References

[1] R. K. Ahuja, T. L. Maganti, and J. B. Orlin, *Network Flows:Theory, Algorithms and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[2] R. Ahlswede, N. Cai, S.-Y. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[3] S.-Y. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Trans. Inf. Theory*, vol. 49, no. 2, pp. 371–381, Feb. 2003.

[4] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 782–795, Oct. 2003.

[5] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, and L. Tolhuizen, "Polynomial time algorithms for multicast network code construction," *IEEE Trans. Inf. Theory*, submitted for publication.

[6] T. Ho, R. Koetter, M. Médard, M. Effros, J. Shi, and D. Karger, "Toward a random operation of networks," *IEEE Trans. Inf. Theory*, submitted for publication.

[7] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. 41st Allerton Conf. Communication, Control, and Computing*, Monticello, IL, Oct. 2003.

[8] M. Médard, M. Effros, T. Ho, and D. Karger, "On coding for nonmulticast networks," in *Proc. 41st Allerton Conf. Communication, Control, and Computing*, Monticello, IL, Oct. 2003.

[9] R. Dougherty, C. Freiling, and K. Zeger, "Insufficiency of linear coding in network information flow," *IEEE Trans. Inf. Theory*, vol. 51, no. 8, pp. 2745–2759, Aug. 2005.

[10] T. Ho, M. Médard, M. Effros, and R. Koetter, "Network coding for correlated sources," in *Proc. Conf. Information Sciences and Systems (CISS)*, Princeton, NJ, Mar, 2004.

[11] A. Ramamoorthy, K. Jain, P. A. Chou, and M. Effros, "Separating distributed source coding from network coding," in *Proc. 42nd Allerton Conf. Communication, Control, and Computing*, Monticello, IL, Oct. 2004.

[12] B. Bollobas, *Random Graphs*. London/New York: Academic, 1985.

[13] M. Penrose, *Random Geometric Graphs*. Oxford, U.K.: Oxford Univ. Press, 2003.

[14] G. Kramer and S. Savari, "On networks of two-way channels," in *Proc. DIMACS Workshop on Algebraic Coding Theory and Information Theory*, Piscataway, NJ, Dec. 2003.

[15] D. R. Karger, "Random sampling in cut, flow and network design problems," *Math. of Oper. Res.*, vol. 24, no. 2, pp. 0383–0413, 1999.

[16] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 1995.

[17] Y. Wu, P. A. Chou, and K. Jain, "A comparison of network coding and tree packing," in *Proc. IEEE Int. Symp. Information Theory*, Chicago, IL, Jun./Jul. 2004, p. 143.

[18] T. Cover and J. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.

[19] C. Bettstetter, "On the minimum node degree and connectivity of a wireless multihop network," in *Proc. MobiHoc*, Lausanne, Switzerland, 2002, pp. 80–91.

[20] K. Jain, M. Mahdian, and M. R. Salavatipour, "Packing Steiner trees," in *Proc. 14th Annu. ACM-SIAM Symp. Discrete Algorithms*, Baltimore, MD, 2003, pp. 266–274.

[21] R. Durrett, *Probability: Theory and Examples*, 2nd ed: Duxbury, 1995.

An Improvement to the Bit Stuffing Algorithm

Sharon Aviran, Paul H. Siegel, *Fellow, IEEE*, and Jack Keil Wolf, *Life Fellow, IEEE*

***Abstract*—The bit stuffing algorithm is a technique for coding constrained sequences by the insertion of bits into an arbitrary data sequence. This approach was previously introduced and applied to (d, k) constrained codes. Results show that the maximum average rate of the bit stuffing code achieves capacity when $k = d + 1$ or $k = \infty$, while it is suboptimal for all other (d, k) pairs. Furthermore, this technique was generalized to produce codes with an average rate that achieves capacity for all (d, k) pairs. However, this extension results in a more complicated scheme. This correspondence proposes a modification to the bit stuffing algorithm that maintains its simplicity. We show analytically that the proposed algorithm achieves improved average rates over bit stuffing for most (d, k) constraints. We further determine all constraints for which this scheme produces codes with an average rate equal to the Shannon capacity.**

***Index Terms*—Bit-stuffing encoder, (d, k)-constrained systems, Shannon capacity.**

I. Introduction

A binary sequence satisfies a run-length-limited (RLL) (d, k) constraint if any run of consecutive zeros is of length at most k and any two successive ones are separated by a run of consecutive zeros of length at least d. Such sequences are called (d, k)*-sequences* and are commonly used in magnetic and optical recording [1], [2]. The (d, k) constraint is used in order to solve two problems that arise when performing peak detection: d minimizes intersymbol interference and k assists in timing recovery. Relevant (d, k) pairs range over all integers d, k, such that $0 \le d < k \le \infty$.

One can use a labeled directed graph to generate all possible (d, k)-sequences by reading off the labels along paths in the graph. This graph is referred to as a (d, k) *constraint graph*. A graph that produces these sequences for $k < \infty$ is shown in Fig. 1.

Let $N_{d,k}(n)$ be the number of distinct (d, k)-sequences of length n. The *Shannon capacity* of a (d, k) constraint is defined as

$$C(d, k) = \lim_{n\to\infty} \frac{\log_2 N_{d,k}(n)}{n}.$$

The capacity can be computed by applying a more general result derived by Shannon [3]. It was shown (see, e.g., [1]) that $C(d, k) = \log_2 \lambda_{d,k}$, where $\lambda_{d,k}$ is the largest real eigenvalue of the adjacency matrix of the constraint graph. Therefore, $\lambda_{d,k}$ is the largest real root of the characteristic polynomial of the matrix $P_{d,k}(z)$, which takes the form

$$P_{d,k}(z) = \begin{cases} z^{k+1} - \sum_{j=0}^{k-d} z^j, & k \text{ is finite} \\ z^{d+1} - z^d - 1, & k = \infty. \end{cases}$$

It was further shown that for all values of d and k the capacity exists and that $\lambda_{d,k} \in (1, 2)$ for all (d, k) pairs such that $(d, k) \ne (0, \infty)$.

Manuscript received February 4, 2002; revised November 22, 2004. This work was supported in part by the National Science Foundation under Grant CCR-0219582, part of the Information Technology Research Program, and in part by the Center for Magnetic Recording Research at the Univevrsity of California, San Diego. The material in this correspondence was presented in part at the IEEE International Symposium on Information Theory, Chicago, IL, June/July 2004.

The authors are with the Department of Electrical and Computer Engineering and the Center for Magnetic Recording Research, University of California, San Diego, La Jolla, CA 92093 USA (e-mail: saviran@ucsd.edu; psiegel@ucsd.edu; jwolf@ucsd.edu).

Communicated by K. A. S. Abdel-Ghaffar, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2005.851764

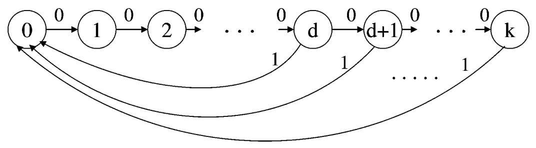

Fig. 1. Constraint graph for the (d,k) constraint with $d>0$ and k finite.

The idea of constrained coding by insertion of extra bits into an uncoded data stream was introduced by Lee [4]. Bender and Wolf [5] proposed a modification to Lee's algorithm which is intended for encoding (d,k) sequences. Their technique is known as the bit stuffing algorithm. The bit stuffing algorithm first converts the input sequence into a sequence having different statistical properties. It then inserts additional bits in a manner that guarantees that the resulting sequences satisfy the (d,k) constraint. Both operations are invertible so that the input sequence can be reproduced.

Bit stuffing has been used in various applications, such as in the X.25 protocol, where it has been used to ensure that the bit pattern of the frame delimiter flag will not appear in the data sequence [6]. The emphasis here is on finding the most efficient bit stuffing algorithm in terms of the asymptotical rate that can be achieved.

Let us define an *optimal algorithm* as an invertible mapping from unconstrained binary data to the constraint having the maximum average information rate. It is well known that this maximum rate equals the capacity of the constraint. Bender and Wolf [5] showed that the bit stuffing algorithm is optimal for all $(d,d+1)$ and (d,∞) constraints and is suboptimal for all other cases. This leaves room for improvement whenever $d+2\le k<\infty$.

In this correspondence, we modify the bit stuffing algorithm by flipping certain bits from the converted input sequence while the logic of insertion of extra bits remains unchanged. We name the proposed modification the bit flipping algorithm. We analyze the performance of both algorithms to obtain the following main results of this correspondence (see Section II-D for the precise statement of Theorem 6).

Theorem 6 : Let $d\ge 1$ and $d+2\le k<\infty$. Then the bit flipping algorithm achieves a greater maximum average rate than the bit stuffing algorithm.

Theorem 7 : Let $d\ge 0$ and $d+2\le k<\infty$. Then the bit flipping algorithm is optimal if and only if $d=2$ and $k=4$.

As will be discussed in Section II-B, there is another modification of the bit stuffing technique [6] that is optimal for all values of d and k. It is therefore superior in performance to the bit flipping algorithm. Nevertheless, our proposed algorithm maintains the simplicity of the bit stuffing algorithm while the mentioned scheme is more complex.

In Section II, we give an example that motivated the idea of flipping. In this section, we study in detail both the bit stuffing and the bit flipping algorithms. We devote the rest of Section II to establishing a sequence of lemmas needed to prove the performance improvement (Theorem 6). In Section III, we characterize all (d,k) constraints for which the bit flipping algorithm is optimal (Theorem 7).

II. Improving the Performance of Bit Stuffing by Bit Flipping

In this section, we introduce the bit stuffing algorithm and propose a modification to it—the bit flipping algorithm. We derive explicit expressions for the asymptotic average rates of both algorithms. We use these expressions to show that the proposed algorithm yields a higher average rate than the bit stuffing algorithm for all $d\ge 1$ and $d+2\le k<\infty$.

A. The Bit Stuffing Algorithm

We begin by describing the bit stuffing encoder, which encodes an arbitrary data sequence into a (d,k)-constrained sequence. The encoder consists of the following two components:

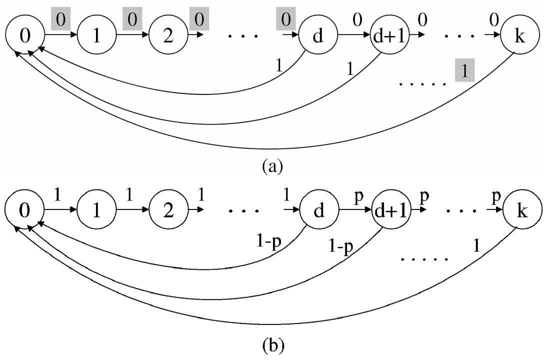

Fig. 2. Graph description of a bit stuffing encoder for the (d,k) constraint with $d>0$ and k finite.

- a distribution transformer,
- a constrained encoder.

Assume that the input is a sequence of independent and identically distributed (i.i.d.) unbiased (i.e., Bernoulli with probability $\frac{1}{2}$) random bits. The distribution transformer converts the unbiased sequence into a sequence of independent bits, whose probability of a 0 is some $p\in[0,1]$ (i.e., Bernoulli with probability p). We say that the output sequence is p*-biased* and refer to it as the *biased sequence*. This conversion can be implemented in a one-to-one manner. Hence, we can apply the reverse transformation to recover the unbiased data. The asymptotic expected rate of such a scheme is $h(p)$, where $h(p)=-p\log_2(p)-(1-p)\log_2(1-p)$ is the binary entropy function. A possible method of conversion would be to use the Elias code [7, pp. 61–62]. However, this approach was designed for infinite input sequences. One modification to this idea that applies to finite sequences and can be implemented using a finite precision arithmetic appears in [8].

The constrained encoder inserts extra bits into the biased sequence in order to avoid possible violations of the (d,k) constraint. It writes the biased sequence while keeping track of the number of consecutive zeros in the sequence, called the *run length*. Once the run length equals k, the encoder inserts a 1 followed by d 0's. This guarantees that both the d and k restrictions are satisfied. Whenever encountering a biased 1, the encoder inserts d 0's so as to satisfy the d limitation. The inserted bits are also called *stuffed bits*. The graph in Fig. 2(a) describes the encoder, where the edge labels are the output symbols. The stuffed bits are highlighted. Note that for $d>0$ and finite k the constrained encoder is in fact a realization of the graph in Fig. 2(b), which is similar to the constraint graph in Fig. 1 except for the edge labels. Here the edge labels represent the probabilities that the next bit will assume the value 0 or 1. If the bit assumes a value of 0, then we move to the next state to the right. If it is a 1, then we return to state 0. We will find this graph representation of the encoder useful in Sections II-B and -C.

The decoder is comprised of the corresponding two components, arranged in a reverse order. The constrained decoder reads the encoded constrained sequence and keeps track of the run length. Whenever reaching a run length of k, it deletes the $d+1$ stuffed bits that follow it. If it encounters a 1, then it removes the next d stuffed 0's. This results in the encoded p-biased sequence, which is then fed into the inverse distribution transformer, so as to obtain the original unbiased data.

The proposed scheme produces a variable-rate code. Its expected rate is the product of the expected rates of the two components, the first being $h(p)$ when the code length goes to infinity. Note that we could directly apply just the constrained encoder component to the unbiased input data to obtain a (d,k)-constrained sequence. However, adding the distribution transformer in fact results in an improved overall average rate. This sheds some light on the role of the transformer, namely, to better fit the data to the constraint. To further explain, observe that each 1 in the biased sequence results in d stuffed 0's. On the other hand, $k-d$ consecutive biased 0's will result in the stuffing of a single

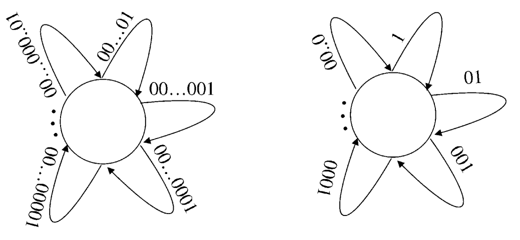

Fig. 3. Graph description of a (d,k) constraint where the edge labels are the allowable runs.

1 followed by d 0's. Thus, as $k-d$ increases we would expect that fewer 1's in the input sequence will result in fewer stuffed bits. Such sequences will yield a higher rate in the bit stuffing encoding process. The distribution transformed sequences have this desired property on average. On the other hand, as we increase the probability of a 0, the rate of the first component $h(p)$ decreases. Thus, we need to optimize p in order to maximize the average overall rate. Optimization is done numerically due to the complexity of the rate expression. We shall show in Section II-D that, as expected, having more 0's than 1's indeed yields a better overall rate for most cases where $d>0$, though this is not always true.

Bender and Wolf [5], [9] analyzed the performance of the bit stuffing algorithm by deriving an expression for its average asymptotic rate. We limit our discussion to a finite k and follow the methods of their derivation. We will later show that an infinite k is not of interest to our work due to the optimality of bit stuffing for this case. We start by modeling the constrained (d,k) sequences by a one-state constraint graph, depicted in Fig. 3(a). The edges in our graph represent the allowable runs of consecutive 0's followed by a 1. In order to get a description of the corresponding biased input data sequences we remove the stuffed 0's and 1's and obtain the graph in Fig. 3(b).

We can now calculate the average rate of the constrained encoder component. Having assumed that the biased sequence is i.i.d. $\mathrm{Ber}(p)$, the average input length is

$$L_{\text{in}} = \sum_{j=0}^{k-d-1} (j+1)p^j(1-p) + (k-d)p^{k-d} = \frac{1-p^{k-d}}{1-p}$$

for all p such that $0 \le p < 1$, and the average output length is

$$L_{\text{out}} = L_{\text{in}} + d + p^{k-d}.$$

Therefore, the asymptotic average information rate of the algorithm is

$$I(p,d,k) = \frac{L_{\text{in}}}{L_{\text{out}}} \times h(p) = \frac{(1-p^{k-d})h(p)}{1-p^{k-d+1}+d(1-p)}$$

for all p such that $0 \le p < 1$ and $I(p,d,k)=0$ for $p=1$.

Bender and Wolf [5], [9] used this expression to obtain a characterization of all cases where the bit stuffing algorithm is optimal, i.e., its maximum average rate equals the capacity of the (d,k) constraint. Their results are summarized in the following proposition.

Proposition 1: The bit stuffing algorithm for (d,k) constraints is optimal for the following cases:

- $k = d+1$ for all $d \ge 0$,
- $k = \infty$ for all $d \ge 0$.

It is not optimal for all other values of d and k.

For the remaining suboptimal cases, numerical optimization of the average rate shows that bit stuffing codes achieve rates that are very close to capacity. Thus, the algorithm is said to be *nearly optimal* for these cases. For detailed results see [9].

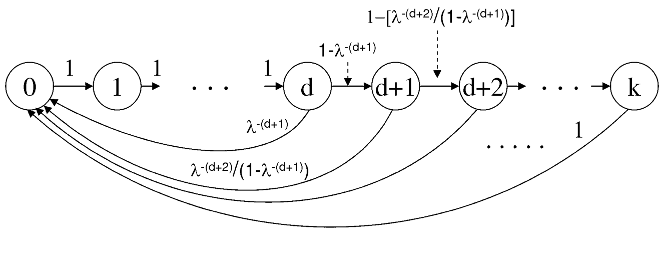

Fig. 4. Edge probabilities for maxentropic (d,k) sequences.

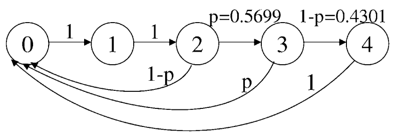

Fig. 5. Edge probabilities for the $(2,4)$ maxentropic measure.

B. Motivating Example—Maxentropic Measure for the $(2,4)$ Case

In this subsection we demonstrate an example that triggered the development of the bit flipping algorithm. Before we go through the example we need to introduce the notion of the *maxentropic measure*.

Consider a constraint graph where we assign nonzero probabilities to the labeled edges leaving each state, thus producing an information source. A result by Shannon [3] states that for any such graph one can always assign certain probabilities to the edges at each state, such that the resulting information source has maximum entropy. This set of probabilities can be computed by formulas that Shannon prescribed and is called the *maxentropic measure*. Shannon further showed that this maximum entropy is equal to the capacity of the constrained system. Applying Shannon's result to (d,k) constraints (see [1] for a complete derivation) yields the probabilities shown in Fig. 4, where $\lambda_{d,k} = 2^{C(d,k)}$.

Recently, Wolf [6] proposed a modification to bit stuffing based on Shannon's result. He showed that the modified scheme achieves rates that are equal to capacity for all values of d and k. The idea is to let the constrained encoder realize the maxentropic measure by feeding it with several distinct biased streams. For example, when k is finite we have states $d, d+1, \ldots, k-1$ with two edges exiting from each state, while the other states have only one exiting edge. Each pair of emanating edges corresponds to a random bit with a certain bias. The single edges correspond to stuffed bits. Denote the maxentropic probability when moving from state i to state $i+1$ by p_i, then $p_i = 1$ for $i = 0, 1, \ldots, d-1, k$. We first "break" the unbiased data into $k-d$ distinct streams, denoted $S_d, S_{d+1}, \ldots, S_{k-1}$, and input the streams into $k-d$ different distribution transformers with biases $p_d, p_{d+1}, \ldots, p_{k-1}$. Note that each stream is fed into exactly one of the transformers. This results in $k-d$ biased streams, $S_d^*, S_{d+1}^*, \ldots, S_{k-1}^*$, with biases $p_d, p_{d+1}, \ldots, p_{k-1}$, respectively. Having multiple biased streams, the constrained encoder takes a bit from stream S_i^* when in state i. Clearly, the encoded sequences have maximum entropy.

Let us look at the maxentropic measure for the $(2,4)$ case, shown in Fig. 5. Denote the probability of moving from state 2 to state 3 by p^*. It turns out (as will be confirmed in Section III) that the probability of moving from state 3 to state 4 equals $1-p^*$, where $p^* \approx 0.5699$. This special property suggests that in this case we do not need two distribution transformers in order to achieve capacity. We can use a single transformer with $\Pr(0) = p^*$ and modify the bit stuffing algorithm to obey the following rule.

- If the current run length equals 2 then write the next biased bit.

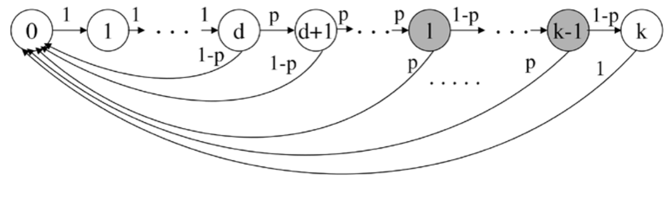

Fig. 6. Graph description of a possible bit flipping encoder for the (d, k) constraint.

- If the current run length equals 3 then write the complement of the next biased bit, i.e., *flip* the next biased bit.

In other words, flip the biased bit only when in state 3.

Recall that bit stuffing is not optimal in the $(2, 4)$ case. Yet, the addition of bit flipping resulted in an optimal algorithm in this case. Thus, at least in this case, a conditional bit flipping improves the performance of bit stuffing with only a single transformer. This observation motivated us to examine whether we could do better by flipping in the general case. We generalize the flipping idea and analyze the resulting algorithm in the subsection to follow.

C. The Bit Flipping Algorithm

Consider the case where k is finite and $k \geq d + 2$ and let l be an integer such that $d + 1 \leq l \leq k - 1$. Suppose we run the bit stuffing algorithm using a single distribution transformer. We modify the logic of the constrained encoder in the following manner.

- If the current run length is smaller than l then write the next biased bit
- If the current run length is greater than or equal to l then flip the next biased bit before writing.

In other words, flip the biased bit starting from state l.The bit flipping algorithm is illustrated by the constraint graph in Fig. 6.

Four interrelated questions arise. What is the optimal flipping position? For which constraints can we improve bit stuffing rate by flipping? Can we achieve capacity for more constraints using flipping? If not, how far from capacity are we? In the rest of this correspondence we settle these four questions. This subsection and Section II-D deal with the first two questions. Section III addresses the latter two.

We first derive an expression for the average rate. When flipping a bit starting from state l, the average biased input length is

$$\begin{aligned} L_{\text{in}} &= \sum_{j=0}^{l-d-1} (j+1)p^j(1-p) \\ &\quad + \sum_{j=0}^{k-l-1} (l-d+j+1)p^{l-d}(1-p)^j p \\ &\quad + (k-d)p^{l-d}(1-p)^{k-l} \\ &= \frac{1-p^{l-d}}{1-p} + p^{l-d-1}(1-(1-p)^{k-l}) \end{aligned}$$

and the average output length is

$$L_{\text{out}} = L_{\text{in}} + d + p^{l-d}(1-p)^{k-l}.$$

The asymptotic overall average rate $R(p, l, d, k)$ is given by

$$\begin{aligned} R(p,l,d,k) &= \frac{L_{\text{in}}}{L_{\text{out}}} \times h(p) \\ &= \frac{h(p)}{1 + \frac{d+p^{l-d}(1-p)^{k-l}}{L_{\text{in}}}} \\ &= \frac{\left[p^{l-d-1}(1-2p-(1-p)^{k-l+1}) + 1\right] h(p)}{p^{l-d-1}\left(1-2p-(1-p)^{k-l+2}\right) + 1 + d(1-p)} \end{aligned}$$

for all p such that $0 \leq p < 1$ and for all l such that $d + 1 \leq l \leq k$, and by $R(p, l, d, k) = 0$ for $p = 1$. Note that the rate of the bit stuffing algorithm, where no flipping occurs, is a special case of this expression with $l = k$, i.e., $R(p, k, d, k) = I(p, d, k)$.

Our main goal is to show that under certain conditions the proposed algorithm can achieve a better average rate than bit stuffing. However, the next lemma suggests a more extensive result. It states that when using a transformer with a bias greater than 0.5, state $k - 1$, i.e., one state before the last, is always the optimal state for flipping. A special case of this result is that $R(p, k, d, k) < R(p, k - 1, d, k)$ when $0.5 < p < 1$. In other words, when l is set to $k - 1$, the bit flipping algorithm performs better than bit stuffing for the constraints given in Lemma 2. As one can observe in the proof, it is actually straightforward to prove this special case. Nonetheless, we are looking for the best possible performance. Moreover, finding that the optimal flipping position is always $k - 1$ provides a simple and general formulation of the algorithm, which is independent of d and k for the considered cases.

Lemma 2: Let $d \geq 1$, $d + 2 \leq k < \infty$, and $0.5 < p < 1$. Then

$$R(p, l, d, k) < R(p, k - 1, d, k)$$

for all l such that $d + 1 \leq l \leq k - 2$ or $l = k$.

Proof: Define

$$\begin{aligned} A_l &= \frac{d + p^{l-d}(1-p)^{k-l}}{L_{\text{in}}} \\ &= \frac{d + p^{l-d}(1-p)^{k-l}}{1 - p^{l-d} + (1-p)p^{l-d-1}(1-(1-p)^{k-l})}(1-p). \end{aligned}$$

Then we can write

$$R(p, l, d, k) = \frac{h(p)}{1 + A_l}.$$

Clearly, $A_l \geq 0$ for all $d + 1 \leq l \leq k$. Therefore, $A_i > A_j$ if and only if $R(p, i, d, k) < R(p, j, d, k)$. We now show that A_{k-1} is strictly smaller than any other A_l. First observe that $A_{k-1} < A_k$ for any $0.5 < p < 1$, as can be seen directly from the simplified forms

$$A_k = \frac{d + p^{k-d}}{1 - p^{k-d}}(1-p)$$

and

$$A_{k-1} = \frac{d + p^{k-d-1}(1-p)}{1 - p^{k-d}}(1-p).$$

In order to prove that $A_{k-1} < A_l$ for any $d+1 \leq l < k-1$ and $0.5 < p < 1$ it suffices to show that $A_l < A_{l-1}$ for any $d + 2 \leq l \leq k - 1$.

It is easy to verify that the denominators of

$$A_l = \frac{[d + p^{l-d}(1-p)^{k-l}](1-p)}{1 - p^{l-d} + (1-p)p^{l-d-1}(1-(1-p)^{k-l})}$$

and

$$A_{l-1} = \frac{[d + p^{l-d-1}(1-p)^{k-l+1}](1-p)}{1 - p^{l-d-1} + (1-p)p^{l-d-2}(1-(1-p)^{k-l+1})}$$

are both positive for any $0 \leq p < 1$ and any $d+1 \leq l \leq k$. Therefore, multiplying the inequality $A_l < A_{l-1}$ by the product of the two denominators and dividing by $(1-p)$ we obtain the following equivalent inequality:

$$\begin{aligned} &(d + p^{l-d}(1-p)^{k-l}) \\ &\quad \cdot (1 - p^{l-d-1} + (1-p)p^{l-d-2}(1-(1-p)^{k-l+1})) \\ &< (d + p^{l-d-1}(1-p)^{k-l+1}) \\ &\quad \cdot (1 - p^{l-d} + (1-p)p^{l-d-1}(1-(1-p)^{k-l})). \end{aligned} \tag{1}$$

For any $0.5 < p < 1$, inequality (1) reduces to

$$(1-p)^{k-l-1}[(1-p)d + p(1-p^{l-d})] < d. \tag{2}$$

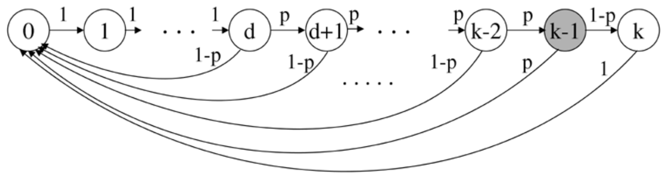

Fig. 7. Graph description of an optimal bit flipping encoder for the (d,k) constraint.

Now, observe that for any $0.5 < p < 1$ the expression in the square brackets is a convex combination of d and $1-p^{l-d}$. Also, note that $1-p^{l-d} < 1 \leq d$. Hence,

$$(1-p)d + p(1-p^{l-d}) < d.$$

Since $l \leq k-1$ and $0.5 < p < 1$, we have

$$0 \leq (1-p)^{k-l-1} \leq 1$$

implying that inequality (2) holds for any $d \geq 1$, $l \leq k-1$, and $0.5 < p < 1$, as desired. □

Looking at the graph in Fig. 7, we can interpret this result as follows. At each of the states $d, d+1, \ldots, k-2$, we would rather have a 0 and move to the right than have a 1 and move back to state 0, where we would have to stuff $d > 0$ bits. However, this is no longer the case when at state $k-1$. Having a 0 will result in $d+1$ stuffed bits as opposed to d stuffed bits due to a 1. In this case, we prefer to go back to state 0 rather than move to the right, a preference reflected in the bit flipping.

Also, note that this result holds only for $p > 0.5$ and in fact is not true for $p < 0.5$. However, the former suffices for our purposes, as we will show later that the optimal bias for bit stuffing is greater than 0.5 for all cases considered but one.

D. Performance Improvement

Recall that the bit stuffing algorithm achieves capacity for the (d,∞) and $(d, d+1)$ constraints for any d and is suboptimal for all other cases. Therefore, there is room for performance improvement for all (d,k) constraints such that $d+2 \leq k < \infty$ and $d \geq 0$. In this subsection, we prove that the bit flipping algorithm achieves a higher rate than bit stuffing for the majority of these constraints.

As mentioned earlier, the result of Lemma 2 is limited to transformers with a bias greater than 0.5. Since the optimal bias for bit stuffing may be smaller than 0.5, Lemma 2 does not guarantee that flipping at state $k-1$ is superior to bit stuffing. Because of the complexity of the bit stuffing rate derivative, we cannot find an explicit form for the optimal bias. Instead, in the next sequence of lemmas we examine the rate derivative to show that the optimal bias for bit stuffing is indeed greater than 0.5 for the following (d,k) pairs:

- $d+3 \leq k < \infty$ and $d \geq 1$,
- $k = d+2$ and $d \geq 2$.

This result together with Lemma 2 guarantees performance improvement in these cases.

Lemma 3: Let $0 < p \leq 0.5$, $d \geq 1$, and $d+3 \leq k < \infty$, then

$$\frac{dI(p,d,k)}{dp} > 0.$$

Proof: Consider the bit stuffing rate derivative

$$\frac{dI(p,d,k)}{dp} = \frac{\frac{d\left((1-p^{k-d})h(p)\right)}{dp} f(p) - \frac{d\left(f(p)\right)}{dp}[(1-p^{k-d})h(p)]}{(f(p))^2}$$

where $f(p) = 1 - p^{k-d+1} + d(1-p)$. We denote the derivative's numerator by $I'_{\text{num}}(p)$ and shall show that $I'_{\text{num}}(p) > 0$ for all $0 < p \leq 0.5$, $d \geq 1$, and $k-d \geq 3$. By rearranging terms we can rewrite

$$I'_{\text{num}}(p) = h(p) \cdot \Big[-(k-d)p^{k-d-1}[1-p^{k-d+1}+d(1-p)] + [(k-d+1)p^{k-d}+d](1-p^{k-d})\Big] + \frac{dh(p)}{dp}\cdot\Big[1-p^{k-d+1}+d(1-p)\Big](1-p^{k-d}).$$

Defining $A = 1-p^{k-d+1}+d(1-p)$, $B = (k-d+1)p^{k-d}+d$, and $C = 1-p^{k-d}$, we see that $A, B, C > 0$ for all $0 < p \leq 0.5$, $d \geq 1$, $k-d \geq 3$, and

$$I'_{\text{num}}(p) = h(p)\Big[-(k-d)p^{k-d-1}A + BC\Big] + \frac{dh(p)}{dp}AC.$$

Now, $\frac{dh(p)}{dp} \geq 0, \ \forall\, 0 < p \leq 0.5$, implying that $\frac{dh(p)}{dp}AC \geq 0$. Hence, since $h(p) > 0$ we need only show that

$$[-(k-d)p^{k-d-1}A + BC] > 0 \tag{3}$$

for the given values of p, k, and d. Writing the latter expression explicitly and rearranging terms yields

$$\begin{aligned}&[-(k-d)p^{k-d-1}A + BC]\\ &\quad = p^{k-d}\Big[(k-d-1)(d+1) + 2 - p^{k-d}\Big]\\ &\qquad + d - p^{k-d-1}(k-d)(d+1)\\ &\quad > 0.\end{aligned}$$

We distinguish between two cases: $k-d \geq 4$ and $k-d = 3$.

In the first case, we use the fact that

$$(k-d-1)(d+1) + 2 - p^{k-d} \geq 3 \times 2 + 2 - \frac{1}{16} > 0$$

hence,

$$p^{k-d}[(k-d-1)(d+1)+2-p^{k-d}] > 0, \qquad \forall 0 < p \leq 0.5,\ d \geq 1,\ k-d \geq 4.$$

It follows that showing that $d - p^{k-d-1}(k-d)(d+1) \geq 0$ will guarantee that inequality (3) holds. Define $l = k-d$; then, we want to prove that $lp^{l-1}(d+1) \leq d$ or

$$lp^{l-1} \leq \frac{d}{d+1}, \qquad \forall\, l \geq 4. \tag{4}$$

We first consider the derivative of the left-hand side of inequality (4)

$$\frac{d(lp^{l-1})}{dl} = p^{l-1} + lp^{l-1}\ln(p) = p^{l-1}\left(1 - l\ln\left(\frac{1}{p}\right)\right).$$

For any $0 < p \leq 0.5$ we have $0.693 \approx \ln(2) \leq \ln(\frac{1}{p}) < \infty$, which implies that $l\ln(\frac{1}{p}) > 1$ and $\frac{d(lp^{l-1})}{dl} < 0$. Consequently, for any $0 < p \leq 0.5$ and any $l \geq 4$ we have

$$lp^{l-1} \leq 4p^3 \leq 4p^3\Big|_{p=\frac{1}{2}} = \frac{1}{2}.$$

The right-hand side of inequality (4) is lower-bounded by $\frac{1}{2}$ for all $d \geq 1$, yielding

$$lp^{l-1} \leq \frac{1}{2} \leq \frac{d}{d+1}, \qquad \forall\, d \geq 1,\ l \geq 4,\ 0 < p \leq 0.5.$$

In the second case, we assign $k-d = 3$ in the left-hand side of inequality (3) and get

$$[-(k-d)p^{k-d-1}A + BC] = d[1+2p^3-3p^2] + 4p^3 - p^6 - 3p^2.$$

Since $1 + 2p^3 - 3p^2$ is positive for all $0 < p \leq 0.5$, then inequality (3) holds if and only if $\frac{3p^2+p^6-4p^3}{1+2p^3-3p^2} < d$. Instead, we show that

$$3p^2 + p^6 - 4p^3 < 1 + 2p^3 - 3p^2 \tag{5}$$

resulting in

$$\frac{3p^2 + p^6 - 4p^3}{1 + 2p^3 - 3p^2} < 1 \leq d.$$

Differentiating both sides of inequality (5) we observe that for $0 < p \leq 0.5$, the left-hand side is an increasing function of p and the right-hand side is a decreasing function of p. Therefore,

$$\begin{aligned} 3p^2 + p^6 - 4p^3 &\leq [3p^2 + p^6 - 4p^3]\big|_{p=\frac{1}{2}} \\ &= \frac{17}{64} \\ &< \frac{1}{2} \\ &= [1 + 2p^3 - 3p^2]\big|_{p=\frac{1}{2}} \\ &\leq 1 + 2p^3 - 3p^2 \end{aligned}$$

confirming inequality (3). □

Lemma 4: Let $0 < p \leq 0.5$, $d \geq 2$, and $k - d = 2$, then

$$\frac{dI(p,d,k)}{dp} > 0.$$

Proof: We proceed along the lines of the preceding proof and want to show that

$$[-(k-d)p^{k-d-1}A + BC] > 0$$

for the given values of p, k, and d. For $k - d = 2$ we need to show that $d(1-p)^2 + 3p^2 - 2p - p^4 > 0$. Now, for $0 < p \leq 0.5$ and $d \geq 2$ we have the following inequalities: $d(1-p)^2 \geq \frac{d}{4} \geq \frac{1}{2}$, $-\frac{1}{3} \leq 3p^2 - 2p < 0$, and $-\frac{1}{16} \leq -p^4 < 0$. Combining the three inequalities we conclude that

$$d(1-p)^2 + 3p^2 - 2p - p^4 \geq \frac{1}{2} - \frac{1}{3} - \frac{1}{16} > 0. \quad □$$

We are now ready to conclude that the optimal bit stuffing bias is strictly greater than 0.5 for the above mentioned (d, k) pairs.

Lemma 5: Let (d, k) satisfy one of the following conditions:

1) $d + 3 \leq k < \infty$ and $d \geq 1$,
2) $k = d + 2$ and $d \geq 2$.

Then

$$\max_{0 \leq p \leq 1} I(p, d, k) = I(p^*, d, k)$$

for some $p^* \in (0.5, 1)$.

Proof: It is easy to verify that the bit stuffing rate function $I(p, d, k)$ is continuous in p on the compact set $[0, 1]$. Thus, it attains a maximum somewhere in that set. The maximum must be attained for some $p^* \in (0, 1)$ since $I(p, d, k) = 0$ for $p \in \{0, 1\}$ and is strictly positive for any $p \in (0, 1)$. The rate derivative exists in the set $(0, 1)$, hence, a necessary condition for a maximum at $p^* \in (0, 1)$ is that $\frac{dI(p,d,k)}{dp}(p^*) = 0$. Lemmas 3 and 4 show that $\frac{dI(p,d,k)}{dp} > 0$ for any $0 < p \leq 0.5$, thus implying that the maximum is attained for some $p^* > 0.5$. □

This result brings us back to our discussion in Section II-A. As said, each 1 we encounter results in d stuffed 0's, while only $k - d$ consecutive 0's result in $d + 1$ stuffed bits, or $\frac{d+1}{k-d}$ stuffed bits per biased 0. It seems that the asymmetry between $\frac{d+1}{k-d}$ and d determines the best bias. We would expect that whenever $\frac{d+1}{k-d} < d$ then inputting fewer 1's will result in fewer stuffed bits and in a better overall rate. Indeed, $\frac{d+1}{k-d} \leq d$ if and only if $d > 0$ and $k - d > 1$, with equality only when $d = 1$ and $k - d = 2$. For all other cases $\frac{d+1}{k-d} > d$. Thus, a transformer that biases the data toward more 0's $(p > 0.5)$ would perform better and the optimum is achieved for $p > 0.5$. The case $(1, 3)$ is not covered by these arguments but can be analyzed explicitly. Also, note that the optimal bit flipping bias may differ from the optimal bit stuffing bias. Nonetheless, once bit flipping performs better than bit stuffing's best performance, then optimizing the bit flipping rate may even further improve its performance.

We are finally in a position to state the main result of this section. We show that for all $d \geq 1$ and $d + 2 \leq k < \infty$, flipping the biased bit at state $k - 1$ strictly improves upon bit stuffing.

Theorem 6: Let $p^*, p^{**} \in [0, 1]$ be the optimal biases for the bit stuffing and the bit flipping algorithms, respectively. Then for all $d \geq 1$ and $d + 2 \leq k < \infty$ the following holds:

$$I(p^*, d, k) < R(p^{**}, k-1, d, k).$$

Proof: Combining Lemmas 2 and 5 we obtain that

$$I(p^*, d, k) = R(p^*, k, d, k) < R(p^*, k-1, d, k) \leq R(p^{**}, k-1, d, k)$$

for all $d+3 \leq k < \infty$ and $d \geq 1$ and for $k = d+2$ and $d \geq 2$. It is left to examine the case of the $(1, 3)$ constraint. In this case, we numerically optimize both algorithms' rate functions and obtain the following optimal biases and optimal rates: $p^* = 0.4906$ and $I(p^*, d, k) = 0.5456$ versus $p^{**} = 0.5557$ and $R(p^{**}, k-1, d, k) = 0.5501$. □

The result of Theorem 6 is reasonable since the asymmetry between $\frac{d+1}{k-d}$ and d still dictates a biasing of the input data toward more 0's. However, the option of flipping allows for more flexibility when fitting the data to the constraint. It enables us to change our preference at a certain state. Indeed, when reaching a run length of $k - 1$ we can save a single stuffed bit by having a 1 versus a 0. Consequently, this changes our preferences in favor of 1's at this state and the opportunity to do so results in an improved rate.

We would like to point out that the case where $d = 0$ was not dealt with in Theorem 6. In this case, it is easy to show that bit stuffing is optimal for $p^* \in (0, 0.5)$ and that there is no bias for which the bit flipping algorithm can improve on the bit stuffing optimum. This observation agrees with the intuitive reasoning that was given for Lemma 5 and Theorem 6.

III. Optimality of the Bit Flipping Algorithm

In this section, we characterize the constraints for which the bit flipping algorithm is optimal. We shall prove that, as numerical evidence suggests, bit flipping is optimal for the $(2, 4)$ constraint. Moreover, we shall find that this is the only optimal case. We conclude with performance results for the $(2, 4)$ case and for some selected suboptimal constraints.

Recall that an algorithm is optimal if its maximum average rate equals the capacity of the constraint. In this case, the resulting (d, k) sequences have maximum entropy that equals the capacity $C(d, k)$, and are referred to as *maxentropic sequences*. A well-known property of maxentropic (d, k) sequences is that the probability of a run of i 0's followed by a 1 is equal to $\lambda_{d,k}^{-(i+1)}$. This property follows from results of Shannon [3] and of Zehavi and Wolf [10]. We use it in the proof of the next theorem, which outlines a complete characterization of optimal cases.

Theorem 7 : Let $d \geq 0$ and $d + 2 \leq k < \infty$. Then the bit flipping algorithm is optimal if and only if $d = 2$ and $k = 4$.

Proof: Let us parse the encoded (d, k) sequence into a concatenation of (possibly empty) runs of 0's followed by a single 1. Let X_i be a random variable denoting the length of the ith phrase in the parsed

TABLE I
SIMULATION RESULTS FOR OPTIMAL PERFORMANCE OF BIT STUFFING VERSUS BIT FLIPPING FOR SOME (d, k) CONSTRAINTS

Constraint	**Bit Stuff Avg. Rate**	**Bit Stuff Optimal Bias**	**Bit Flip Avg. Rate**	**Bit Flip Optimal Bias**	**Capacity**	**$\frac{\text{Bit Stuff Avg. Rate}}{\text{Capacity}}$**	**$\frac{\text{Bit Flip Avg. Rate}}{\text{Capacity}}$**
(1,3)	0.5456	0.4906	0.5501	0.5557	0.5515	98.94%	99.76%
(1,4)	0.6103	0.5275	0.6157	0.5628	0.6175	98.83%	99.71%
(1,7)	0.6754	0.5831	0.6779	0.5928	0.6792	99.44%	99.81%
(2,4)	0.4006	0.5206	0.4057	0.5699	0.4057	98.74%	**100.00%**
(2,5)	0.4579	0.5634	0.4638	0.5930	0.4650	98.47%	99.74%
(3,6)	0.3680	0.5845	0.3730	0.6097	0.3746	98.24%	99.57%
(4,8)	0.3364	0.6320	0.3403	0.6480	0.3432	98.02%	99.16%
(5,9)	0.2914	0.6434	0.2946	0.6577	0.2978	97.85%	98.93%

sequence. As mentioned earlier, the bit flipping algorithm is optimal if and only if it generates maxentropic (d, k) sequences. These sequences must satisfy the following properties [1], [3], [10].

1) The X_i's are i.i.d.
2) $\Pr(X = i) = \lambda_{d,k}^{-i}$, where $\lambda_{d,k} = 2^{C(d,k)}$.

We start with the case where $d + 2 < k < \infty$ and then proceed to deal with $k = d + 2$.

We now use the bit flipping graph description in Fig. 7 for $d + 2 < k < \infty$ in order to translate these optimality properties to the following set of $k - d + 1$ equations:

$$\begin{cases} \Pr(X = i) = p^{i-d-1} \times (1 - p) = \lambda_{d,k}^{-i} \\ \Pr(X = k) = p^{k-d-1} \times p = \lambda_{d,k}^{-k} \\ \Pr(X = k + 1) = p^{k-d-1} \times (1 - p) = \lambda_{d,k}^{-(k+1)} \end{cases}$$

where $d + 1 \le i \le k - 1$. The first $k - d - 1$ equations yield

$$\frac{\Pr(X = i + 1)}{\Pr(X = i)} = p = \frac{1}{\lambda_{d,k}}, \qquad \forall\, d + 1 \le i \le k - 2. \tag{6}$$

Dividing the equation for $i = k$ by the equation for $i = k - 1$ yields

$$\frac{\Pr(X = k)}{\Pr(X = k - 1)} = \frac{p^{k-d}}{p^{k-d-2}(1 - p)} = \frac{p^2}{1 - p} = \frac{1}{\lambda_{d,k}}. \tag{7}$$

Combining (6) and (7) we have

$$\begin{aligned} &\frac{p^2}{1-p} = \frac{1}{\lambda_{d,k}} = p \\ \Leftrightarrow\quad & p(2p - 1) = 0 \\ \Leftrightarrow\quad & p = 0 \text{ or } p = \frac{1}{2}. \end{aligned}$$

Since $p = 0$ results in zero entropy and zero rate then we are left with $p = \frac{1}{2}$. However, $p = \frac{1}{2}$ implies $\lambda_{d,k} = 2$, which contradicts k being finite. Consequently, the bit flipping algorithm is never optimal when $d + 2 < k < \infty$.

We now refer to the graph in Fig. 7 as it appears for the special case of $k = d + 2$. An argument similar to that at the beginning of the proof shows that the bit flipping algorithm is optimal if and only if the following three equations hold:

$$\begin{cases} \Pr(X = d + 1) = 1 - p = \lambda_{d,k}^{-(d+1)} \\ \Pr(X = d + 2) = p \times p = \lambda_{d,k}^{-(d+2)} \\ \Pr(X = d + 3) = p \times (1 - p) = \lambda_{d,k}^{-(d+3)}. \end{cases}$$

The first two equations reduce to

$$\begin{aligned} &\begin{cases} p = 1 - \lambda_{d,k}^{-(d+1)} \\ p = \lambda_{d,k}^{-\frac{(d+2)}{2}} \end{cases} \\ \Leftrightarrow\quad & 1 - \lambda_{d,k}^{-(d+1)} = \lambda_{d,k}^{-\frac{(d+2)}{2}} \\ \Leftrightarrow\quad & \lambda_{d,k}^{d+1} - \lambda_{d,k}^{\frac{d}{2}} - 1 = 0. \end{aligned}$$

We now plug the two expressions we have for p into the third equation and get

$$\begin{aligned} &\lambda_{d,k}^{-\frac{(d+2)}{2}} \times \lambda_{d,k}^{-(d+1)} = \lambda_{d,k}^{-(d+3)} \\ \Leftrightarrow\quad & \lambda_{d,k}^{-(d+3)} \times [1 - \lambda_{d,k}^{-\frac{d}{2}+1}] = 0 \\ \Leftrightarrow\quad & \lambda_{d,k} = 0 \text{ or } \lambda_{d,k}^{-\frac{d}{2}+1} = 1. \end{aligned}$$

Since $\lambda_{d,k} \in (1, 2)$ we are left with $\lambda_{d,k}^{-\frac{d}{2}+1} = 1$, which requires $-\frac{d}{2} + 1 = 0$ or $d = 2$. Consequently, the bit flipping algorithm produces an optimal code if and only if $d = 2$, $k = d + 2 = 4$, and $\lambda_{d,k}$ is a root of $z^{d+1} - z^{\frac{d}{2}} - 1$.

It remains to show that $\lambda_{d,k}^{d+1} - \lambda_{d,k}^{\frac{d}{2}} - 1 = 0$ in the $(2, 4)$ case, i.e., $\lambda_{2,4}^3 - \lambda_{2,4} - 1 = 0$. Recall that $\lambda_{d,k}$ is the largest real root of the characteristic polynomial $P_{d,k}(z)$, which for a finite k takes the form

$$P_{d,k}(z) = z^{k+1} - \sum_{j=0}^{k-d} z^j.$$

When $d = 2$ and $k = 4$ we can factor $P_{2,4}(z)$ and write it as

$$P_{2,4}(z) = z^5 - z^2 - z - 1 = (z^3 - z - 1) \times (z^2 + 1).$$

Since $\lambda_{d,k}$ is real, then it must be a root of $z^3 - z - 1$, which completes the proof. □

For the remaining suboptimal cases we can numerically optimize the rates of both algorithms. Table I shows optimal average rates for the bit stuffing and the bit flipping algorithms for a number of constraints. Also shown are the corresponding optimal biases (i.e., the probability of a 0), the capacity of each constraint, and the relative performance of the algorithms.

REFERENCES

[1] K. A. S. Immink, P. H. Siegel, and J. K. Wolf, “Codes for digital recorders,” *IEEE Trans. Inf. Theory*, vol. 44, no. 6, pp. 2260–2299, Oct. 1998.

[2] B. H. Marcus, P. H. Siegel, and J. K. Wolf, “Finite-state modulation codes for data storage,” *IEEE J. Select. Areas Commun.*, vol. 10, no. 1, pp. 5–37, Jan. 1992.

[3] C. E. Shannon, “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, Jul. 1948.

[4] P. Lee, “Combined error-correcting/modulation recording codes,” Ph.D. dissertation, Univ. California, San Diego, La Jolla, CA, 1988.

[5] P. E. Bender and J. K. Wolf, “A universal algorithm for generating optimal and nearly optimal run-length-limited, charge constrained binary sequences,” in *Proc. IEEE Int. Symp. Information Theory*, San Antonio, TX, Jan. 1993, p. 6.

[6] J. K. Wolf, “An information theoretic approach to bit stuffing for network protocols,” in *Proc. 3rd Asia-Europe Workshop on Information Theory*, Kamogawa, Japan, Jun. 2003, pp. 18–21.

[7] N. Abramson, *Information Theory and Coding*. New York: McGraw-Hill, 1963.

[8] C. B. Jones, “An efficient coding system for long source sequences,” *IEEE Trans. Inf. Theory*, vol. IT-27, no. 3, pp. 280–291, May 1981.

[9] P. E. Bender, “Redundancy re-organization for the magnetic channel,” Ph.D. dissertation, Univ. California, San Diego, La Jolla, CA, 1992.

[10] E. Zehavi and J. K. Wolf, “On runlength codes,” *IEEE Trans. Inf. Theory*, vol. 34, no. 1, pp. 45–54, Jan. 1988.