An Improvement to the Bit Stuffing Algorithm

Sharon Aviran, Paul H. Siegel, and Jack K. Wolf [1]
Department of Electrical and Computer Engineering
University of California, San Diego, USA
{saviran, psiegel, jwolf}@ucsd.edu

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. We propose a modification to the bit stuffing algorithm. We show analytically that the proposed algorithm achieves improved average rates over bit stuffing for most (d,k) constraints.**

I. Introduction

A binary sequence satisfies a runlength (d,k) constraint if the number of consecutive zeros is at most k and any two successive ones are separated by at least d zeros. Such sequences are called (d,k)-*sequences* and are commonly used in magnetic and optical recording. The *capacity* of a (d,k) constraint is defined as $C(d,k) = \lim_{n\to\infty} \frac{1}{n} \log_2 N_{d,k}(n)$, where $N_{d,k}(n)$ is the number of distinct (d,k)-sequences of length n. The capacity is known for all values of d and k.

The *bit stuffing algorithm* encodes (d,k)-sequences by inserting extra bits into an uncoded data stream [1]. The algorithm first converts the input sequence into a sequence having different statistical properties. It then inserts (*stuffs*) additional bits in a manner that guarantees that the resulting sequences satisfy the (d,k) constraint. It was shown in [1] that the maximum average bit stuffing rate equals capacity for all $(d,d{+}1)$ and (d,∞) constraints. It was further shown that the maximum rate is strictly less than capacity for all other cases. This leaves room for improvement whenever $d+2 \leq k < \infty$.

In this paper 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. The increase in complexity is minor. We name the proposed modification the *bit flipping algorithm*. We analyze the performance of both algorithms to obtain the following result:

Theorem 1 *Let $d \geq 1$ and $d+2 \leq k < \infty$. Then the bit flipping algorithm achieves a greater maximum average rate than the bit stuffing algorithm.*

II. The bit stuffing algorithm

The bit stuffing encoder consists of two components: a distribution transformer and a constrained encoder. The distribution transformer converts an *unbiased* independent and identically distributed binary input sequence into a *biased* sequence of independent random bits, whose probability of a 0 is some $p \in (0,1)$. This conversion can be implemented in a one-to-one manner. Hence we can apply the reverse transformation to recover the unbiased data. The constrained encoder 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 (*stuffs*) a 1 followed by d 0's. Whenever encountering a biased 1, the encoder inserts d 0's. To decode, one keeps track of the run length in the constrained sequence, identifying the stuffed bits and discarding them. The resulting sequence is then fed into the inverse distribution transformer, so as to obtain the original unbiased data.

Finally, we optimize p to find the maximum average rate for a given (d,k) constraint.

III. Improving bit stuffing by bit flipping

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 while modifying 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 or equal to l then flip the next biased bit before writing.

In other words, flip the biased bit starting from a run length of l. This is called *the bit flipping algorithm*.

Two questions arise. What is the optimal run length to start flipping? When can we improve the bit stuffing rate by flipping? Theorem 1, whose proof we now describe, answers these two questions.

We first derive an analytical expression for the average bit flipping rate. For $d \geq 1$ and $p > 0.5$, we show that $k-1$ is the optimal run length to start flipping. Moreover, the resulting bit flipping rate is strictly greater than the corresponding bit stuffing rate.

We next show that for all runlength constraints considered in Theorem 1, except for the case $(d,k)=(1,3)$, the optimal bit stuffing bias is indeed greater than 0.5. Thus, for these (d,k) pairs, bit flipping yields a higher rate than bit stuffing. In fact, by considering specific examples, we have seen that it may be possible to further improve the bit flipping rate by optimizing the bit flipping bias.

For the $(1,3)$ case, bit flipping is inferior to bit stuffing when the optimal bit stuffing bias is used. However, by numerically optimizing the bit flipping bias, we show that the maximum bit flipping rate exceeds the maximum bit stuffing rate, completing the proof of Theorem 1.

References

[1] P.E. Bender, J.K. Wolf, "A universal algorithm for generating optimal and nearly optimal run-length-limited, charge constrained binary sequences," *Proc. 1993 IEEE Int. Symp. Inform. Theory,* San Antonio, Texas (1993), p. 6.

[1]This research was supported in part by NSF Grant CCR-0219582, part of the Information Technology Research Program and by the Center for Magnetic Recording Research at UCSD.