy
$N_{\theta,a,d}^*$	$\{n_{\theta,a,d}^*(i)\}$, the lengths of an optimal code minimizing dth exponential redundancy
$n^{**}(i)$	Length of codeword i of an optimal code minimizing maximum redundancy
N^{**}	$\{n^{**}(i)\}$, the lengths of an optimal code minimizing maximum redundancy
$O(\cdot)$	Order of $\cdot$ asymptotic complexity
$p(i)$	Probability of input symbol i

Notation	Meaning
	continued
$(p_\theta(i))$	(...for geometric dist^r with parameter θ)
$(p_\lambda(i))$	(...for Poisson dist^r with parameter λ)
P	$\{p(i)\}$, the input probability mass function
(P_θ)	(...for geometric dist^r with parameter θ)
(P_λ)	(...for Poisson dist^r with parameter λ)
$\bar{R}_a(N,P)$	$L_a(P,N) - H_{\alpha(a)}(P)$, the average pointwise redundancy
$R_d(N,P)$	$d^{-1}\log_2 \sum_{i\in\mathcal{X}} p(i)2^{d(n(i)+\log_2 p(i))}$, the dth exponential redundancy
$R^*(N,P)$	$\max_{i\in\mathcal{X}}[n(i)+\log_2 p(i)]$, the maximum pointwise redundancy
$\mathbb{R}$	The set of real numbers
$\mathbb{R}_+$	The set of positive real numbers
s_0	Upper bound on s^*
s^*	$\ln a$ for a corresponding to optimal coding for buffer overflow
$w(i)$	Weight (for symbol) i
W	$\{w(i)\}$, the set of weights
$w^{(r)}(i)$	$p(i)$ for $i \le r$, $2p(r+1)$ for $i = r+1$
$W^{(r)}$	$\{p(0),p(1),\ldots,p(r),2p(r+1)\}$
$\mathcal{X}$	Input alphabet (usually $\mathcal{X}_\infty = \{0,1,\ldots\}$)
$\alpha(a)$	$1/(1+\log_2 a)$ (parameter for Rényi entropy)
θ	Geometric dist^r parameter ($p_\theta(i) = (1-\theta)\theta^i$)
λ	Poisson dist^r parameter ($p_\lambda(i) = \lambda^i e^{-\lambda}/i!$)
Φ	Golden ratio, $\frac{1}{2}(1+\sqrt{5})$

REFERENCES

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

[2] S. Yang and P. Qiu, "Efficient integer coding for arbitrary probability distributions," *IEEE Trans. Inf. Theory*, vol. IT-52, no. 8, pp. 3764–3772, Aug. 2006.

[3] S. W. Golomb, "Run-length encodings," *IEEE Trans. Inf. Theory*, vol. IT-12, no. 3, pp. 399–401, July 1966.

[4] R. G. Gallager and D. C. van Voorhis, "Optimal source codes for geometrically distributed integer alphabets," *IEEE Trans. Inf. Theory*, vol. IT-21, no. 2, pp. 228–230, Mar. 1975.

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

[6] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, July 2003.

[7] M. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," *IEEE Trans. Image Processing*, vol. 9, no. 8, pp. 1309–1324, Aug. 2000, originally as Hewlett-Packard Laboratories Technical Report No. HPL-98-193R1, November 1998, revised October 1999. Available from http://www.hpl.hp.com/loco/.

[8] L. L. Campbell, "Definition of entropy by means of a coding problem," *Z. Wahrscheinlichkeitstheorie und verwandte Gebiete*, vol. 6, pp. 113–118, 1966.

[9] J. Aczél and Z. Daróczy, *On Measures of Information and Their Characterizations*. New York, NY: Academic, 1975.

[10] F. Jelinek, "Buffer overflow in variable length coding of fixed rate sources," *IEEE Trans. Inf. Theory*, vol. IT-14, no. 3, pp. 490–501, May 1968.

[11] 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.

[12] A. C. Blumer and R. J. McEliece, "The Rényi redundancy of generalized Huffman codes," *IEEE Trans. Inf. Theory*, vol. IT-34, no. 5, pp. 1242–1249, Sept. 1988.

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

[14] P. Mendenhall. (2002, Oct. 26) Cell phones were rebels' downfall. MSNBC News.

[15] J. Taranto. (2002, Oct. 28) Best of the Web Today. OpinionJournal, from The Wall Street Journal Editorial Page. Available from http://www.opinionjournal.com/best/?id=110002538.

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

[17] T. Linder, V. Tarokh, and K. Zeger, "Existence of optimal prefix codes for infinite source alphabets," *IEEE Trans. Inf. Theory*, vol. IT-43, no. 6, pp. 2026–2028, Nov. 1997.

[18] 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.

[19] D. S. Parker, Jr., "Conditions for optimality of the Huffman algorithm," *SIAM J. Comput.*, vol. 9, no. 3, pp. 470–489, Aug. 1980.

[20] P. A. Humblet, "Source coding for communication concentrators," Ph.D. dissertation, Massachusetts Institute of Technology, 1978.

[21] M. C. Golumbic, "Combinatorial merging," *IEEE Trans. Comput.*, vol. C-25, no. 11, pp. 1164–1167, Nov. 1976.

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

[23] M. B. Baer, "A general framework for codes involving redundancy minimization," *IEEE Trans. Inf. Theory*, vol. IT-52, no. 1, pp. 344–349, Jan. 2006.

[24] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, July 1948.

[25] A. Rényi, "On measures of entropy and information," in *Proc. 4th Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, 1961, pp. 547–561.

[26] M. J. Golin, "A combinatorial approach to Golomb forests," *Theoretical Computer Science*, vol. 263, no. 1–2, pp. 283–304, July 2001.

[27] J. Abrahams, "Huffman-type codes for infinite source distributions," *Journal of the Franklin Institute*, vol. 331B, no. 3, pp. 265–271, May 1994.

[28] N. Merhav, G. Seroussi, and M. Weinberger, "Optimal prefix codes for sources with two-sided geometric distributions," *IEEE Trans. Inf. Theory*, vol. IT-46, no. 2, pp. 121–135, Mar. 2000.

[29] M. J. Golin and K. K. Ma, "Algorithms for constructing infinite Huffman codes," Hong Kong University of Science & Technology Theoretical Computer Science Center, Tech. Rep. HKUST-TCSC-2004-07, Aug. 2004, available from http://www.cs.ust.hk/tcsc/RR/index_7.html.

[30] F. Bassino, J. Clément, G. Seroussi, and A. Viola, "Optimal prefix codes for two-dimensional geometric distributions," in *Proc., IEEE Data Compression Conf.*, Mar. 28–30, 2006, pp. 113–122.

[31] P. A. Humblet, "Optimal source coding for a class of integer alphabets," *IEEE Trans. Inf. Theory*, vol. IT-24, no. 1, pp. 110–112, Jan. 1978.

[32] A. Kato, T. S. Han, and H. Nagaoka, "Huffman coding with an infinite alphabet," *IEEE Trans. Inf. Theory*, vol. IT-42, no. 3, pp. 977–984, May 1996.

[33] M. B. Baer, "Coding for general penalties," Ph.D. dissertation, Stanford University, 2003.

[34] M. Drmota and W. Szpankowski, "Precise minimax redundancy and regret," *IEEE Trans. Inf. Theory*, vol. IT-50, no. 11, pp. 2686–2707, Nov. 2004.

[35] R. G. Gallager, "Variations on a theme by Huffman," *IEEE Trans. Inf. Theory*, vol. IT-24, no. 6, pp. 668–674, Nov. 1978.

[36] D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, 2nd ed. Reading, MA: Addison-Wesley, 1998.

[37] W. Szpankowski, "Asymptotic redundancy of Huffman (and other) block codes," *IEEE Trans. Inf. Theory*, vol. IT-46, no. 7, pp. 2434–2443, Nov. 2000.